UNIVERSIDAD DE NAVARRA

ESCUELA SUPERIOR DE INGENIEROS INDUSTRIALES

SAN SEBASTIÁN

COLISIONES EN ESTUDIOS DE MANTENIBILIDAD CON RESTITUCIÓN DE ESFUERZOS SOBRE MAQUETAS DIGITALES MASIVAS Y COMPACTAS

M E M O R I A
que para optar al Grado de Doctor
presenta

DIEGO BORRO YÁGÜEZ

San Sebastián, Junio 2003

Servicio de Publicaciones de la Universidad de Navarra

ISBN 84-8081-143-9

A mis padres Germán y Mercedes
y a mi hermana Beatriz.

AGRADECIMIENTOS

Al igual que una historia, esta Tesis ha tenido un principio, un final (espero) y una serie de sucesos en los que ha participado muchísima gente, directa o indirectamente, haciendo más llevadero todos estos años transcurridos. Por motivos de espacio o por olvido involuntario, no puedo mencionar a toda la gente que me ha ayudado estos años a seguir adelante. Mis disculpas de antemano a todos ellos y mi agradecimiento de todo corazón.

En primer lugar tengo que agradecer el inicio de esta aventura a mis dos directores, Luis Matey y Alejandro García-Alonso, por ofrecerme este trabajo y por la confianza depositada en mí. He tenido la suerte de trabajar en un proyecto impresionante y a la vez sacar esta Tesis y eso te lo debo a ti Luis. Aún y todo, este trabajo no podría haber llegado a buen puerto sin los consejos, ayudas y por qué no, tu eterna paciencia Alex.

Me gustaría agradecer también a Juan Tomás Celigüeta y Jordi Viñolas por darme la oportunidad de trabajar en el departamento de mecánica del CEIT.

Todo inicio empieza en algún lugar y el mío fue en el grupo de mecánica computacional del departamento, nuestro querido "agujero". Tengo que agradecer a toda la gente del grupo por su buena acogida. Algunos me acompañaron y ayudaron con sus aportaciones y otros me legaron valores humanos tanto o más importantes que el resto de ayudas. Por supuesto, el primer agradecimiento es para el *REVIMA Software Team*: Iker Aguinaga y Aiert Amundarain. Juntos (y con mucha paciencia) integramos este módulo autista en el sistema y oye, al final parece que el pequeño monstruo funciona. Iñaki Ayucar, que aunque te hayas ido al mundo empresarial estás siempre cerca ya sea para resolver cualquier duda gracias a tus conocimientos gráficos como para ir al cine. El día a día en las comidas se hacía más fácil gracias a Mikel Pérez, Aitor Rodríguez y Sergio Ausejo. También tengo que mencionar a los que ya no están por aquí pero que me dejaron huella con su personalidad como la de Javi González, Juanmi López, la humildad de Pablo Garoña o el "sentido del humor" de David Flecha; y a los que todavía están aquí compartiendo el mismo barco: Alberto Lozano, Aritz Ustarroz, Ángel Suescun, Iñaki Anso, Luis Unzueta, los Migueles (Seco y Castro), Josune Hernantes, Aitor Cazón y como no Iñigo Iparraguirre ¿eh? Un guiño también para mi

compañero Iñaki Aliaga y para la pareja enemiga Javier Pargada y Jonny Alberdi.

No me olvido de ti Leire Suárez, la "infiltrada", este párrafo está dedicado a ti. ¿Qué haría yo sin tu calidad y sin esos *coffees* de la mañana?

Después de este inicio, no me voy a olvidar de los que, por un traslado de laboratorio, me acogieron entre ellos como a uno más y han sido mis compañeros estos años. Me refiero a mis queridos robóticos: Joan Savall del que aprendo todo los días, Jaime Rubí el *crack*, Emilito Sánchez que tuviera la duda que tuviera siempre me la resolvía (con una respuesta pesimista pero satisfactoria), Jorge Juan Gil con el que espero coincidir en la entrega de la Tesis y como no, a la horma de mi zapato, al *jugón man* Javier Martín Amezaga ¿qué sería de mí sin esos ratillos hasta las tantas y esas charlas geométricas? Mi agradecimiento a todos ellos y al resto de robóticos que comparten o han compartido el laboratorio, los proyectos y los actos "sociales" como Asier Ibeas, Dorletilla Otaduy, Nuria Merino, Ana Sancho, Gorka Zorzano, Raúl Lara, Ángel Rubio, Dimas París, Ainhoa Cortés, José Carballo, Nahia del Valle, Ignacio Mansa, Pablo Callejo y los nuevos Mikels (Ares y Etxeberria).

Gracias también a Javier Atencia, Paul Bustamante y Javier Díaz por iniciarme en el gratificante mundo de la docencia y ofrecerme la oportunidad de colaborar en las asignaturas de Informática I e Informática II. Muchos frutos de esta Tesis se los debo a sus conocimientos acerca de Matlab y Maple.

Quiero agradecer al resto del departamento, a los Mekatronikos el buen ambiente siempre presente e imprescindible. Gracias por esas pachangas y actos de confraternización. Es inevitable mencionar también a dos personas importantísimas en el día a día: Manolo Rodríguez y Ana Leiza. Gracias por todo.

Un saludo a todas las personas de los demás departamentos, de la Escuela y al servicio de limpieza y mantenimiento. Ellos han colaborado en otros aspectos de la vida igual de importantes.

Finalmente, quiero dedicar estas líneas a todos mis amigos (por su paciencia) y sobre todo a mi familia. A mi padre que siempre me aconsejó esta decisión y de la que no me arrepiento, a mi madre porque siempre está a mi lado y a mi hermana que es la mejor. Ellos han sufrido conmigo esta Tesis y a ellos va dedicada.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Uno de los aspectos de mayor interés de la mantenibilidad concierne al análisis de accesibilidad del hombre y la herramienta, donde se acomete el cálculo de vías de acceso, de secuencias de ensamblaje y desensamblaje, y de evaluación de tiempos.

Hoy en día todavía es necesario el empleo de maquetas físicas, para poder evaluar la mantenibilidad de las partes externas del motor en la etapa de desarrollo. Aunque estas maquetas pueden ser empleadas en otras aplicaciones, la razón de que se construyan suele ser evaluar la mantenibilidad. Este trabajo busca que las maquetas físicas dejen de ser necesarias para los estudios de mantenibilidad, para lograr una reducción de costes y tiempo en el desarrollo de motores de avión. Este objetivo viene facilitado e impulsado por el siguiente hecho: actualmente, en la industria aeronáutica, el diseño basado en maquetas electrónicas está siendo empleado de forma habitual en la creación de los elementos externos del motor (tuberías, guarniciones e instalaciones).

La Realidad Virtual, combinada con el uso de hardware específico que emula sensaciones táctiles, puede ofrecer una solución al objetivo propuesto. En esa línea, el objetivo principal de esta Tesis, y también una tarea importante en la Realidad Virtual, es el estudio e investigación dentro del campo de las colisiones, en este caso para su aplicación en herramientas de mantenibilidad. Los resultados finales se traducirán en el desarrollo de un módulo de colisiones exacto y eficiente que ayude a la aplicación a predecir la mantenibilidad de un motor de avión, simulando las tareas necesarias para ello.

La extremada complejidad de los modelos CAD aeronáuticos requiere que el cálculo de las interferencias implemente técnicas específicas para asegurar un *frame rate* interactivo.

Además, y por las características de las tareas de mantenimiento, no sólo se necesita detectar las colisiones del entorno, sino que también hace falta calcular una respuesta, que es necesaria para ofrecer una salida táctil al usuario. Esto último es de gran importancia ya que condiciona el grado de sensación recibida en el momento de la manipulación de los objetos de la escena.

ABSTRACT

One of the most relevant aspects of maintainability concerns man and tool accessibility task analysis, which is undertaken in order to calculate paths and assembly-disassembly sequences, and time analyses.

Nowadays the use of a physical mock-up at the developmental stage is mandatory in the evaluation of engine maintainability. Although these mock-ups can be used in other applications, the main reason for building them are maintainability tests. This work seeks to avoid building them, in order to reduce new aircraft engines development time and costs. This aim is encouraged by the following fact: currently, in the aeronautics industry, design based on electronic mock-up is widely used in the creation of engine externals (piping, harnesses and installations).

Virtual Reality combined with haptic devices, which are able to offer tactile feeling, can provide a solution the goal we seek. In this line of research, the main aim of this work, and also an important task within Virtual Reality, is the study and research of the collision problem, in this case taking into account maintainability applications. The final results will serve to develop an accurate and efficient collision module, which will permit to accomplish maintainability tasks for aircraft engines.

The complexity of aeronautic CAD models demands that the interference computation has an interactive frame rate thanks to advanced techniques.

Besides collision detection, virtual maintainability requires the calculation of a force response, which is necessary to offer a comfortable tactile output to the user. An adequate response is of outmost importance because comfort feelings heavily rely in proper responses.

CAPÍTULO 0

INTRODUCCIÓN

0.1 PLANTEAMIENTO DEL PROBLEMA

La Tesis que se presenta a continuación es resultado del estudio del problema de la detección de colisiones y otros problemas asociados, en concreto en aplicaciones orientadas al mantenimiento y ensamblaje de motores aeronáuticos.

En el campo de la aeronáutica el término mantenibilidad se define de la siguiente forma: "la habilidad de un elemento de mantenerse en servicio o volver al estado adecuado para que desarrolle su función, después de haber sido mantenido en condiciones previamente establecidas, empleando el personal, los medios y los procedimientos adecuados" (Blanchard et al. 1995).

La incorporación de los gráficos por computador y su rápido avance gracias al abaratamiento del hardware, permiten representar visualmente los modelos CAD (*Computer Aided Design*) de los motores aeronáuticos para su posible interacción con el usuario consiguiendo representar virtualmente la realidad sin necesidad de costes de fabricación (Figura 0.1).

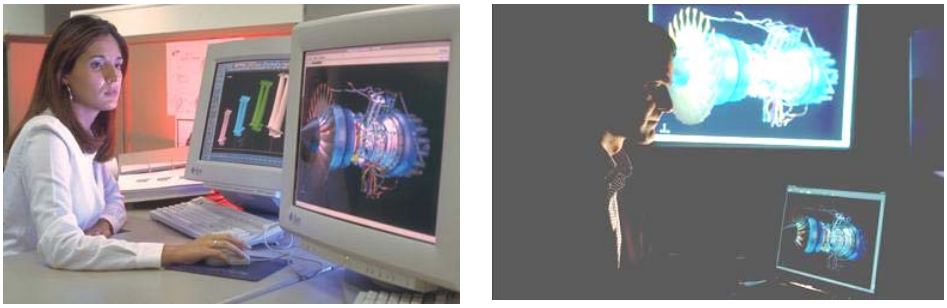


Figura 0.1: Usuarios utilizando programas CAD.

Uno de los aspectos de mayor interés de la mantenibilidad concierne al análisis de accesibilidad del hombre y la herramienta, donde se acomete el cálculo de vías de acceso, de secuencias de ensamblaje y desensamblaje y de tiempos. El diseño basado en maquetas electrónicas esta siendo empleado de forma habitual en la creación de las partes externas del motor (tuberías, guarniciones e instalaciones) en la industria aeronáutica.

A la hora de validar un diseño CAD de motor, se construyen maquetas de madera totalmente idénticas, en escala y en detalle, al motor real. Estas maquetas también sirven para el entrenamiento de los operarios en las tareas de mantenimiento y ensamblaje propias de estos motores. Durante esta etapa son frecuentes los cambios de diseño, bien por errores en el diseño o bien porque los operarios detectan emplazamientos poco accesibles, lo que origina nuevos costes de diseño y de fabricación de las nuevas maquetas de madera. La necesidad de mejorar el proceso de diseño fomenta el desarrollo de aplicaciones virtuales que manejen únicamente un modelo virtual y permitan simular esas tareas de mantenimiento. Esta es la base del prototipado virtual (VP).

Una vez creado y validado el modelo CAD, tiene lugar la fase de fabricación del motor real con los que se realizarán las tareas de mantenimiento por parte de los operarios entre los distintos vuelos (Figura 0.2). Sin embargo, antes de realizar estas tareas sobre los motores reales, es necesario un periodo de aprendizaje por parte de los operarios para evitar o minimizar los posibles errores. El aprendizaje se realiza sobre las maquetas de madera pero esta formación también puede ser hecha sobre los modelos virtuales de los motores.

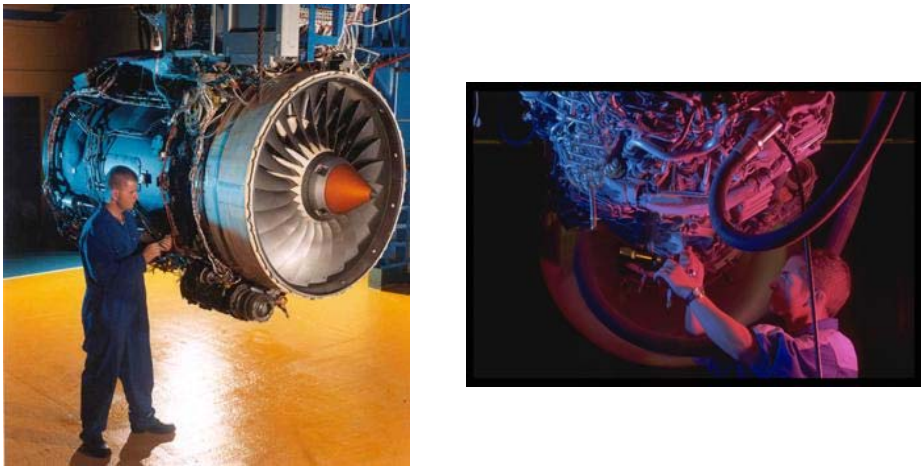


Figura 0.2: Operarios realizando tareas de mantenimiento sobre motores de aviación.

En las aplicaciones gráficas, para representar los objetos, se utilizan diferentes modelos o representaciones geométricas. Esta representación permite la visualización realista, en la posición y orientación deseada, de toda la escena gráfica. El problema de la realidad virtual (RV) es que los objetos, al no ser reales, carecen también de las propiedades físicas inherentes a los objetos reales como es la imposibilidad de que dos sólidos ocupen un mismo espacio en el mismo instante de tiempo.

Por todo ello, en las aplicaciones virtuales se necesita un módulo de Colisiones que detecte e impida la penetración de distintos sólidos entre sí y de esta forma poder realizar una simulación realista de las tareas.

Este tipo de aplicaciones generalmente no están integradas junto con las encargadas de diseñar los modelos. Sin embargo, siempre se tiene la posibilidad de compartir la información de una aplicación CAD mediante la exportación/importación de los modelos geométricos.

Una de las representaciones geométricas más usada en este intercambio de información es la representación mediante facetas planas. Ya que en CAD se diseña con un detalle muy elevado y estos motores constan de miles de piezas, los modelos manejados por el módulo de Colisiones pueden alcanzar los millones de triángulos. Esto deriva en una complejidad añadida desde el punto de vista computacional. El algoritmo directo o de fuerza bruta (todas las posibles combinaciones de triángulos entre los objetos) es inviable. Un solo cálculo, en un instante de tiempo dado, requeriría horas de computación.

Por otro lado, el componente clave de estos entornos es la habilidad de percibir y manipular interactivamente los objetos virtuales. La dificultad añadida que poseen los sistemas de simulación de mantenibilidad, y que los diferencian de los sistemas simples de visualización, es la interacción con la escena virtual impidiendo movimientos que produzcan interferencias geométricas. Hace falta una interfaz hardware que sea capaz de restituir al usuario una fuerza que simule los contactos con el entorno. Estos dispositivos son llamados hápticos y necesitan que se calcule la fuerza a restituir con la información geométrica que se posea en ese instante de tiempo.

Gracias a la unión de este hardware y su control asociado junto con el módulo de Visualización y el de Colisiones se puede lograr tener un sistema de simulación realista de tareas de mantenimiento en motores aeronáuticos en este caso, pero escalable a otros modelos de maquinaria.

0.2 ANTECEDENTES

El objetivo de la detección de colisiones (también llamado detección de interferencias o intersecciones) es informar automáticamente cuándo se detecta algún contacto geométrico evitando que dos sólidos ocupen el mismo espacio, algo que ocurre por sí mismo en el mundo real pero en el virtual se necesita calcular.

El problema de detectar la colisión entre dos objetos, calcular exactamente los puntos de contacto y con ello determinar una apropiada respuesta física, significa un gran coste computacional. Por ello en casi todas las aplicaciones en las que existe un módulo de Colisiones, éste se convierte en el cuello de botella del sistema convirtiendo a este problema en uno de los más estudiados dentro del campo de la geometría computacional.

El prototipado virtual, dentro del llamado *digital mock-up* (DMU), está siendo más y más importante a la hora de reducir tiempos de fabricación y por consiguiente los costes asociados a un nuevo modelo o producto.

Muchas compañías, especialmente industrias del mundo automovilístico y aeronáutico, han empezado a evaluar la RV como un back-end del CAD y CAE para investigar un DMU de un nuevo diseño. La idea es permitir diseñar, analizar y evaluar aspectos del nuevo producto interactiva e inmersivamente.

Este tipo de aplicaciones que combina la RV, la geometría computacional y las interfaces hápticas han aparecido recientemente en la bibliografía.

0.3 OBJETIVOS DE LA TESIS

El principal objetivo de esta Tesis consiste en el estudio y desarrollo de diferentes métodos del cálculo de colisiones orientados a su aplicación dentro de sistemas de mantenibilidad mediante la realidad virtual. Para ello es necesario informar y responder automáticamente cuándo se detecta algún contacto geométrico, evitando así que dos sólidos intersecten. A continuación se describe con más detalle los diferentes objetivos.

0.3.1 Modelo geométrico

Para la resolución del problema, el presente trabajo usa como información geométrica el modelo de representación B-Rep o de facetas planas. La representación de estos modelos consiste en listas de triángulos (u otros polígonos) que delimitan la superficie frontera de un objeto válido.

Aunque los métodos propuestos sirvan para todo tipo de tamaño de los modelos, dentro del mundo aeronáutico se necesita una precisión muy elevada. Para este nivel de detalle, los modelos virtuales de los motores contienen del orden de millones de polígonos (triángulos en este caso) por lo que los algoritmos desarrollados deben estar optimizados para el uso de modelos masivos y relativamente compactos.

0.3.2 Precisión de cálculo

El nivel de precisión del cálculo de colisiones no tiene porque ser igual al de diseño. Algunos métodos de la bibliografía sólo necesitan llegar a una aproximación del resultado y no analizan fronteras.

Sin embargo, en aplicaciones de simulación de mantenibilidad en las que se necesita conocer exactamente la accesibilidad a diferentes elementos del modelo, la precisión en el cálculo de colisiones tiene que ser lo más próxima posible a la del diseño (el modelo CAD). Evidentemente, no es posible realizar cálculos geométricos con más precisión que la ofrecida por la representación utilizada (el modelo poligonal exportado del sistema CAD).

0.3.3 Restricciones de la geometría

Existen muchos trabajos en la detección de colisiones en los que se introducen ciertas restricciones a la hora de definir los objetos de la escena, como por ejemplo restringirse al uso de objetos convexos. Teniendo en cuenta esta clase de restricciones, se pueden implementar métodos de detección de colisiones más rápidos y sencillos.

Si se quiere detectar las interferencias entre distintos objetos de un motor aeronáutico, no se puede mantener esta restricción ya que se limitaría el tipo de objetos a usar en el diseño. Por lo tanto se tiene que contemplar la posibilidad del uso tanto de objetos convexos como cóncavos.

Esta tesis se centra en el estudio de las colisiones sobre cuerpos rígidos sin entrar en la problemática de los cuerpos deformables.

0.3.4 Restricción del tiempo

Al contrario que muchas aplicaciones no orientadas al tiempo real y que realizan el cálculo de colisiones *off-line* (*Path Planning* en robótica, animación de simulaciones físicas), en simulación interactiva mediante la RV, se necesita detectar colisiones y calcular una respuesta de colisión *on-line*, es decir, en tiempo real para asegurar el efecto de interacción.

Hablar de tiempo real es un concepto muy amplio y depende de a qué tipo de aplicación se vaya a aplicar. No es lo mismo el tiempo real de una central nuclear que el tiempo real en una simulación. En esta Tesis se considera tiempo real un grado lo suficientemente razonable de interactividad con el usuario. Este valor se fijó en no menos de 10 frames por segundo por lo que había que lograr que el cálculo de colisiones se realice en menos de 100 milisegundos. Existen trabajos previos reafirmando que el cómputo de un cálculo en 100 milisegundos o menos se puede considerar tiempo real (Card et al. 1983).

0.3.5 Interacción con el usuario

El módulo de Colisiones tiene que ser sensible a requerimientos que el usuario pueda imponer durante la simulación. En las operaciones de ensamblaje es posible desmontar objetos aislados u objetos unidos a otros mediante restricciones impuestas. El módulo de Colisiones debe ser capaz de ofrecer al usuario la capacidad de añadir o eliminar pares de objetos al cálculo de las colisiones. Estas operaciones deben ser además poco costosas ya que se realizarán en mitad de una simulación interactiva, al igual que los estudios de accesibilidad que el usuario pueda pedir al sistema a la hora de comprobar posibles interferencias entre la herramienta y el entorno.

0.3.6 Respuesta de colisión

El módulo de Colisiones del sistema global tiene que ser capaz de detectar todas las posibles interferencias entre los distintos objetos de la escena virtual. Pero también es necesario transmitir al usuario una sensación de contacto cuándo

éste colisione con el entorno. Para lograrlo hay que calcular una respuesta de colisión que será específica para el dominio de la aplicación (O'Sullivan et al. 2001). Esta respuesta podría ir desde la destrucción de los objetos en un juego hasta el cálculo de las posteriores trayectorias una vez ocurrida la colisión como puede ser en una simulación física. En el caso del uso de interfaces de restitución de esfuerzos, la respuesta vendrá dada por el cálculo de una fuerza lo más realista posible.

0.3.7 Recursos del sistema

Hace falta llegar a un compromiso entre la velocidad de cálculo y los recursos utilizados como puede ser la memoria del computador. Los modelos masivos utilizados requieren grandes cantidades de memoria tanto para el módulo de colisiones como para el de visualización por lo que ninguno de estos módulos puede utilizar toda la memoria que considere oportuna. Hace falta optimizar la estructura de datos almacenada para que ocupe el menor espacio pero guardando la mayor información posible.

0.4 CONTENIDO DE LA MEMORIA

El contenido de la Tesis está distribuido en 7 capítulos. A continuación se resume el contenido de cada uno de ellos.

El primer capítulo presenta una serie de clasificaciones útiles en el marco de la detección de colisiones. Primero se resume brevemente los modelos de representación geométrica más comunes, después se analizan los distintos tipos de problemas y las características de cada uno de ellos, y para finalizar se muestra una clasificación de los métodos existentes en la bibliografía.

En el segundo capítulo se analiza y compara la utilización de distintas representaciones para afrontar el problema de la detección de colisiones. Dentro de los métodos vistos en el capítulo anterior, se han analizado aquellos que se han considerado relevantes para abordar los objetivos de la Tesis. En este capítulo se razona la elección de uno de los métodos como base para el método que se ha adoptado finalmente.

El tercer capítulo presenta la descripción completa del método propuesto, analizando cada una de sus fases evaluando los dos parámetros a optimizar: el rendimiento y la memoria consumida.

Un problema muy conocido en la bibliografía al hablar de las técnicas de enumeración espacial es la elección del tamaño de celda que se va a usar. Este

tamaño se desconoce a priori y en el capítulo 4 se presenta una solución analítica para calcular una aproximación al tamaño óptimo de celda.

El capítulo 5 se centra en el problema de la respuesta de colisión. Primero se exponen los antecedentes en este campo. Posteriormente se pasa a detallar todos los algoritmos utilizados tanto por el módulo de Colisiones como por el módulo de Control para calcular los parámetros necesarios de la fuerza a restituir por el dispositivo háptico.

Todo el trabajo de esta Tesis está integrado dentro de un sistema multidisciplinar que permite realizar tareas de mantenimiento y ensamblaje en motores aeronáuticos virtuales. Tanto la arquitectura del sistema como la descripción y comunicación de sus módulos está descrita en el capítulo 6. Asimismo, también se presentan un conjunto de resultados experimentales para comprobar el buen comportamiento del método.

Finalmente, en el capítulo 7 se resumen las conclusiones derivadas tras finalizar el trabajo de la presente Tesis y las posibles futuras líneas de investigación.

0.5 REFERENCIAS

Blanchard, B., Verma, S.D., y Peterson, E.L., "*Maintainability*", John Wiley & Sons Inc., New York, USA, 1995.

Card, S.K., Moran, T.P. y Newell, A., "*The Psychology of Human-Computer Interaction*", Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

O'Sullivan, C., Dingliana, D., Ganovelli, F., y Bradshaw, G., "Collision Handling for Virtual Environments" Proceedings of Eurographics Tutorial, 2001.

CAPÍTULO 1

ANTECEDENTES Y TAXONOMÍAS

1.1 INTRODUCCIÓN

En este capítulo se presenta una serie de clasificaciones con el propósito de ilustrar sobre la diversidad que posee el problema de las colisiones. En general, cada tipo de problema o aplicación demanda distintos requerimientos: precisión, rapidez en el cálculo, etc. Además, también se pueden utilizar diferentes modelos geométricos para la representación de los objetos. Por lo tanto se hace necesario considerar varias clasificaciones que ayuden a contemplar globalmente el conjunto de posibilidades ofrecidas.

Muchos de los trabajos mencionados en este capítulo no contemplan todos los objetivos impuestos en esta Tesis (grado de precisión, geometría de los objetos, etc.) pero son muy útiles como punto de partida y el método planteado tiene en cuenta varios de esos trabajos.

Todas las clasificaciones se centran en las colisiones, menos la primera en la que se exponen los modelos de representación geométrica más relevantes. La segunda clasificación se centra en el estudio detallado de los diferentes tipos de problemas, según sus características, con los que se puede enfrentar el cálculo de colisiones. Esta clasificación sólo pretende mostrar los problemas, no las soluciones. Para estas, posteriormente se propone la tercera clasificación en la que se estudian los diferentes métodos que existen en la detección de colisiones.

Tanto en la segunda como en la tercera clasificación, se han considerado aquellos problemas y métodos que puedan tener alguna relación con el problema de colisiones planteado en esta Tesis.

1.2 CLASIFICACIÓN DE LOS MODELOS GEOMÉTRICOS

Como ya se ha dicho, esta primera clasificación pretende exponer de forma muy breve los diferentes tipos de representación de sólidos que existen. Foley et al. (1990), siguiendo el clásico trabajo de Requicha (1980), clasificaban estas representaciones en cinco grupos: *Instanciación de primitivas*, *Representaciones de Barrido*, *Representación de superficies frontera*, *Representaciones de subdivisión espacial* y *Representaciones tipo CSG*. Los métodos para resolver el problema de las colisiones vendrán condicionados en gran medida por la decisión del tipo de representación. A continuación se describen brevemente cada uno de estos grupos.

En la *instanciación de primitivas* se define un conjunto de las formas 3D que más frecuentemente van a ser utilizadas. A partir de dichas formas y de los parámetros introducidos por el usuario, se pueden generar entidades específicas.

En las *representaciones de barrido*, el objeto se representa especificando una entidad geométrica, ya sea bidimensional o tridimensional, y definiendo un recorrido que mueve esa entidad a través del espacio. El volumen creado en esa trayectoria define el objeto. Este tipo de representaciones son útiles para construir objetos tridimensionales que tienen simetría de traslación, rotación y de otro tipo.

Como ya se ha comentado, el modelo geométrico que se usa en esta Tesis es el B-Rep o *representación frontera* mediante facetas planas. Es además el más usado en la bibliografía de la detección de colisiones por la diversidad de algoritmos existentes. En este tipo de representación, el objeto es definido por las superficies que delimitan el volumen del objeto. La B-Rep basada en caras planas es una representación muy frecuente para el intercambio de información entre diferentes aplicaciones de CAD debido a la gran cantidad de formatos que soportan esta representación. Otra ventaja es el hecho de que la mayoría de las tarjetas gráficas ofrecen aceleraciones hardware si se utilizan polígonos como representación geométrica.

Las *representaciones de subdivisión espacial* descomponen o subdividen el sólido en una colección de entidades geométricas mucho más simples como pueden ser cubos, paralelepípedos, etc. Las representaciones más usadas dentro de este grupo pertenecen a los voxels y a los octrees. En los primeros, el espacio

es particionado uniformemente mientras que en los segundos la subdivisión se hace jerárquicamente.

La geometría sólida constructiva (CSG, *Constructive Solid Geometry*) determina un volumen al aplicar operaciones de conjuntos como la unión, intersección o diferencia en dos volúmenes específicos. La totalidad de las operaciones se va representando en un árbol binario cuyos nodos no hoja son los operadores de conjuntos y los nodos hoja corresponden con primitivas de geometría simple.

1.3 CLASIFICACIÓN DE PROBLEMAS

Existen numerosos trabajos sobre el problema de la detección de colisiones. Ya que la detección de colisiones aparece en una amplia y diversa gama de situaciones, han aparecido multitud de problemas distintos y cada uno con su propio conjunto de técnicas para resolverlo. Hay que destacar trabajos anteriores en los que se aportan taxonomías de distintos campos y problemas de la detección de colisiones (Hubbard 1996, Lin et al. 1996, Lin y Gottschalk 1998, Jiménez et al. 2001 y O'Sullivan et al. 2001).

Esta revisión pretende englobar información dispersa por la bibliografía y clasificar los tipos de problemas más comunes teniendo en cuenta las diferentes características de cada uno de ellos. Los factores que se detallan a continuación son requerimientos impuestos por las aplicaciones y condicionarán tanto las representaciones de los objetos como los algoritmos utilizados por los métodos de resolución.

1.3.1 Campos de aplicación

Los campos más comunes en los que el cálculo de colisiones toma una importancia relevante son la robótica, los gráficos por computador y los entornos de simulaciones físicas; todos ellos compartiendo el mismo y clásico problema de geometría computacional: encontrar las intersecciones entre un conjunto de objetos.

La robótica ha estudiado este problema ampliamente. Por ejemplo en el ámbito del *Path Planning*, es decir, dado un robot, o un conjunto de elementos móviles de ese robot, y dada la trayectoria de ese robot, decidir si ocurrirá alguna colisión con su entorno. Este tipo de problemas se puede resolver en un pre-proceso, con lo que se estaría ante sistemas sin tiempo real en los que el cálculo de colisiones puede ser *off-line* (Hayward 1986, Latombe 1991 y Cameron y Qin 1997). Pero con el desarrollo de robots más modernos y autónomos se necesita que estos puedan operar en entornos desconocidos o con

movimientos no predeterminados, por lo que el cálculo de colisiones debe ser *on-line* (Shaffer y Herb 1992). En este último trabajo, además de proponer un método para un sistema de tiempo real en condiciones extremas, se presenta también un buen estudio del arte de los diferentes trabajos y soluciones al problema de las colisiones dentro del campo de la robótica.

Dentro de los gráficos por computador existe una amplia variedad de aplicaciones como son las orientadas a CAD/CAM, Realidad Virtual y animación.

En el área de *Computer-Aided Design and Manufactured* (CAD/CAM) la búsqueda de intersecciones es usada para refinar y corregir errores en los diseños realizados con sistemas CAD. De esta forma se consigue reducir costos sin tener que recurrir a la producción de prototipos físicos (Rossignac et al. 1992).

La RV pretende el uso de computadores para simular entornos físicos de tal forma que los humanos puedan visualizar, explorar e interactuar con los objetos del entorno virtual (Cohen et al. 1995, Wilson et al. 1999).

En la RV, y muy ligadas al CAD/CAM, se encuentran todas las aplicaciones destinadas al prototipado virtual. Se utiliza la RV para analizar, evaluar y validar diseños creados en CAD con la idea de ahorrar la producción de prototipos físicos. Se pueden tener aplicaciones "simples" parecidas a los sistemas de CAD pero que permiten tomar todo tipo de medidas y realizar análisis de interferencias entre aquellos objetos que se desea analizar (Zachmann 1997). También existen aplicaciones que aplican la RV al prototipado virtual para verificar ensamblajes y realizar procesos de mantenimiento (Gomes de Sá y Zachmann 1999). Aunque añadiendo más dificultad (y realismo) se podría calcular una respuesta de colisión adecuada para ser utilizada con algún dispositivo de reflexión de fuerza. De esta forma se consigue que la aplicación de RV parezca más realista (Chen 1999 y McNeely et al. 1999). A la plataforma que integra todas las herramientas y métodos digitales para el desarrollo, diseño y fabricación del modelo se le llama *Digital Mock-Up* (DMU).

Dentro de este tipo de aplicaciones también se encuentran los juegos modernos en 3D en los que las colisiones pueden llegar a tener un papel muy importante y presentan un reto al intentar conseguir el tiempo real (Blow 1997).

La desventaja de la RV con respecto al mundo real es que los objetos virtuales no tienen características físicas como los objetos reales, por lo tanto hay que chequear constantemente si los diferentes objetos de la escena colisionan entre sí.

Las aplicaciones de animación también se consideran dentro de los gráficos por computador. Una sistema de animación no necesita realmente una precisión absoluta como puede requerir una simulación física. Puede ser suficiente que los objetos se comporten y muevan de forma realista. Un tipo de aplicación muy común con estas características es la simulación de "telas" como puede ser ropa, paños, etc. (Volino y Magnenat-Thalmann 1997 y Zhang y Yuen 2002) donde se realizan animaciones de modelos humanos para probar con ellos diferentes tipos de vestiduras.

En el campo de la simulación física, se puede tener desde la de mecanismos (García-Alonso et al. 1994) hasta simulaciones de choques de automóviles pasando por cualquier aplicación de análisis de ingeniería que se pueda imaginar. En cualquiera de estas aplicaciones la trayectoria de los objetos móviles no viene impuesta por el usuario sino que está sujeta a restricciones físicas gobernadas por las leyes de la dinámica. Se puede calcular una apropiada acción para cada objeto (respuesta de colisión) basada en los puntos de contacto junto con información de los objetos como son sus masas, posiciones, velocidades, etc. En muchos casos, las aplicaciones de simulación mecánica se ayudan de módulos de visualización gráfica por computador que sirven de gran ayuda para interpretar el comportamiento del sistema (Barriuso 1998).

1.3.2 Tipo de pregunta

Además del modelo geométrico elegido, otro de los grandes condicionantes a la hora de elegir un algoritmo de detección de colisiones es el tipo de pregunta a la que se quiere buscar respuesta (Lin y Gottschalk 1998). Se pueden encontrar cuatro tipos de preguntas: *colisión*, *intersección*, *distancia* y *predicción*.

El caso más básico es el primero y consiste en determinar únicamente si dos modelos están o no en *colisión*. Muchas aplicaciones necesitan de este tipo de pregunta, sobre todo las orientadas a CAD/CAM. En el prototipado virtual se tienen modelos de piezas mecánicas y se quiere saber si los movimientos y operaciones que un técnico realiza con determinadas piezas pueden alcanzar a otras partes del modelo (Zachmann 1997).

A veces sin embargo, interesa encontrar la *intersección* exacta entre dos modelos en contacto. Este tipo de preguntas es importante para los sistemas de simulación física y animación dónde se necesita conocer exactamente todos los contactos para poder calcular una respuesta apropiada de colisión (Moore y Wilhelms 1988 y Hahn 1988). Dentro de este tipo de pregunta también entra el tipo de aplicación enfocada en esta Tesis. Aunque en una simulación de tareas de mantenimiento puede interesar únicamente saber si la herramienta está en colisión con alguna zona del modelo de motor (primer tipo de pregunta), es

necesario conocer todas las intersecciones si se quiere calcular una fuerza de restitución al usuario (McNeely et al. 1999).

Un tipo de pregunta típico en el campo de la robótica es el del cálculo de la *distancia*. Para realizar el *Path Planning* es necesario conocer la distancia del robot a los diferentes obstáculos de su entorno. Controlando esa distancia se puede evitar las colisiones del robot. Algunos autores, como Cameron y Culley (1986), han modificado el algoritmo del cálculo de la distancia para calcular también la penetración entre dos objetos que definen como la mínima distancia de traslación requerida para separarlos. En las aplicaciones de simulación mecánica también es usada este tipo de pregunta como ayuda al ingeniero de diseño para calcular por ejemplo distancias entre piezas de sistemas mecánicos (Barriuso 1998).

El último tipo de pregunta corresponde a la *predicción* de las colisiones. Si se conoce la posición y el movimiento de los objetos entonces se puede calcular o predecir su próxima colisión. Esta predicción permite controlar el *time step* en una simulación de, por ejemplo, el movimiento de un robot (Lin 1993). El robot debe juzgar si habrá colisión con obstáculos si mantiene el movimiento actual hasta el siguiente *time step* en el que volverá a analizar una nueva posición.

1.3.3 Número de objetos en el sistema

En un principio el número de objetos no tiene límite. Se puede tener aplicaciones que manejan decenas de objetos (García-Alonso et al. 1994), hasta los sistemas con centenares o miles de objetos (Cohen et al. 1995, Wilson et al. 1999).

Dependiendo de cuántos modelos u objetos se encuentren en el sistema se estará ante problemas de detección de colisiones distintos. Pero también existe diferencia si esos objetos están en movimiento o permanecen estáticos. Esta diferencia en el movimiento se tratará posteriormente.

Si el número de objetos a testear, *n*, es mayor que dos se estará ante el problema llamado *n*-body o multi-body (Selim y Almohamad 1999). Existen muchas aplicaciones que tratan con este tipo de problemas como las de simulación dinámica (Mirtich 2000) o simulación de mecanismos (García-Alonso 1990).

El problema entre un par de objetos suele ocurrir en sistemas de un objeto móvil y varios objetos estáticos, frecuentes en aplicaciones de prototipado virtual o de mantenibilidad. Los objetos estáticos al no tener movimiento se consideran como un único objeto global y estático (McNeely et al. 1999). Sin embargo, existen trabajos que, aunque manejan miles de objetos estáticos y un

único objeto móvil, siguen considerando cada objeto independientemente (Wilson et al. 1999). Aunque existan *n* objetos en el sistema, para no confundirlo con el *n*-body (*n* objetos en movimiento) a este tipo de aplicaciones las llamaremos ***n-body estáticas***.

El problema de un par de objetos es un subconjunto del *n*-body. En éste, hay que detectar la posible colisión entre los *n* objetos, lo que significa $\binom{n}{2}$ posibles pares. Como $\binom{n}{2} = n(n-1)/2$, este algoritmo es cuadrático $O(n^2)$ al aumentar el número de objetos en la escena. Hubbard (1993) clasifica primero como *broad phase* el problema *n*-body y a los algoritmos encargados de detectar la colisión entre un par de objetos los coloca en la *narrow phase*.

El problema *n*-body es de naturaleza combinatoria. Existen multitud de técnicas, sobre todo basadas en la partición espacial o volúmenes contenedores, para lograr descartar rápidamente pares de objetos y lograr algoritmos de orden logarítmico $O(n \log n)$ o incluso lineales $O(n)$ (Cohen et al. 1995). Sea cual sea el método utilizado, en todos los algoritmos siempre se podría dar el caso peor de $O(n^2)$ (Lawlor y Kalé 2002), pero estos casos no son nada corrientes y aparecen únicamente en situaciones y configuraciones geométricas muy especiales. Por ejemplo en la Figura 1.1 los *n* segmentos de línea se disponen de tal manera que existe $\binom{n}{2}$ colisiones entre ellas.

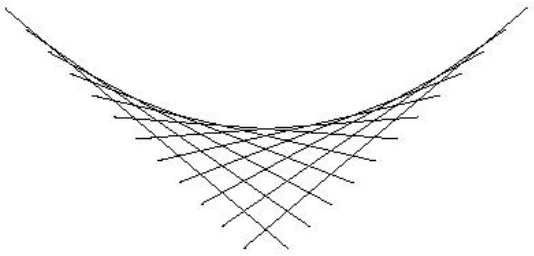


Figura 1.1: Configuración geométrica especial con $\binom{n}{2}$ pares de objetos en colisión (Lawlor y Kalé 2002).

1.3.4 Problema estático vs. Problema dinámico

Para el caso estático el cálculo de colisiones resuelve el problema en una posición y en un instante dado de los objetos dentro de la escena. En el segundo caso se resuelve para la trayectoria que describen los objetos a lo largo del tiempo.

Algunos autores llaman problema pseudo-dinámico a una aproximación del problema dinámico constituida por una sucesión de problemas estáticos (Held et al. 1995). Al tener un planteamiento más sencillo, permite separar la detección de colisiones de las trayectorias de los objetos.

La diferencia entre el pseudo-dinámico y el estático es que en el primero se puede tener información del movimiento o de instantes anteriores para acelerar el cálculo del presente instante. Estos algoritmos se dice que explotan la coherencia temporal.

Existe un problema posible: no detectar la colisión entre dos objetos que se produzca entre dos instantes de tiempo consecutivos. Esto se puede apreciar en la Figura 1.2.

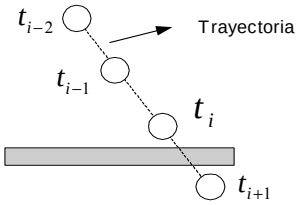


Figura 1.2: Ejemplo de la detección de no colisión en una sucesión de intervalos discretos de tiempo.

El intervalo de discretización del tiempo depende del tipo de aplicación y de la arquitectura seguida. Si el intervalo es demasiado grande ocurre lo mostrado en la Figura 1.2; y si es demasiado fino el coste computacional puede ser demasiado elevado. Jiménez et al. (2001) muestran también algunos métodos con un cálculo adaptativo del intervalo.

Una solución muy común es asumir que el objeto va a sufrir cambios en su desplazamiento muy pequeños en cada frame (Smith et al. 1995). De esta forma se detecta a tiempo la colisión. Para objetos muy rápidos y con grandes desplazamientos se puede usar volúmenes contenedores expandidos en el tiempo para solucionar el problema. En la Figura 1.3 se muestran estas dos posibles soluciones.

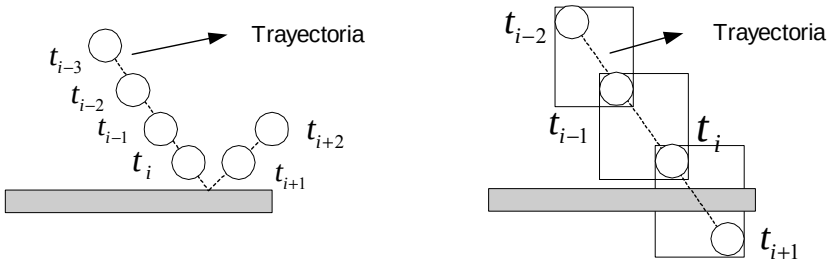


Figura 1.3: Asumiendo un desplazamiento muy pequeño (a) se puede detectar la colisión en dos momentos discretos de tiempo. Con desplazamientos grandes, la técnica del volumen contenedor expandido en el tiempo soluciona el problema (b).

En las aplicaciones de realidad virtual en las que se usan dispositivos de restitución de fuerzas, como pueden ser las simulaciones de tareas de mantenimiento, el problema es complejo al tener diferentes periodos de muestreo: por un lado el de visualización y por otro el del controlador de la interfaz háptica.

La combinación de este tipo de interfaces con las aplicaciones de realidad virtual no es evidente ya que estos dispositivos requieren una frecuencia de muestreo de 1kHz comparado con los 20-30 refrescos por segundo que necesitan las aplicaciones gráficas. Si el cálculo de colisiones se ajustara al de visualización podría suceder el caso mostrado en la Figura 1.2. Una mejora a este problema es la de separar en procesadores diferentes las tareas de visualización y las del cálculo de colisiones (Mark et al. 1996, Thompson II et al. 1997 y Amundarain et al. 2002). De este modo el intervalo de discretización del tiempo vendrá impuesto por la velocidad de cálculo del propio módulo de colisiones.

Los algoritmos pseudo-dinámicos son los más usados para la resolución del problema dinámico; pero también existen otros métodos, como por ejemplo realizar el cálculo a lo largo de un intervalo de tiempo (Aliyu y Al-Sultan 1999). Un estudio detallado de esta clase de algoritmos es ofrecido en Rozas (1998) y en Jiménez et al. (2001).

1.3.5 Grado de precisión

Algunas aplicaciones requieren más precisión en el cálculo que otras y el equilibrio entre precisión y rapidez se debe tener en cuenta a la hora de resolver el problema de las colisiones. Por ejemplo en robótica lo que importa es dar una respuesta en tiempo real que logre evitar accidentes con el robot (Shaffer y Herb

1992). En estos casos el parámetro que impera es la seguridad frente a la precisión.

A veces la precisión en el cálculo de colisiones dependerá del tiempo que se disponga para calcular esas colisiones. Se puede permitir un cálculo sin llegar al nivel máximo de precisión si el sistema necesita dar una respuesta y se ha sobrepasado el tiempo máximo del que se dispone para el módulo de colisiones (Hubbard 1996). Esta clase de algoritmos son llamados algoritmos con interrupción (O'Sullivan et al. 2001) o de tiempo crítico (Hubbard 1993, Hubbard 1995a, Hubbard 1995b, Hubbard 1996). Su ventaja es que la aplicación tiene un total control sobre el tiempo que puede estar dedicando al cálculo de las colisiones. En cuanto se sobrepasa el tiempo máximo se devuelve una respuesta de colisión y se consigue mantener las condiciones impuestas de tiempo real y *frame rate* constante (Zachmann 1997).

El inconveniente de agotar ese tiempo límite y ofrecer una respuesta de colisión sin haber alcanzado una exactitud en el cálculo puede derivar en algunas malas impresiones en el manejo de las colisiones. Existen algunos trabajos en esta línea que intentan aliviar el problema planteado dando prioridad a las colisiones más importantes (Dingliana y O'Sullivan 2000).

Cada día se requieren aplicaciones de realidad virtual que manejen escenarios cada vez más complejos y numerosos. El argumento central de Hubbard (1995a) es que sólo los algoritmos basados en el tiempo crítico pueden alcanzar el tiempo real y un *frame rate* constante aunque la escena se escale. El razonamiento es simple: los demás algoritmos que no tienen un tiempo máximo establecido darán un *frame rate* variable, y si la complejidad geométrica de la escena gráfica se agranda, el algoritmo será más ineficiente y por tanto el tiempo real quedará afectado.

Sin embargo, en otras aplicaciones como la ingeniería mecánica, aunque la velocidad de cálculo sigue siendo un factor importante, la precisión en el cálculo también es necesaria. Casos típicos en los que la prioridad es la precisión de cálculo pueden ser los estudios de mecanismos, por ejemplo una suspensión, o las simulaciones de mantenibilidad. Para la validación de un modelo dado, no se puede pretender representar los distintos objetos que lo componen de una manera simplificada, ya que los errores geométricos introducidos invalidarían la construcción de una maqueta válida.

Más ejemplos se encuentran en aquellas simulaciones que precisen comprobar colisiones entre sus piezas mecánicas o las que necesitan dar una respuesta física dependiendo exactamente de qué objetos estén en contacto (Mirtich 2000). En este último trabajo, por ejemplo, cada *frame* de la

simulación final necesitaba 97 segundos de cómputo debido a la complejidad de la escena y al número elevado de objetos de la misma.

En aplicaciones de realidad virtual, el cálculo de colisiones debe ser en tiempo real para mantener el *frame rate* y la sensación de inmersión. Es quizás un problema más complejo que los anteriores porque se requieren las dos precisiones: precisión en el cálculo y consumir el menor tiempo posible (Smith et al. 1995, Held et al. 1995, Cohen et al. 1995 y Gottschalk et al. 1996). En tales aplicaciones entran también las que combinan realidad virtual y dispositivos hápticos. La exactitud en el cálculo es importante a la hora de calcular la fuerza necesaria a restituir.

1.3.6 Coherencia espacial

Muchas aplicaciones explotan la coherencia espacial ya que conocen la distribución de los objetos en la escena gráfica. Si estos objetos están alejados unos de otros es entonces cuándo algoritmos basados en la partición espacial pueden ser especialmente eficientes. Cada objeto estará alocado en una región del espacio y tendrá un número muy pequeño de objetos vecinos a su alrededor (Lawlor y Kalé 2002). Será sólo entre esos objetos vecinos entre los que se tienen que realizar los testeos de intersección.

1.3.7 Coherencia temporal y geométrica

Como ya se ha dicho, los algoritmos que explotan la coherencia temporal y geométrica hacen uso de información de intervalos anteriores para lograr un cálculo más efectivo. Las aplicaciones que más utilizan este tipo de característica son las de animación. Para muchas aplicaciones de robótica, asumir coherencia es también algo razonable (Lin 1993).

Por ejemplo, Zhang y Yuen (2002) usan la propiedad de coherencia en sus animaciones para acelerar la detección de colisiones entre la ropa y los modelos humanos fijándose en que en cada paso de tiempo el desplazamiento de cada elemento geométrico es muy pequeño.

Este tipo de coherencia temporal sirve para acelerar cálculos computacionales como describen Jiménez et al. (2001) dónde explican algoritmos basados en el cálculo de distancia para seguir la pista a los elementos (vértices, aristas o facetas) más cercanos entre dos modelos (Cohen et al. 1995, Ponamgi et al. 1997, Hudson et al. 1997). En el siguiente intervalo de tiempo los elementos más cercanos serán los mismos o algún vecino.

En algunos de los trabajos anteriores se explota también la coherencia temporal y geométrica no sólo para acelerar cálculos sino para eliminar pares de

objetos a testear (Cohen et al. 1995, Ponamgi et al. 1997). La idea es muy parecida, entre dos *frames* los objetos se habrán movido muy poco. Este ligero movimiento se aprovecha en la coherencia geométrica ya que la relación entre la geometría de los objetos ha cambiado mínimamente entre los dos *frames*.

1.3.8 Densidad de objetos/polígonos

El tipo de método a elegir depende mucho de la densidad de objetos y polígonos que exista en la escena gráfica. Normalmente la densidad de uno de los parámetros depende del otro.

En general, a los sistemas con baja densidad de objetos y polígonos se les llama sistemas dispersos, y sistemas compactos serán aquellos que posean una mayor densidad (García-Alonso 1990).

Ejemplos de sistemas dispersos son los entornos de simulación dónde aunque pueden existir miles de objetos, su densidad es pequeña en comparación con el volumen del espacio de trabajo. En Cohen et al. 1995 y en Vemuri et al. 1998 se trabaja con entornos de miles de objetos simulando su comportamiento una vez inicializados estos objetos a unas velocidades. La densidad de objetos en el entorno es parametrizable y la miden como el porcentaje del espacio que ocupan los volúmenes de los objetos.

Tener escenarios con un número masivo de polígonos no significa necesariamente estar ante un sistema compacto. En el trabajo de Wilson et al. (1999) se manejan escenarios masivos de polígonos llegando incluso a modelos de plantas de energía de 15 millones de polígonos. Aunque este número es muy elevado en comparación con las aplicaciones normales de detección de colisiones existentes en la bibliografía, éste sigue siendo un sistema disperso ya que el modelo cubre un área de centenares de metros cuadrados.

Los sistemas compactos están compuestos de un número muy elevado de polígonos los cuales están ubicados dentro de un volumen pequeño (en comparación con el volumen total de los objetos). Por ejemplo, en Gottschalk et al. (1996) se trabaja con modelos de cientos de miles de polígonos estudiando el impacto de la simulación de un campo gravitacional en un conjunto de tubos dentro de otro conjunto más grande de tubos.

Sin embargo, los ejemplos más claros de sistemas compactos corresponden a los sistemas de RV que simulan tareas de accesibilidad, mantenibilidad, ensamblaje y análisis en general de modelos de maquinaria (Klosowski et al. 1998, Zachmann 1998, McNeely et al. 1999 y Savall et al. 2002). El número de polígonos puede llegar a ser del orden de millones pero encerrados en un volumen de unos pocos metros cúbicos.

La diferencia entre los dos tipos de sistemas es la proximidad entre los distintos objetos. En la detección de colisiones una técnica muy común es usar volúmenes contenedores en los objetos. Mientras que en los sistemas dispersos la no colisión entre esos volúmenes es el estado natural debido al tamaño del espacio de trabajo, en los sistemas compactos ocurre al revés.

En un sistema compacto, como la mantenibilidad de motores, el estado natural será de colisión entre los volúmenes contenedores de los objetos de la escena y el volumen de la herramienta. Por eso no se pueden seguir las mismas técnicas para unos sistemas que para los otros. En los dispersos gracias a los volúmenes contenedores se puede deducir rápidamente la no colisión. Y aunque esa colisión ocurra, será entre muy pocos objetos.

Si se usa la misma técnica de volúmenes en los sistemas compactos, prácticamente en cada *frame* se estaría dando una respuesta positiva a la pregunta de colisión entre volúmenes. Esto conllevaría el testear en cada *frame* a la herramienta contra todos los objetos cuyos volúmenes contenedores intersectan con el de la herramienta (en tareas de accesibilidad y mantenibilidad normalmente la herramienta siempre está muy próxima a un conjunto grande de objetos, aunque realmente la detección de colisiones entre los polígonos sea negativa como se aprecia en la Figura 1.4).

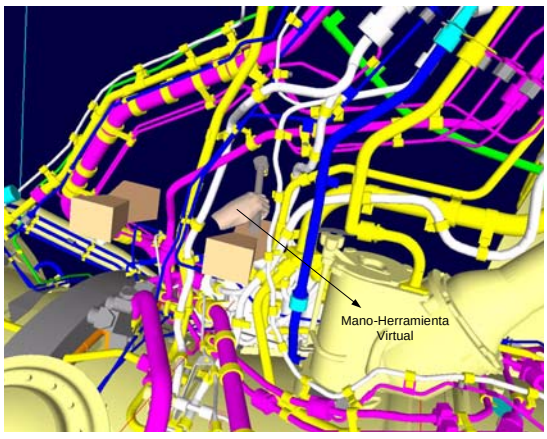


Figura 1.4: Mano y herramienta virtual en las cercanías de un conjunto de tubos.

1.3.9 Objetos móviles vs. Objetos estáticos

Normalmente, los sistemas en los que interesa estudiar el problema de la detección de colisiones, constan al menos de un objeto móvil el cual se va a ir moviendo por la escena y calculando su colisión con el resto de objetos (problema *n*-body estático).

Ejemplos de un objeto móvil contra toda una escena estática se puede observar en las aplicaciones llamadas *Virtual Walkthrough* o paseos virtuales. En estos paseos se van comprobando las colisiones entre el usuario, representado a veces mediante una mano virtual (Cohen et al. 1995) o mediante un avatar (Wilson et al. 1999), y un mundo virtual lleno de objetos.

Estudios de accesos y colisiones para su uso en tareas de mantenibilidad entre una herramienta virtual y móvil y todo un escenario masivo también se pueden ver en Amundarain et al. (2002).

Las aplicaciones en las que sólo existe un objeto móvil son más propias de interacción directa del usuario con un entorno virtual para coger, ensamblar o utilizar diferentes objetos de la escena.

Aplicaciones con varios objetos en movimiento (problema *n*-body) son más propias de simulaciones físicas en las que se quiere ver el comportamiento de una serie de objetos en movimiento y de cómo interactúan unos con otros. El número de objetos del sistema es muy diverso y depende del tipo de aplicación. Existen desde aplicaciones que detectan pares de colisiones entre decenas de objetos (Smith et al. 1995), cientos de objetos (Mirtich 2000), o hasta los entornos simulados que poseen miles (Cohen et al. 1995).

Normalmente, en los problemas *n*-body estáticos el criterio para buscar un buen método es el de acceso y búsqueda rápida de los pares de objetos o polígonos próximos. En los entornos dinámicos se necesita además un algoritmo que actualice rápidamente la estructura de ordenación espacial usada para las colisiones.

1.3.10 Características del movimiento

El método a usar para detectar las posibles colisiones dependerá también de si se conoce a priori el movimiento de los objetos o no.

Si el movimiento de los objetos está especificado previamente, mediante sus posiciones y orientaciones en cada instante, este movimiento puede ser expresado como una función paramétrica del tiempo, como se ha estudiado en el *Path Planning* dentro del campo de la robótica (Latombe 1991, Selim y Almohamad 1999).

Según Hubbard (1995b), para trabajar en una aplicación interactiva, un algoritmo de detección de colisiones debe cumplir dos criterios. Primero, el algoritmo debe conseguir el tiempo real, es decir, un *frame rate* lo más alto y constante posible y además escalable (que no dependa del número de objetos en la escena). Segundo, que el algoritmo debe tolerar objetos cuyo movimiento es

guiado por el usuario, o lo que es lo mismo, que el movimiento no esté predefinido.

En estos casos no se puede contar con funciones dependientes del tiempo ya que el movimiento estará sujeto a fuerzas del usuario, en el caso de entornos virtuales (McNeely et al. 1999), o a leyes dinámicas en el caso de simulaciones dinámicas (Moore y Wilhelms 1988, Hahn 1988 y Pentland 1990).

1.3.11 Últimas consideraciones

Este apartado no pretende resolver ningún problema en concreto sino que es un estudio para orientar a todo aquél que se introduzca en el campo de las colisiones. Gracias a esta clasificación se pueden obtener rápidamente las áreas y campos de interés para un problema dado así como puntos en común con otros problemas que a priori podrían parecer totalmente distintos al interesado.

1.4 CLASIFICACIÓN DE LOS MÉTODOS DE RESOLUCIÓN

Los métodos de detección de colisiones se centran en descubrir la colisión entre los distintos objetos o entre los distintos polígonos de los objetos. Existen multitud de técnicas y algoritmos para lograr ese objetivo. Dependen fundamentalmente del modelo geométrico elegido para representar la escena y del tipo de pregunta que se quiere realizar al entorno (además de otros factores mencionados en el apartado anterior). La presente Tesis trabaja con modelos poligonales y rígidos por lo que la clasificación se centrará en métodos que ataquen este tipo de problemas.

En el cálculo de colisiones, el mayor porcentaje de tiempo se lo lleva la comparación de polígonos y objetos que realmente no están intersectándose. Las técnicas que se detallan a continuación intentan aproximarse a esta solución ideal embebiendo a la escena dentro de distintas estructuras que permiten rechazar trivialmente pares de objetos o polígonos a testear.

Los principales grupos de métodos se clasifican a continuación:

- *Volúmenes contenedores*: los objetos de la escena se encierran en volúmenes y gracias a su geometría más simple la detección de colisiones entre esos volúmenes es más fácil y eficiente.

- *Métodos de subdivisión o partición espacial*: se divide el espacio de la escena en volúmenes más pequeños limitando el problema y

reduciendo el número de objetos/polígonos a testear. Dentro de este grupo también entran las subdivisiones jerárquicas.

- *Métodos de jerarquía de volúmenes*: al contrario que el grupo anterior, la jerarquía de volúmenes no se construye en base al espacio sino partiendo de los mismos objetos. Se dice que estos métodos están orientados al objeto y no al espacio como los anteriores.

- *Métodos de proximidad*: en la escena se tiene información de la vecindad geométrica de cada elemento.

En los próximos apartados se explica con más detalle cada grupo de técnicas expuestas en esta clasificación.

1.4.1 Volúmenes contenedores

Una técnica muy usada para discriminar pares de objetos/polígonos a comparar es encerrar a cada objeto (o grupo de objetos, o grupo de primitivas) dentro de un volumen contenedor. El coste de comparación entre los volúmenes (por lo general con una geometría muy simple) es muy bajo y si el testeo da negativo se puede rechazar objetos enteros sin tener que llegar hasta el nivel más bajo de su geometría.

A veces ni siquiera hace falta tener en cuenta la geometría del objeto. En robótica, por ejemplo, las distintas partes de los robots y su entorno en ocasiones se pueden simplificar utilizando primitivas simples (Rubí 2002). La razón es que se puede permitir el tener errores del orden de varios milímetros siempre y cuando los errores estén del lado de la seguridad.

Existen muchos y muy diversos tipos de volúmenes contenedores, cada uno con sus ventajas y desventajas. Antes de pasar a describir los diferentes volúmenes, a continuación se describen diferentes criterios que se podrían considerar a la hora de evaluar un volumen contenedor. Al final de este apartado, la Tabla 1.1 muestra un resumen comparando todos los volúmenes mostrados en base a estos criterios:

- Criterio geométrico (Cr_G): describe cómo se ajusta el volumen a la forma del objeto. El volumen convexo que más eficientemente se ajusta a un objeto es el llamado *Convex Hull* (Barber et al. 1996) pero su geometría ya no es tan simple y el test de solapamiento entre este tipo de volúmenes no es tan trivial por lo que no se tendrá en cuenta en esta clasificación.

- Criterio de definición (Cr_D): valores necesarios para describir al volumen.

- Criterio de construcción (Cr_C): el grado de dificultad a la hora de construir el volumen a partir del objeto o grupo de objetos.

- Criterio de intersección de volumen (Cr_V): el grado de eficiencia que tiene el testeo de colisión entre dos de esos volúmenes.

- Criterio de actualización de transformaciones (Cr_U): si el objeto se traslada o rota, el volumen también se tiene que transformar para permanecer coherente con el objeto.

1.4.1.1 Cajas alineadas a los ejes (AABB)

Las cajas alineadas a los ejes globales o AABB (*Axis-Alignment Bounding Box*) se construyen considerando las mínimas y máximas coordenadas del objeto que encierran. Es por tanto muy sencillo de construir, sólo hay que buscar esos mínimos y máximos. La convergencia del método es *O(n)* siendo *n* el número de vértices del objeto. Como almacenamiento requiere únicamente los 6 valores ($\mathbf{p}_{min}$ y $\mathbf{p}_{max}$ siendo $\mathbf{p}=(x,y,z)$).

La ventaja del AABB es la facilidad y eficiencia del cálculo de intersección entre dos volúmenes, sólo hay que mirar si se solapan las coordenadas en cada eje como aparece en la Figura 1.5 (para el caso 3D la aplicación es inmediata). Sin embargo, el inconveniente es la baja eficiencia geométrica que posee (Cr_G). Este volumen no se ajusta muy bien a los objetos, sobre todo si los objetos son largos, delgados y orientados en direcciones diagonales como por ejemplo los tubos de un motor.

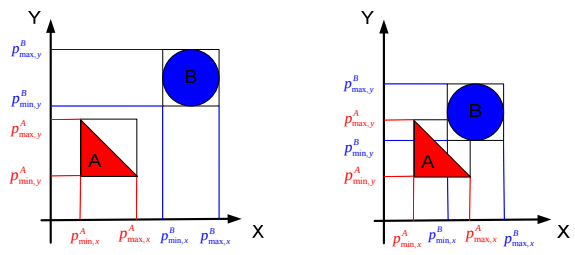


Figura 1.5: Ejemplo 2D de la intersección entre 2 AABBs.

Para transformar el AABB acorde al movimiento del objeto contenido se tienen dos posibilidades (se supone que existe translación y rotación a la vez ya

que si la orientación del objeto no cambia entonces sólo hay que trasladar los contenedores):

- Una posibilidad es construir la caja alineada del objeto según sus nuevas coordenadas con lo que requiere una pasada por todos los vértices del objeto y por lo tanto *O(n)*.

- Como se explica en la sección siguiente, otra posibilidad consiste en transformar los ocho vértices del OBB del objeto con la transformación sufrida por el objeto y calcular la nueva caja alineada de esos vértices. Esta solución es eficiente pero el inconveniente es que el volumen resultante puede llegar a ser el doble del AABB original por lo que esta solución no suele interesar.

Un método muy usado para manejar eficientemente un número elevado de objetos y sus volúmenes contenedores es utilizar el método de proyección. Se evalúan las colisiones considerando por separado las proyecciones de la escena en diferentes ejes (Cohen et al. 1995). A esta técnica se la llama *Sweep and Prune*.

Para implementar esta técnica se proyectan los AABB's de los objetos en cada uno de los ejes de coordenadas resultando un conjunto de intervalos unidimensionales. A partir de ahora se puede utilizar un algoritmo unidimensional de solapamiento en cada dimensión. En la Figura 1.6 se puede ver un ejemplo, si hay intersección en los dos ejes, entonces hay colisión entre los AABB's de esos objetos (para el caso 3D es idéntico).

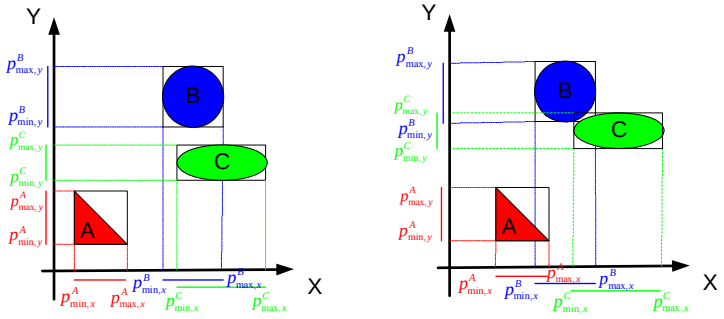


Figura 1.6: Ejemplo 2D del método de proyección Sweep and Prune.

Gracias a esta técnica y a caracterísiticas específicas del problema, estos autores logran reducir el orden cuadrático $O(n^2)$ en la *broad phase* del problema *n*-body a un problema de orden lineal *O(n)* utilizando técnicas de ordenación de listas. Este algoritmo es usado en las librerías de detección de colisiones I-

COLLIDE (Cohen et al. 1995) y V-COLLIDE (Hudson et al. 1997). En ambas se tiene una arquitectura de detección de colisiones de varios niveles.

Los autores también utilizan un segundo método de proyección, en este caso bidimensional. La idea es la misma que el anterior método pero proyectando los AABBs en los planos *xy*, *xz* e *yz*. En lugar de intervalos como en el caso 1D, el resultado de estas proyecciones son rectángulos en el espacio 2D. Después utilizan un algoritmo que detecta los solapamientos entre rectángulos y los va ordenando. Este método lo utilizan cuándo las proyecciones de la técnica unidimensional *Sweep and Prune* dan como resultado un conjunto numeroso de intervalos densos. Generalmente hay menos solapamientos entre rectángulos 2D que entre los intervalos 1D.

1.4.1.2 Cajas orientadas al objeto (OBB)

Las cajas orientadas al objeto u OBBs (*Oriented Bounding Box*) son cajas rectangulares con una orientación arbitraria en el espacio 3D. Gracias a esta orientación, su Cr_G es mucho mejor que el de las cajas alineadas a los ejes. Se pueden almacenar de varias formas. Una requiere el punto central del OBB, p_c, y tres vectores normalizados, **u**, **v** y **w,** positivos y orientados con las respectivas longitudes medias l_u, l_v, l_w. Otra forma es guardar 8 coordenadas, una por cada vértice de la caja. Para actualizar el OBB sólo es necesario aplicar la matriz de transformación a los vectores orientados o a los 8 vértices, según el tipo de almacenamiento elegido.

Sus principales características son: su Cr_C tiene convergencia *O(nlgn)*, mayor que el caso del AABB aunque en el caso de objetos rígidos se calcula sólo en preproceso. El test de interferencia es más complejo. En Gottschalk et al. (1996) se logra realizar este test en cerca de 200 operaciones (en el caso peor) gracias al llamado *teorema de los ejes separadores*.

Este test consiste en proyectar las cajas en algún eje del espacio (no necesariamente en los ejes de coordenadas). En esta proyección, cada caja forma un intervalo; si estos intervalos no se solapan entonces las dos cajas no se intersectan y a este eje se le llama *eje separador*. Un ejemplo se puede ver en la Figura 1.7.

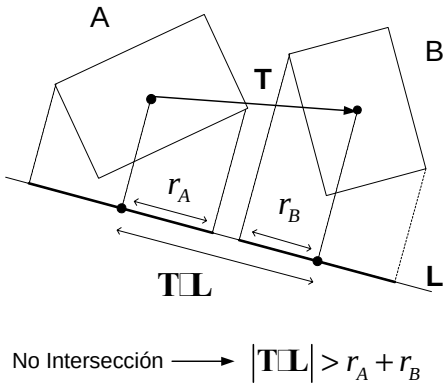


Figura 1.7: Ejemplo 2D en el que dos OBBs A y B no intersectan. **L** es un *eje separador* porque las proyecciones de las cajas no se solapan.

Si los intervalos se solapan es necesario realizar más testeos con diferentes ejes. Existe un número máximo a chequear. Gottschalk demuestra que entre dos poliedros convexos disjuntos en el espacio 3D siempre existe un plano separador que puede ser paralelo a alguna cara de los poliedros, o paralelo a una arista de cada poliedro. Una consecuencia de esto último es que dos poliedros convexos no intersectarán si y sólo si existe un *eje separador* perpendicular a alguna cara de los poliedros o perpendicular a una arista de cada poliedro. Por tanto, como mucho se realizarán 15 chequeos de *ejes separadores* (3 por las orientaciones de caras de una caja, otras 3 por la otra caja y 9 más por las combinaciones entre las 3 orientaciones de aristas de cada caja).

En García-Alonso et al. (1994) ya se utilizaba el OBB para encerrar a cada objeto de la escena pero para evitar el coste computacional del test de solapamiento entre OBBs, se encerraban estos a su vez en AABBs. De esta forma el primer testeo se realizaba a nivel de cajas paralelas a los ejes siendo rápida su respuesta.

1.4.1.3 Esferas

Para definir una esfera se requieren cuatro valores: las tres coordenadas del centro, $\mathbf{p}_c$, y un cuarto valor definiendo el radio de la esfera r. El test de interferencia entre dos esferas es también muy eficiente (menos que el del AABB pero mejor que el del OBB). Para comprobar la colisión sólo hay que mirar si la distancia entre los dos centros es menor que la suma de los dos radios como se muestra en la Figura 1.8.

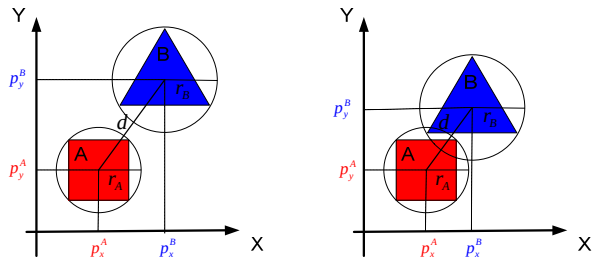


Figura 1.8: Ejemplo 2D de la intersección de dos esferas.

La desventaja es su criterio geométrico, una esfera no delimita bien al objeto (contra menos esférica sea la forma del objeto, menos eficiencia geométrica poseerá la esfera contenedora).

Según Ritter (1990) construir una esfera requiere dos recorridos por todos los vértices del objeto (*O(n)*).

Una ventaja considerable que tiene la esfera y que no poseen los volúmenes AABB y OBB es su independencia en cuanto a la orientación. No depende de ningún eje (local o global), sólo le afectan las transformaciones de translación (al centro) o de escalado (al radio).

Un ejemplo gráfico de este volumen contenedor y de los dos anteriores se muestra en la Figura 1.9.

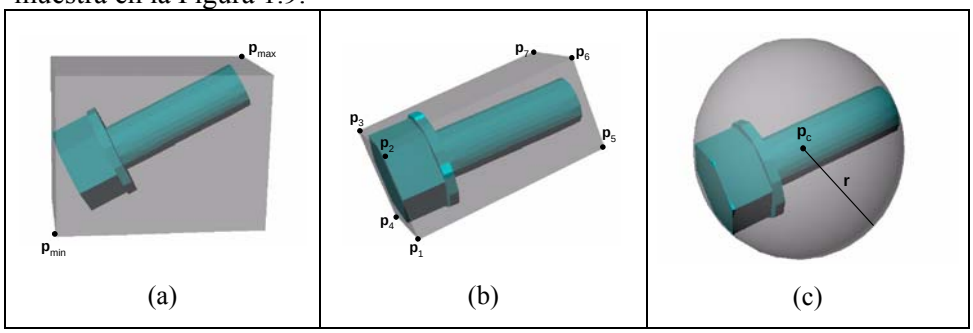


Figura 1.9: Volumen encerrado por un AABB (a), un OBB (b) y una esfera (c) junto con los datos necesarios para su definición.

1.4.1.4 Polihedro discreto orientado (*k*-Dop)

Los k-Dops (*Discrete Orientation Polytopes*) son volúmenes definidos por planos a lo largo de direcciones fijas (Klosowski et al. 1998). Un AABB corresponde a un 6-Dop truncado por los planos (*x=1*, *y=1* y *z=1*) y sus opuestos. Uno de los *k*-Dops más utilizado es el 14-Dop que se crea a partir del

6-Dop añadiendo a éste los planos definidos por las normales (1,1,1), (1,-1,1), (1,1,-1), (1,-1,-1) y sus opuestos (corresponde a una caja contenedora AABB pero con las esquinas truncadas). Las diferencias entre las distintas cajas aparecen en la Figura 1.10.

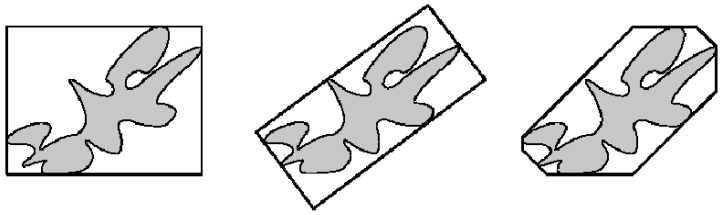


Figura 1.10: Ejemplo 2D de un AABB (a), un OBB (b) y un 8-DOP (c) (Klosowski et al. 1998).

Para construir fácilmente un *k*-Dop basta con encontrar las posiciones extremas del objeto en cada una de las *k* direcciones consideradas, su construcción es por tanto lineal *O(n)*.

Para definir un *k*-Dop se requieren *k/2* tripletas ($\mathbf{n}_i$, $d_i^{\min}$, $d_i^{\max}$)[1] que describen el volumen $\mathbf{V}_i$ (encerrado por los planos $\boldsymbol{\pi}_i^{\min} : \boldsymbol{n}_i \cdot \boldsymbol{x} + d_i^{\min} = 0$ *y* $\boldsymbol{\pi}_i^{\max} : \boldsymbol{n}_i \cdot \boldsymbol{x} + d_i^{\max} = 0$) y dónde la intersección de todos los volúmenes, $\bigcap_{1 \le l \le k/2} \boldsymbol{V}_l$ corresponde exactamente con el volumen encerrado por el *k*-Dop.

De los volúmenes vistos hasta ahora, el *k*-Dop es el más eficiente desde el punto de vista geométrico. Encierra al objeto mejor que el OBB ya que puede ir recortando esquinas de la caja y ajustándose de esa manera mucho mejor al objeto minimizando el espacio vacío.

El test de interferencia es muy rápido y sencillo y basta realizar *k/2* tests de solapamiento 1D (como en el caso del AABB pero para cada normal $\mathbf{n}_i$).

Al mover un objeto, el coste de actualizar su *k*-Dop (sistema de referencia global) es demasiado costoso computacionalmente por lo que se suele recalcular un nuevo *k*-Dop no en base al conjunto de vértices del objeto sino al *k*-Dop

[1] En realidad sólo son necesarios los valores mínimos y máximos ya que para una escena llena de objetos, todos compartirán las mismas normales $\mathbf{n}_i$ que serán almacenadas una única vez.

original transformado de acuerdo al movimiento del objeto. El resultado aumenta el volumen contenedor pero es válido como solución de compromiso.

1.4.1.5 Últimas consideraciones

A continuación, la Tabla 1.1 resume los criterios definidos al comienzo del apartado para cada volumen contenedor.

	AABB	OBB	Esfera	*k*-DOP
Cr_G	Mejora la esfera	Peor que *k*-DOP	El peor	El mejor
Cr_D (nº de valores)	6	24	4	$K>6$
Cr_C	$O(n)$	$O(nlgn)$	$O(n)$	$O(n)$
Cr_V (nº de ops)	≤ 6	< 200	8	$\leq K$
Cr_U (nº de transf)	$O(n)$	8	1	$2K$

Tabla 1.1: Comparativa de criterios entre los diferentes volúmenes contenedores.

A la hora de tomar una decisión entre los distintos volúmenes contenedores, hay que considerar el tipo de aplicación en la que se va a usar. Por ejemplo, la esfera parece una buena candidata para casi cualquier aplicación siendo la más eficiente en el consumo de memoria y en la actualización de su volumen. Sin embargo, si se requiere precisión en el cálculo la esfera no es una buena elección al ser la que peor aproxima al objeto original.

La caja paralela a los ejes es uno de los volúmenes más usados por su simplicidad y rapidez en el cálculo de intersección. La única desventaja se encuentra en su Cr_U por lo que este volumen sería un buen candidato para aproximar sólo a los objetos estáticos.

Para objetos móviles la decisión entre el OBB o el *k*-DOP depende de a qué se le de prioridad. Mientras que el k-DOP es más eficiente en el cálculo de intersección, el OBB no pierde Cr_G al actualizar el volumen con el movimiento del objeto.

1.4.2 Métodos de subdivisión o partición espacial

En un problema *n*-body o *n*-body estático, una consideración clara es realizar el cálculo de colisiones únicamente entre los objetos que estén geométricamente cercanos unos a otros. Las técnicas de subdivisión espacial dividen o particionan el espacio en volúmenes más pequeños y mantienen una lista de los objetos contenidos en cada región. La *narrow* phase de la que hablaba Hubbard

(1993) sólo se tendrá que calcular para aquellos pares de objetos que compartan la misma región o volumen espacial en un instante dado.

1.4.2.1 Subdivisión espacial uniforme

1.4.2.1.1 Enumeración espacial

El método de subdivisión más común y sencillo es la enumeración espacial en la cual el espacio es descompuesto mediante un conjunto de celdas idénticas unidas entre sí, disjuntas y ordenadas en una malla tridimensional fija y regular (Foley et al. 1990). Estas celdas son iguales entre sí y pueden ser cubos (Held et al. 1995 y McNeely et al. 1999) o bien paralelepípedos (García-Alonso et al. 1994). Las celdas o elementos de la malla tridimensional usados para subdividir el espacio son llamados voxels.

La simplicidad de este método radica en que la división del espacio es regular, por lo que es muy fácil encontrar el voxel que contiene a cualquier punto **p** (y por consiguiente el voxel que contiene a cualquier objeto/polígono de la escena) tal y como aparece en la Figura 1.11. Esta operación se realiza en tiempo constante (*O(1)*).

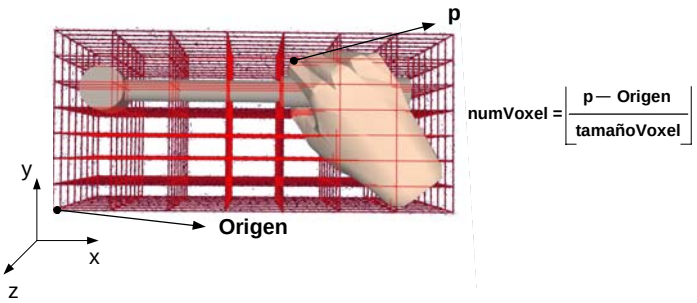


Figura 1.11: Subdivisión en voxels y su búsqueda de *O(1)*.

Esta técnica (y los métodos de partición espacial en general) puede ser usada tanto para problemas *n*-body (Lawlor y Kalé 2002) en los que los voxels apuntarían a objetos, como para los *n*-body estáticos (Savall et al. 2002) en cuyo caso los voxels contienen polígonos. Los problemas son distintos pero ambos explotan la coherencia espacial y sacan provecho de la rápida búsqueda y acceso a la hora de determinar qué voxels ocupa un objeto (en el primer caso) o un polígono (en el segundo caso).

Un problema existente es la elección del nivel óptimo de discretización o voxelización, es decir, el número de voxels más adecuado en los que conviene subdividir el espacio tridimensional. Siendo un problema tan complejo, se ha

dedicado todo un capítulo de esta Tesis a su estudio. Los resultados y la aportación dada en esta Tesis se ofrecen en el capítulo 4.

En la clasificación de problemas al comienzo de este capítulo, se comentaba que los problemas *n*-body podían llegar a tener una complejidad cuadrática en algunas situaciones especiales. En la Figura 1.12 se puede apreciar un ejemplo en el que prácticamente todos los objetos están localizados en el mismo voxel. El algoritmo normal de discriminación de objetos gracias a los voxels no daría apenas resultado en este caso y daría un orden de $O(n^2)$.

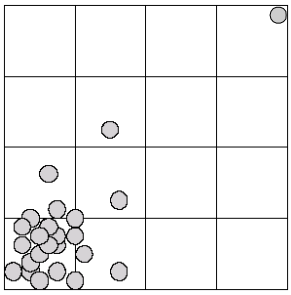


Figura 1.12: Caso especial de orden cuadrático utilizando enumeración espacial.

Como suele ocurrir, es frecuente que exista alguna desventaja. En el caso de los voxels la primera es que consume demasiada memoria cuando el nivel de voxelización empieza a ser elevado. Además, como la subdivisión es independiente del escenario, la mayoría de esa memoria está desaprovechada en espacios vacíos. Esto se puede evitar mediante métodos jerárquicos (capítulo 2) o utilizando técnicas de *hashing* (capítulo 3).

La subdivisión uniforme se suele utilizar en problemas estáticos para calcular en un único preproceso qué voxels contienen a qué objetos y no haya que modificar esa estructura en el resto de la ejecución de la aplicación. Sin embargo, hay trabajos en los que se utilizan estructuras de voxels con entornos de múltiples objetos móviles (García-Alonso et al. 1994). En estos casos se calcula una estructura voxel para cada objeto (partiendo del OBB de cada uno) y no para toda la escena gráfica. El inconveniente es que al ser objetos móviles, al comparar dos objetos, hay un mayor gasto computacional para evaluar coordenadas, aunque existen métodos incrementales que lo hacen menos costoso que las transformaciones de coordenadas.

Una aplicación muy parecida a los objetivos de esta Tesis y en la que también utilizan métodos de subdivisión espacial uniforme es el trabajo de McNeely et al. (1999) para la compañía Boeing. Su objetivo es realizar tareas de mantenimiento en modelos masivos de motores aeronáuticos con un objeto

móvil (una mano, una figura o una herramienta) moviéndose por la escena virtual.

En ese trabajo utilizan voxels para implementar el software VPS^{TM} (*Voxmap PointShell*) basado en dos estructuras distintas. Una es el *Voxmap* y se utiliza para voxelizar el objeto estático en todo su volumen. Consiste en crear una estructura de voxels para todo el entorno como si fuera un único objeto estático. Cada voxel contiene 2 bits para definir su tipo que puede ser de espacio libre, interior, de superficie o de proximidad. A partir de ahí trabajan únicamente con el modelo de partición espacial. Para aliviar el problema de memoria implícito en la solución enumerativa, optan por una generalización del octree con una profundidad limitada a 3 niveles. Después, cada volumen cúbico del octree es dividido en subvolúmenes siguiendo una metodología de voxels. De este modo los espacios libres se ocupan con celdas del octree y serán las celdas hoja dónde se calcule la voxelización.

La otra estructura es el *Point Shell* utilizada como alternativa a la representación poligonal del objeto móvil. Consiste en una nube de puntos definiendo la superficie del objeto. Cada punto tiene asociado una normal que apunta al interior del objeto. Para calcular la nube de puntos también se hace uso de una estructura voxel del objeto y de su B-Rep. Una vez se tiene el *Voxmap* del objeto móvil, la colección de puntos centrales de los voxels de superficie define el *Point Shell*, tal y como aparece en la Figura 1.13.

Figura 1.13: Modelo poligonal de la tetera (a), modelo de voxels (b), y el modelo *Point Shell* (c) (McNeely et al. 1999).

El uso combinado de estas dos estructuras ofrece un método muy atractivo para calcular la penetración del objeto móvil con los objetos del entorno. El problema de esta solución es la precisión que posee. Al representar el objeto mediante voxels, y no mediante su representación original de polígonos, tanto el *Voxmap* como el *Point Shell* tienen un error $\varepsilon \leq \sqrt{3}s/2$ donde s es el tamaño del lado del voxel. En la Figura 1.14 se muestra un ejemplo de las dos estructuras usadas por estos autores.

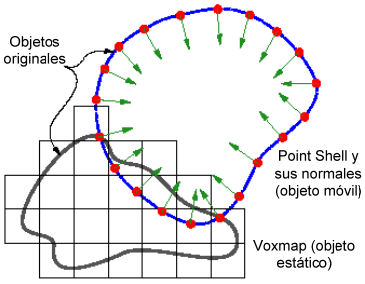


Figura 1.14: *Voxmap* de un objeto estático colisionando con el *Point Shell* de un objeto móvil.

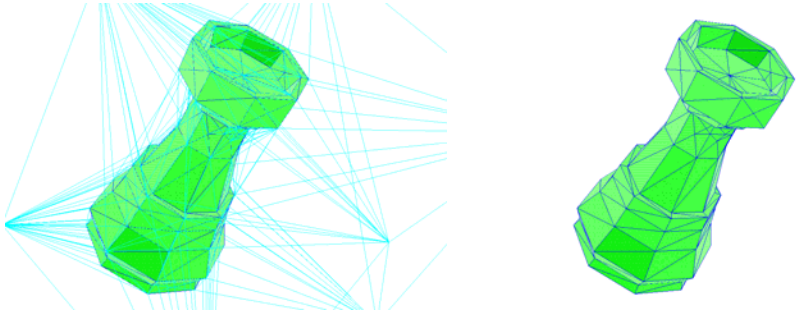
En un trabajo posterior se mejora el algoritmo, en cuanto a precisión, proyectando los puntos del *Point Shell* a los triángulos de la superficie del objeto (Renz et al. 2001).

1.4.2.1.2 Descomposición en celdas

La enumeración espacial es en realidad un caso especial de la descomposición en celdas. En la enumeración, las celdas son iguales y de idéntico tamaño cosa que no tiene por que ocurrir en la descomposición en celdas. Normalmente, el tipo de celda a usar es el tetraedro. En Held et al. (1995) se utiliza como estructura de datos para el módulo de colisiones.

Para generar la estructura de tetraedros, se parte del conjunto de vértices originales de la escena 3D y se calcula su triangulación Delaunay (Sapidis y Perucchio 1991a y Sapidis y Perucchio 1991b). Después, para cada triángulo de la escena gráfica se mira a ver si es intersectado por alguna faceta de algún tetraedro de la triangulación Delaunay. Si no hay intersección alguna, el proceso se ha terminado; pero si existe intersección se obtienen nuevos puntos (resultado de la intersección de la arista del entorno con la faceta del tetraedro) y se actualiza la triangulación Delaunay.

La detección de colisiones se realiza con el método del *ray shooting*. El rayo es obtenido del vector trayectoria del objeto móvil entre un frame y otro y se compara con la malla de tetraedros del objeto estático. Las facetas de aquellos tetraedros que intersecten con el rayo serán las que se comparen con los triángulos del objeto móvil.



(a) (b)

Figura 1.15: Malla de tetraedros para un modelo 3D. Las líneas representan aristas del tetraedro Delaunay (a). En (b) sólo se dibuja aquellos tetraedros de la malla que forman la frontera del modelo.

1.4.2.2 Subdivisión espacial jerárquica

Una de las desventajas de los métodos uniformes de subdivisión es su falta de adaptación a las formas geométricas de los objetos o a su distribución en el espacio, con la consecuencia de que regiones vacías del espacio también ocupan memoria en el ordenador. Este problema se elimina jerarquizando la subdivisión espacial.

A diferencia de los algoritmos basados de jerarquía de volúmenes (que se verán en el siguiente apartado), los que se presentan ahora subdividen o particionan el espacio 3D y no el conjunto de objetos. El resultado es que los volúmenes usados nunca intersectan unos con otros (la única excepción es el árbol de esferas presentado en este apartado pero se debe a la geometría implícita de la primitiva usada y no a la filosofía del particionamiento).

1.4.2.2.1 BSP (Binary Space Partitioning)

El algoritmo BSP expuesto más detalladamente por Naylor et al. (1990) va cortando recursivamente la escena en dos mitades mediante planos (ver la Figura 1.16) para ir formando celdas convexas. Al contrario que en el octree, los nodos del árbol son separadores del espacio 3D en lugar de regiones.

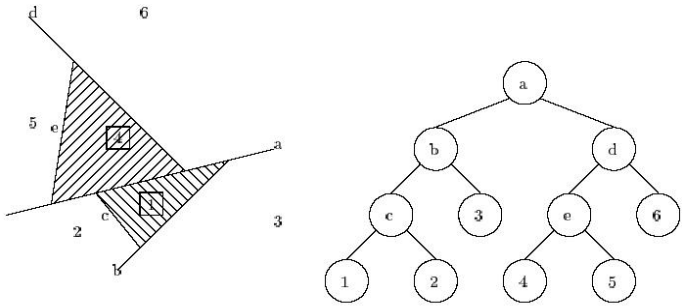


Figura 1.16: BSP-Tree de un conjunto de polígonos.

Los árboles BSP se usan en el campo del modelado sólido para representar los objetos, como alternativa a la representación frontera, y poder así realizar operaciones sobre ellos (uniones, intersecciones, etc.).

Un inconveniente de esta estructura consiste en encontrar los planos de corte para dividir la escena gráfica manteniendo el árbol BSP lo más pequeño posible. La longitud que puede llegar a tener cada uno de estos árboles puede ser considerable ya que su profundidad mínima es $\Omega(n^2)$ para n objetos, lo cual repercute negativamente sobre el almacenamiento requerido por este tipo de estructuras.

Este problema puede ser aliviado en aplicaciones *walkthrought* donde se puede conocer a priori el camino que va a seguir el usuario y de esta forma construir el árbol con ese conocimiento previo (Sigal et al. 2002).

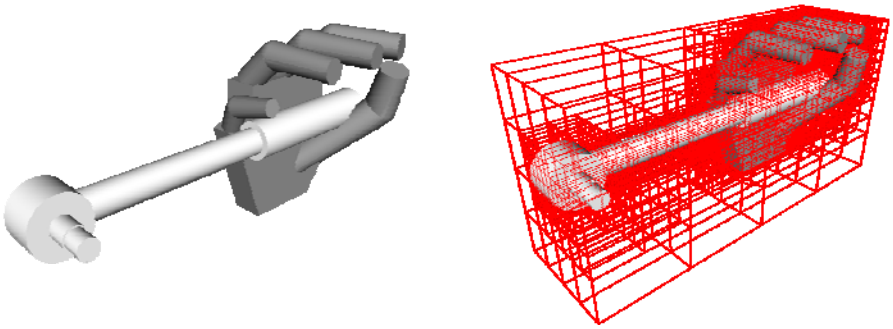
1.4.2.2.2 Octree

En los octrees se parte del AABB de la escena global y se divide éste en ocho cubos o cajas usando los planos alineados a los ejes globales. Los nodos hijos se irán subdividiendo recursivamente hasta alcanzar la resolución requerida (Ayala et al. 1985, Hayward 1986 y Vemuri et al. 1998). Los nodos de la estructura octree tendrán tres posibles estados:

- *Lleno*: el nodo está totalmente contenido en el objeto (nodo hoja).
- *Vacío*: el nodo no contiene ningún volumen del objeto (nodo hoja).
- *Parcial*: el nodo está intersectando parcialmente al objeto (nodo interno).

La subdivisión sólo se irá produciendo en los nodos parciales, ya que en los nodos llenos y vacíos no tiene sentido seguir subdividiendo. De aquí que

una de las ventajas de los octrees es el ahorro de almacenamiento que supone el no seguir particionando el espacio. Un ejemplo de este tipo de subdivisión es mostrado en la Figura 1.17.



(a) (b)

Figura 1.17: Subdivisión de un modelo 3D (a) usando una jerarquía de octrees (b).

Al igual que los voxels, los nodos hoja de un octree mantendrán una lista a los objetos (Shaffer y Herb 1992) o a los polígonos (Smith et al. 1995) contenidos en ese nodo.

La ventaja de tener un octree de objetos (llamado *octree de N objetos* siendo *N* el número máximo de objetos en un nodo hoja) es que la estructura árbol es más pequeña y compacta permitiendo rápidas actualizaciones si existe movimiento de objetos. Sin embargo los octrees de polígonos son mejores si lo que se necesita es precisión en el cálculo y se quiere evitar el tener que comparar más polígonos de los necesarios.

Un esquema simple del procedimiento para detectar las colisiones entre dos estructuras octree (octreeA y octreeB) es el siguiente. La función *collisionTest* realiza un test básico de intersección entre dos AABBs.

```
NodoA = root (octreeA)
NodoB = root (octreeB)
Si  (collisionTest (nodoA, nodoB)==false) devolver NO_COLLISION
sino
  Para cada hijo i de nodoA
        Para cada hijo j de nodoB
              collisionTestHijo(hijoI, hijoJ)
```

La función *collisionTestHijo (hijoI, hijoJ)* además de realizar el test básico de intersección entre AABBs, debe mirar qué tipo de nodos son los hijos. Si no son nodos hojas entonces el test debe seguir mirando recursivamente los hijos de esos nodos. Si los nodos hoja están vacios no se sigue mirando pero si

están llenos o parcialmente llenos entonces hay que realizar un test entre los objetos o polígonos que mantiene cada nodo en sus listas.

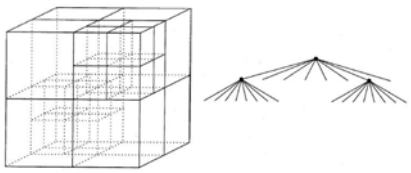
La desventaja en cuanto a los métodos de subdivisión uniformes es el acceso. Un acceso más rápido que el conseguido con una estructura voxel es muy difícil. Los algoritmos que maneja el octree son algoritmos recursivos que recorren una estructura de tipo árbol. Este tipo de algoritmos tienen una complejidad logarítmica *O(logn)*, en el peor de los casos, siendo *n* el número de nodos hoja.

1.4.2.2.3 K-d Tree

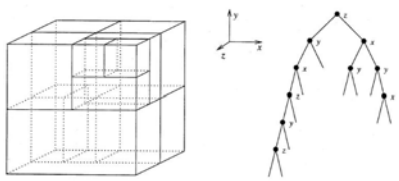
Uno de los métodos que utilizan Held et al. (1995) en su trabajo es el llamado *k*-d *Tree* (El número *k* indica el número de dimensiones que se utiliza). Se construye su estructura de tal modo que puede ser considerado como un caso especial del octree y del BSP *Tree*. También es llamado *BSPTree* Ortogonal porque usa planos paralelos a los ejes de coordenadas del objeto para ir subdividiendo el espacio (Zachmann 1997) en lugar de elegir planos completamente arbitrarios.

La diferencia con el octree radica en que en lugar de subdividir el espacio en 8 octantes mediante los 3 ejes de coordenadas, el *k*-d Tree subdivide en dos la región a lo largo de uno de los ejes de coordenadas.

En la Figura 1.18 se puede ver una representación visual del *k*-d *Tree* comparándolo con el octree. Según Van Den Bergen (1999a), con el k-d *Tree* se puede llegar a conseguir tener menos celdas que con una estructura octree.



(a)



(b)

Figura 1.18: Comparación entre dos estructuras de subdivisión espacial jerárquica. Octree (a) y *k*-d Tree (b) (Van Den Bergen 1999a).

A la hora de subdividir un nodo del árbol, surgen dos preguntas:

- Elección del plano de corte (ortogonal al eje *x*, *y* o *z*).
- Una vez elegido el plano de corte, decidir en qué valor del eje de coordenadas cortará la caja.

Los nodos del árbol resultan ser cajas orientadas a los objetos (OBB's) así que este mismo autor, en un trabajo posterior (Zachmann 2000), utiliza la misma jerarquía de cajas orientadas al objeto por el buen C_G que se consigue; pero siendo consciente del coste computacional que conlleva el testeo entre dos OBB's, opta por una solución intermedia que consiste en encerrar los OBB's dentro de AABB's. A la estructura generada le llama BoxTree pero en realidad usa la metodología de los *k*-d Tree (no se parece a la metodología orientada al objeto/polígono que se verá en el siguiente apartado).

En Held et al. (1995) se describen los diferentes criterios utilizados para las dos cuestiones anteriores y se realiza también un estudio comparativo entre esta técnica y otros métodos de partición espacial implementados por los autores (el R-*Tree*, una malla de tetrahedros y un malla de celdas uniforme).

1.4.2.2.4 SphereTree

La idea es muy parecida al octree. En lugar de ir subdividiendo en cubos, se utiliza la esfera como primitiva pero el algoritmo de recursión está basado en el octree por lo que cada nodo padre tendrá también ocho nodos esfera hijos (Hubbard 1993 y Palmer y Grimsdale 1995). El número de resultados positivos entre esferas será mayor que con otros volúmenes ya que esta primitiva no ajusta tan bien como un cubo, pero ese mismo cálculo de intersección es también más rápido.

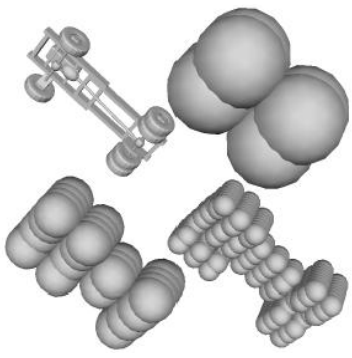


Figura 1.19: Ejemplo de varios niveles de recursión del árbol de esferas (Hubbard 1995a).

1.4.3 Métodos de jerarquía de volúmenes

A diferencia de los anteriores métodos en los que se particiona el espacio, en las jerarquías de volúmenes se particionan los mismos objetos (o polígonos) agrupándolos y usando sus volúmenes contenedores hasta formar un árbol. Si el volumen contenedor de un nodo no intersecta con un objeto, tampoco lo harán sus nodos hijos por lo que se puede saltar en la jerarquía. Mientras dos nodos se intersecten habrá que ir descendiendo por el árbol hasta llegar a los nodos hoja dónde se resuelve el test entre polígonos.

La dificultad de estos métodos radica en construir el árbol eficientemente. Se puede construir con metodología *Top-Down* en dónde se empieza el árbol por su raíz conteniendo el volumen ocupado por todos los objetos de la escena y se va subdividiendo el volumen hasta llegar a los nodos hoja; o *Bottom-Up* en la que primero se tienen los volúmenes correspondientes a los nodos hoja (cada volumen contiene a una primitiva o grupo de primitivas) y se van uniendo en volúmenes más grandes hasta que se completa toda la jerarquía del árbol.

Existen jerarquías para todos los volúmenes contenedores comentados anteriormente. Dependiendo del volumen elegido, el test de intersección será muy eficiente, en el caso de esferas, o tendrá un coste computacional mayor como es el caso de los OBBs. Aunque este último se ajusta mucho mejor a la geometría del objeto y requerirá por ello realizar menos tests.

Existen muchos trabajos basados en estos métodos jerárquicos. A la vez que se describen los distintos tipos de jerarquías volumétricas, se irá enunciando distintos paquetes públicos disponibles en la Web. A continuación se enumeran las jerarquías de volúmenes más usadas en la bibliografía. En Zachmann (2000) se puede ver un estudio comparativo entre las jerarquías *OBBTree*, *k*-DOP *Tree* y dos métodos implementados por el autor (el *BoxTree* comentado en el apartado anterior cuya base son los *k*-d *Tree*, y el DOP-*Tree*).

1.4.3.1 Árbol de cajas alineadas (AABBTree)

Los árboles de cajas alineadas son atractivos por el bajo coste que supone el test de intersección entre un par de cajas alineadas. En la Figura 1.20 se puede ver un ejemplo 2D de una jerarquía de objetos utilizando cajas alineadas. A la derecha se tiene el volumen contenedor que engloba a toda la escena y a la izquierda se ven todas las cajas pertenecientes a los nodos de la jerarquía.

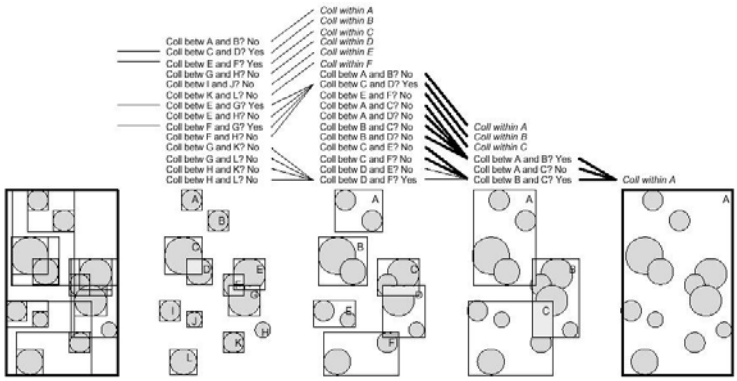


Figura 1.20: Jerarquía de objetos utilizando un AABBTree.

Held et al. (1995) comparan este método, al que llaman R-*Tree*, con otros métodos de partición espacial. Utilizan para sus experimentos escenarios compuestos de decenas de miles de triángulos.

Se parte del nodo raiz que contiene a toda la escena gráfica (*Top-Down*). La metodología de subdivisión se basa en localizar los centroides del conjunto de triángulos de ese nodo. Se corta con un plano ortogonal a uno de los ejes de coordenadas (se elige en base a los volúmenes de los AABBs que quedan a cada lado) y que pasa por la mediana de esos centroides. Cuando un nodo contiene un número de triángulos menor que un cierto umbral *K*, entonces se deja de subdividir ese nodo.

Ya que lo que se subdivide es el conjunto de objetos y después se calcula la caja que engloba a esos objetos, diferentes cajas ubicadas en diferentes nodos pueden, y de hecho lo hacen, intersectarse entre sí (a diferencia de los métodos jerárquicos de subdivisión espacial como el octree y el *k*-d Tree).

Van Den Bergen (1997) también utiliza esta jerarquía para implementar el sistema de detección de colisiones entre objetos deformables (con miles de triángulos) llamado SOLID donde demuestra que para entornos con deformación de objetos, este tipo de árboles son más eficientes que otros que utilicen cajas orientadas debido a su rápida construcción.

1.4.3.2 Árbol de cajas orientadas (OBBTree)

La librería RAPID (Gottschalk et al. 1996) hace uso de una jerarquía de OBBs para detectar posibles pares de triángulos intersectándose. Después realiza un test exacto entre esos pares de triángulos para comprobar cuáles de ellos están realmente en intersección. El número de triángulos de las escenas con las que

realizan los experimentos alcanzan el orden de magnitud de centenas de miles de polígonos.

Para construir el árbol de cajas orientadas, primero muestran el algoritmo que utilizan para, dado un conjunto de polígonos, computar su OBB. Se construye el árbol *Top-Down* empezando con el volumen contenedor de toda la escena. Como regla de subdivisión se corta el eje con longitud más larga con un plano ortogonal asignando los polígonos dependiendo de dónde se situen sus puntos centrales. Para cada uno de esos dos *sets*, se evalua su OBB que se volverá a subdividir (en la Figura 1.21 se puede ver un ejemplo).

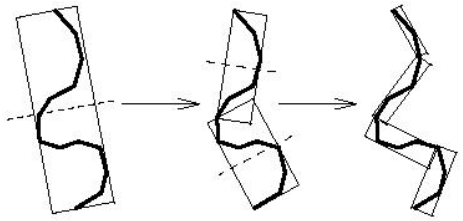


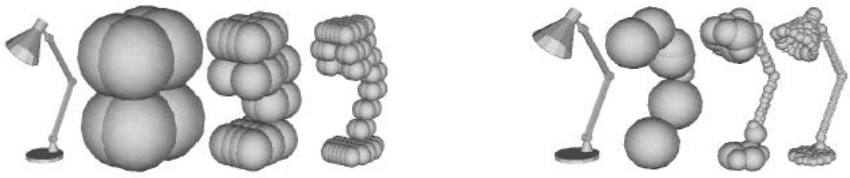
Figura 1.21: Subdivisión recursiva de un OBB para construir el OBBTree (Gottschalk et al. 1996).

Otros trabajos con OBBs aparecen en Barequet et al. (1996) dónde utilizan una jerarquía de OBBs (Bottom-Up) llamada boxtree que es distinto del *k*-d Tree de Zachmann (1997) visto anteriormente.

Otro ejemplo se tiene en la librería V-COLLIDE (Hudson et al. 1997) que maneja decenas de miles de polígonos y en la que se hace uso del OBBTree para resolver la *narrow phase*, aunque en un primer nivel se hace uso de la técnica *Sweep and Prune* (en realidad V-COLLIDE es un híbrido entre las librerías I-COLLIDE y RAPID).

1.4.3.3 Árbol de esferas (SphereTree)

Aunque ya se ha hablado de la jerarquía *SphereTree* en el apartado de subdivisión jerárquica espacial, ese tipo estaba basado en el octree por lo que le correspondía ese lugar en la clasificación. Sin embargo, en trabajos posteriores se mejoró la forma de construir el árbol orientando más la jerarquía al objeto y no al espacio (Del Pobil y Serna 1994, Hubbard 1995b, Dingliana y O'Sullivan 2000). Para crear la jerarquía primero se construye el *Medial Axis*, una representación "esqueleto" del objeto, y después se recubre ese esqueleto con esferas. De esta manera se consigue ajustar mejor el recubrimiento. En la Figura 1.22 se puede apreciar la diferencia entre esta técnica y la que utiliza el método de los octree. En sus trabajos manejan modelos que constan de miles de triángulos.



(a) (b)

Figura 1.22: Diferencia entre la SphereTree basada en el octree (a) y la SphereTree basada en el *Medial Axis* (b) (Hubbard 1996).

Para trabajar con metodología *Bottom-Up*, se construyen las esferas partiendo de los nodos hoja creando por ejemplo árboles binarios (Quinlan 1994). Primero se construye una malla regular de esferas (de idéntico tamaño) que cubren a cada polígono y cuyo centro está situado en los planos de dichos polígonos. Para construir los nodos internos del árbol se sigue un procedimiento en el que se divide el conjunto de nodos hoja en dos subconjuntos resultando un nodo padre con los dos nodos hijos correspondientes a los dos subconjuntos anteriores. Este procedimiento se aplica recursivamente hasta que los subconjuntos creados sean los nodos hoja. El resultado se puede ver en la Figura 1.23.

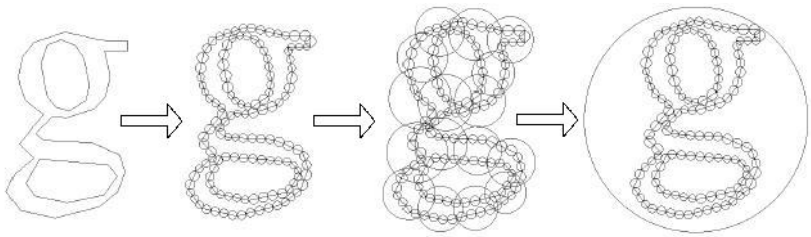


Figura 1.23 : Árbol de esferas *Bottom-Up* para un objeto (Quinlan 1994).

1.4.3.4 Árbol de Poliedros discretos orientados (k-DOP Tree)

El sistema QuickCD (Klosowski et al. 1998) construye una jerarquía de k-DOPs con metodología *Top-Down* para aproximar los modelos complejos con los que trata. Para construir el árbol utiliza características similares a las jerarquías de AABBs y de OBBs explicadas anteriormente, es decir, el plano de corte tiene en cuenta funciones dependientes del volumen de los hijos resultantes.

Zachmann (1998) utiliza también el DOP-*Tree* para demostrar su buen rendimiento (en cuanto a tiempo de ejecución y consumo de memoria) frente a la implementación de *OBBTree* usada por la librería RAPID.

En ambos trabajos se manejan entornos complejos compuestos de centenares de miles de triángulos.

1.4.3.5 Jerarquía de volúmenes para escenarios masivos

Como se ha podido comprobar en la clasificación del presente capítulo, los métodos de jerarquía de volúmenes han sido ampliamente usados en la bibliografía. El inconveniente que presentan es la baja escalabilidad que poseen. Algunas jerarquías vistas anteriormente llegan a tratar con escenarios compuestos de hasta centenares de miles de polígonos, sin embargo, si se quiere manejar escenarios masivos (el orden de magnitud mínimo de estos entornos son los millones de triángulos), estas estructuras necesitarían gigas de memoria principal.

Para solucionar este problema, algunos autores combinan un conjunto de métodos y técnicas software/hardware para conseguir grados de interactividad altos en modelos masivos de hasta 15 millones de triángulos (Wilson et al. 1999). La idea básica del sistema, llamado IMMPACT, es almacenar todo el modelo, que llega a ocupar 1.3 GB, en memoria secundaria y únicamente cargar en memoria caché, de la que disponen de 160 MB, las regiones u objetos próximos a colisionar.

Para entender esta tarea se puede observar la Figura 1.24 dónde se ve el flujo de la aplicación.

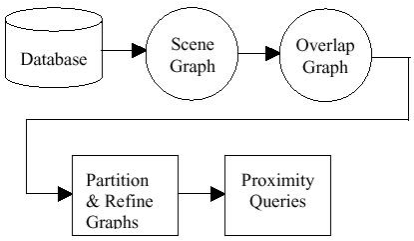


Figura 1.24: Flujo de información en la aplicación IMMPACT (Wilson et al. 1999).

Dada una geometría almacenada en disco, se calcula el grafo de la escena usando para ello AABBs para cada objeto. Los nodos de este grafo son usados para construir el *grafo de solapamiento* cuyos nodos son los objetos. Las aristas del grafo que conectan a dos nodos informan de la intersección entre los AABBs de esos nodos. Para calcular los subgrafos que finalmente serán cargados en memoria, hacen uso de algoritmos de particionamiento de grafos teniendo en cuenta si los pesos asociados a los subgrafos caben o no en la caché. El calcular subgrafos y cargarlos en memoria individualmente con el fin de

realizar la detección de colisiones en ellos tiene como finalidad explotar la localidad espacial.

1.4.3.6 Coste computacional de las jerarquías de volúmenes

Para analizar las estructuras jerárquicas de volúmenes contenedores Klosowski et al. (1998) modifican una función de coste que ya se utilizaba en trabajos previos (Gottschalk et al. 1996). Dado dos modelos con sus jerarquías construidas en una simulación con movimiento, el coste de la detección de colisiones en un instante dado viene dada por:

$$T = N_v \times C_v + N_p \times C_p + N_u \times C_u \tag{1.1}$$

Donde

T: función de coste total de la detección de colisiones.

N_V: nº de tests de solapamiento entre pares de volúmenes contenedores.

C_V: coste de testear un par de volúmenes contenedores.

N_P: número de tests de interferencia entre pares de primitivas.

C_P: coste de testear un par de primitivas.

N_U: nº de nodos que deben ser actualizados tras una transformación.

C_U: coste de actualización de cada nodo en la jerarquía.

Dada esta función de coste, y siguiendo con los criterios que se enumeraban en el apartado 1.4.1, lo ideal sería tener un volumen que maximice el criterio geométrico (para hacer disminuir a N_V y N_P), el criterio de intersección (para disminuir C_V) y el criterio de transformación (para disminuir el C_U).

Normalmente es imposible conseguir todos los objetivos simultáneamente por lo que se tendrá que escoger una solución de compromiso entre los diferentes criterios. Por ejemplo para modelos distanciados entre sí, las jerarquías de esferas y AABBs son muy eficientes, pero con modelos que están muy cerca unos de otros, N_V y N_P empiezan a tomar valores elevados. Es aquí dónde volúmenes con un Cr_G más ajustado al modelo llegan a ser más eficientes.

1.4.4 Métodos de proximidad

Siguiendo con la idea de la subdivisión espacial en la que se detectan las colisiones entre los objetos cercanos entre sí, los métodos de proximidad se

basan también en la coherencia espacial pero utilizan métodos completamente distintos.

1.4.4.1 Regiones *Voronoi*

Una línea de trabajo que utiliza la noción de distancia es la basada en las Regiones *Voronoi* que consiste en, para cada objeto, calcular mediante técnicas de geometría computacional un mapa de vecindad. Estos mapas reciben el nombre de regiones *Voronoi*. Una región *Voronoi* contiene todos los puntos del espacio que están más cerca de un objeto dado que del resto.

Para el cálculo de colisiones, se calcula una región *Voronoi* para cada elemento (vértice, arista y faceta) (Lin 1993 y Ponamgi et al. 1997). En la Figura 1.25 se puede apreciar las regiones *Voronoi* asociadas a cada elemento del objeto (R_1 es la región asociada a la faceta F_1 y R_2 es la región asociada a la arista E_2). *CP* es el *constraint plane* entre R_1 y R_2. Cada par de regiones *Voronoi* tienen un *CP* que comparten.

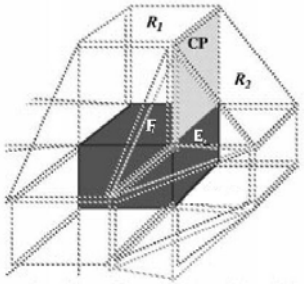


Figura 1.25: Regiones *Voronoi* (Lin 1993).

El método consiste en coger un par de elementos candidatos, uno de cada objeto a testear, y mirar sus regiones *Voronoi*. Si un elemento no está dentro de la región *Voronoi* del otro elemento, gracias al *CP* se puede conocer la región *Voronoi* vecina a testear. De esta forma el algoritmo converge hacia la solución. Esta se hallará cuando los dos elementos candidatos en ese momento, estén dentro de las regiones *Voronoi* del otro elemento respectivamente.

Para cada par de objetos en el sistema, se mantiene almacenado el par de elementos más cercanos entre sí. Los autores parten de la premisa de que los objetos se moverán muy poco entre *frame* y *frame* por lo que en sucesivos pasos, los pares de elementos más cercanos serán los mismos o algún vecino. La detección de colisiones sólo se necesitará realizar entre los objetos con regiones vecinas. Esto implica que la verificación para cada punto de un objeto se podría realizar en *O(f)* siendo *f* el número de elementos vecinos.

Este algoritmo es usado para implementar la librería de detección de colisiones I-COLLIDE (Cohen et al. 1995) disponible en la Web. Esta librería primero hace uso del algoritmo *Sweet and Prune* para descartar aquellos pares de objetos en los que sus AABBs no estén intersectándose. Después, para cada par de objetos con los volúmenes solapados, hace uso de las regiones *Voronoi* para detectar su colisión. Las escenas que maneja contienen miles de facetas teniendo en cuenta todos los objetos.

Otra librería, V-CLIP (*Voronoi Clip*), se inspira en el I-COLLIDE pero superando las limitaciones de éste en los aspectos del cálculo de la penetración, es simple de implementar y es más robusto (Mirtich 1998).

1.4.4.2 Esferas de barrido (Swept Spheres)

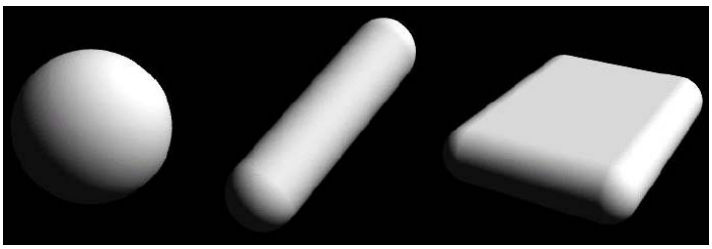
Ya que es difícil encontrar algún volumen contenedor que sea óptimo para todos los criterios, algunos trabajos optan por utilizar varios volúmenes diferentes en un mismo árbol. Un ejemplo es la librería PQP que utiliza volúmenes esfera de barrido para computar preguntas de proximidad (Larsen et al. 1999).

Estos volúmenes se construyen en base a una primitiva simple (punto, línea o rectángulo) a la cual se le aplica una suma de Minkowski, o convolución, con una esfera. En otras palabras, el volumen resultante es el conjunto de puntos barridos por una esfera cuyo centro es trasladado sobre todos los puntos de la primitiva escogida. El resultado son los siguientes tipos de volúmenes (en la Figura 1.26 se representa un renderizado de los mismos):

- Esfera de barrido con punto o PSS (*Point Swept Sphere*): la primitiva es un punto y el resultado corresponde con una esfera.

- Esfera de barrido con línea o LSS (*Line Swept Sphere*): la primitiva es una línea y por lo tanto el resultado es un cilindro con los extremos redondeados.

- Esfera de barrido con rectángulo o RSS (*Rectangle Swept Sphere*): la primitiva es un rectángulo en 3D y el resultado es una caja redondeada.



(a) (b) (c)

Figura 1.26: Diferentes volúmenes de esferas de barrido: PSS (a), LSS (b) y RSS (c) (Larsen et al. 1999).

La ventaja del uso de estos volúmenes es que el cálculo de la distancia es relativamente sencillo (basta calcular la distancia entre las primitivas y restar el radio de la esfera utilizada para el barrido). Otro factor positivo es que se puede utilizar el volumen que más se ajuste al objeto que se quiere aproximar, sus C_G son similares al del OBB.

1.4.4.3 Otros métodos

Existen otros muchos métodos en la bibliografía pero no se hablará de ellos porque quedan ya muy lejos del propósito y objetivos de esta Tesis. Algunos se basan en modelos geométricos distintos al usado en esta Tesis, otros tienen restricciones en la geometría de los objetos y necesitan por ejemplo que estos sean convexos. En esta línea están los trabajos que calculan la distancia entre poliedros para saber si dos de ellos intersectan o no (si la distancia es mayor que cero no están colisionando). Para calcular la distancia eficientemente, estos algoritmos se basan en la diferencia de Minkowski entre dos poliedros convexos (Cameron y Culley 1986, Cameron 1997, y Van Den Bergen 1999b).

Otros utilizan la restricción geométrica de la convexidad para detectar la posible intersección entre dos poliedros calculando su *plano separador* (Chung 1996) usando una versión mejorada del algoritmo de Gilbert et al. (1988) para encontrar el par de puntos más cercanos. La idea es que dos poliedros P y Q no intersectarán si y sólo si existe un plano π (plano separador) tal que todos los vértices de P estén a un lado de π y todos los vértices de Q estén en el otro lado. Como fruto de esta línea de trabajo existe la librería de detección de colisiones Q-COLLIDE.

Por último hay que citar algunos trabajos que usan la idea de un entorno de 4 dimensiones: espacio y tiempo (Cameron 1990, Hubbard 1995b, Aliyu y Al-Sultan 1999). Conociendo las velocidades de los objetos, se puede definir un volumen contenedor no sólo en un instante dado sino a lo largo del tiempo

(como si fuera una extrusión del objeto en el tiempo). Gracias a esto, se puede calcular exactamente cuándo sucederá la colisión entre los objetos.

1.4.5 Conclusiones

Como se verá en los capítulos siguientes, el método propuesto y presentado en esta Tesis pertenece a la clase de enumeración espacial. En el siguiente capítulo se comparan sus características con métodos jerárquicos del estilo del octree para justificar su elección. Sin embargo, hay que justificar de alguna manera el rechazo de los demás algoritmos vistos en este capítulo y que no se han comparado experimentalmente en la Tesis.

Los métodos jerárquicos que no se verifican numéricamente son muy utilizados en la bibliografía. Aquí no los usamos por su naturaleza jerárquica. Como primera intuición, la lógica diría que un acceso directo de una enumeración espacial siempre será más rápido que una búsqueda en un árbol. Otra diferencia radica en los factores N_u y C_u de la ecuación (1.1). En un objeto móvil al que se le ha aplicado una enumeración espacial, únicamente hay que aplicar matrices de transformación a su caja orientada (3 u 8 multiplicaciones de matriz según se veía en el apartado 1.4.1.2) pero no hace falta realizar ninguna acción sobre las celdas interiores por lo que ese coste se podría obviar. Resumiendo, la ecuación de coste quedaría de la siguiente manera:

$$T = N_v \times C_v + N_p \times C_p \tag{1.2}$$

En este caso N_v sería el número de búsquedas dentro del mallado de celdas y C_v el coste de cada búsqueda (tiempo constante según el apartado 1.4.2.1.1. Por todo esto, y sin tener en cuenta el coste de memoria, parece que la enumeración espacial es más interesante de cara al rendimiento del algoritmo.

En cuanto a los métodos de proximidad, como el basado en las regiones *Voronoi*, estos se comportan bien con objetos pequeños o en escenarios dispersos llegando a tener tiempos constantes para cada par de objetos según los autores. Sin embargo, para entornos masivos el funcionamiento no puede ser tan bueno teniendo en cuenta que primero hay que pasar la *broad phase* que puede dar un número elevado de pares de objetos en colisión si el escenario es muy compacto. Y segundo, el número de elementos (vértices, aristas y facetas) se dispara en esta clase de entornos. Además, en las simulaciones de mantenimiento, las acciones son totalmente interactivas y dependientes del usuario por lo que no es tan fácil el aprovechamiento de la coherencia temporal.

1.5 REFERENCIAS

Aliyu, M.D.S. y Al-Sultan, K.S., "Fast Collision Detection in Four-Dimensional Space", *European Journal of Operational Research*, Vol. 114, No. 2, pp. 437-445. 1999.

Amundarain, A., Borro, D., García-Alonso, A., Gil, J.J., Matey, L. y Savall, J., "Virtual Reality for Aircraft Engines Maintainability", *Proceedings of Virtual Concept 2002*, pp. 60-65. Biarritz, Francia. 9-10 Octubre 2002.

Ayala, D., Brunet, P., Juan, R., y Navazo, I., "Object representation by means of non-minimal division quadtrees and octrees", *ACM Transactions on Graphics*, Vol. 4, No. 1, pp. 41-59, 1985.

Barber, C.B., Dobkin, D.P. y Huhdanpaa, H., "The Quickhull Algorithm for Convex Hulls", *ACM Transactions on Mathematical Software*, Vol. 22, No. 4, pp. 469-483. Diciembre 1996.

Barequet, G., Chazelle, B., Guibas, L.J., Mitchell, J.S.B y Tal, A., "BOXTREE: A Hierarchical Representation for Surfaces in 3D", *Computer Graphics Forum (Proceedings of the Eurographics'96)*, Vol. 15, No. 3, pp. 387 – 396, Futuroscope – Poitiers, Francia. Agosto (26-30) 1996.

Barriuso, J.R., "Cálculo de la Distancia entre Objetos representados con precisión mediante Poliedros Cóncavos durante una Simulación Mecánica", *Tesis Doctoral*. Universidad de Navarra. Octubre 1998.

Blow, J., "Practical Collision Detection", *Proceedings of the Computer Game Developer's Conference*, 1997.

Cameron, S. y Culley, R.K., "Determining the Minimum Translation Distance between Two Convex Polyhedra", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 591-596, San Francisco, USA. Abril 1986.

Cameron, S., "Collision Detection by Four-Dimensional Intersection Testing", *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 3, pp. 291-302. Junio 1990.

Cameron, S.A., "Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra", *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3112-3117. Albuquerque, New Mexico. Abril (20-25) 1997.

Cameron, S. y Qin, C., "Motion Planning and Collision Avoidance with Complex Geometry", *Proceedings of the International Conference on Manufacturing Automation*, Hong Kong. Abril (28-30) 1997.

Chen, E., "Six Degree-of-Freedom Haptic System for Desktop Virtual Prototyping Applications", *Proceedings of the International Workshop on Virtual Reality and Prototyping*, pp. 97-106. Laval, Francia. Junio 1999.

Chung T.L., "An Efficient Collision Detection Algorithm for Polytopes in Virtual Environments", *Tesis de Master*. Universidad de Hong Kong, 1996.

Cohen, J.D., Lin, M.C., Manocha, D. y Ponamgi, M.K., "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments.", *Proceedings of ACM Interactive 3D Graphics Conference*, Vol. 1, pp. 189-196. 1995.

Del Pobil, A.P. y Serna, M.A., "A New Object Representation for Robotics and Artificial Intelligence Applications", *International Journal of Robotics & Automation*, Vol. 9, No. 1, pp. 11-21. 1994.

Dingliana, J. y O'Sullivan, C., "Graceful Degradation of Collision Handling in Physically Based Animation", *Computer Graphics Forum (Proceedings of the Eurographics 2000)*, Vol. 19, No. 3, pp. 239-247, Interlaken, Switzerland. Agosto (21-25) 2000.

Foley, J.D., Dam, A. Van, Feiner, S.K., y Hughes, J.F., "*Computer Graphics principles and practice*", Second Edition, Addison-Wesley, 1990.

García-Alonso, A., "Simulación Interactiva y Análisis de Colisiones en Mecanismos Tridimensionales con Gráficos Realistas", *Tesis Doctoral*. Universidad de Navarra. 1990.

García-Alonso, A., Serrano, N. y Flaquer, J., "Solving the Collision Detection Problem", *IEEE Computer Graphics and Applications*, Vol. 14 (3), pp. 36-43. Mayo 1994.

Gilbert, E.G., Johnson, D.W. y Keerthi, S.S., "A Fast Procedure for Computing the Distance between Complex Objects in Three Dimensional Space", *IEEE Journal of Robotics and Automation*, Vol. 4, No. 2, pp. 193-203. Abril 1988.

Gomes de Sá, A. y Zachmann, G., "Virtual reality as a tool verification of assembly and maintenance processes", *Computers & Graphics*, Vol. 23, No. 3, pp. 389-403. 1999.

Gottschalk, S., Lin, M.C. y Manocha, D., "OBBTree: A Hierarchical Structure for Rapid Interference Detection", *Proceedings of the ACM SIGGRAPH'96 – Computer Graphics*, pp. 171-180. 1996.

Hahn, J.K., "Realistic Animation of Rigid Bodies", *Proceedings of the ACM SIGGRAPH'88 – Computer Graphics*, Vol. 22, No. 4, pp. 299-308. Atlanta. Agosto (1-5) 1988.

Hayward, V., "Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Workspace", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 1044-1049, San Francisco, CA. 1986.

Held, M., Klosowski, J.T. y Mitchell, J.S.B., "Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs", *Proceedings of the Seventh Canadian Conference on Computer Geometry*, Vol. 3, pp. 205-210, Québec City, Québec, Canada. Agosto 1995.

Hubbard, P.M., "Interactive collision detection", *Proceedings of the 1993 IEEE Symposium on Research Frontiers in Virtual Reality*, pp. 24-31. Octubre 1993.

Hubbard, P.M., "Real-Time collision detection and time-critical computing", *Proceedings of the First ACM Workshop on Simulation and Interaction in Virtual Environments*, Vol. 1, pp. 92-96. Julio 1995.

Hubbard, P.M., "Collision detection for interactive graphics applications", *IEEE Transactions on Visualization and Computer Graphics*, 1 (3), pp. 218-230. Septiembre 1995.

Hubbard, P.M., "Approximating Polyhedra with Spheres for Time-Critical Collision Detection", *ACM Transactions on Graphics*, Vol. 15, No. 3, pp. 179-210. Julio 1996.

Hudson, T.C., Lin, M.C., Cohen, J., Gottschalk, S. y Manocha, D., "V-COLLIDE: Accelerated Collision Detection for VRML", *Proceedings of the second symposium on Virtual Reality Modeling Language (VRML'97)*, pp. 119-125. 1997.

Jiménez, P., Thomas, F. y Torras, C., "3D collision detection: a survey", *Computer & Graphics*, Vol. 25, No. 2, pp. 269-285. Abril 2001.

Klosowski, J.T., Held, M., Mitchell, J.S.B., y Sowizral, H., "Efficient Collision Detection Using Bounding Volume Hierarchies of *k*-DOPs", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, No. 1, pp. 21-36. Marzo 1998.

Larsen, E., Gottschalk, S., Lin, M.C. y Manocha, D., "Fast Proximity Queries with Swept Sphere Volumes", *Technical report TR99-018*, Departamento de Ciencias de la Computación, Universidad de Carolina del Norte, Chapel Hill. 1999.

Latombe, J.C., "*Robot Motion Planning"*. Kluwer Academic Publishers. 1991.

Lawlor, O.S. y Kalé, L.V., "A Voxel-Based Parallel Collision Detection Algorithm", *Proceedings of the ICS'02 (International Conference on Supercomputing)*, New York, New York, USA. Junio 22-26, 2002.

Lin, M.C., "Efficient Collision Detection for Animation and Robotics", *Tesis Doctoral*. Departamento de Ingeniería Electrica y Ciencias de la Computación, Universidad de California, Berkeley. Diciembre 1993.

Lin, M.C., Manocha, D., Cohen, J. y Gottschalk, S., "Collision Detection: Algorithms and Applications", *Proceedings of Algorithms for Robotics Motion and Manipulation*, pp. 129-142. 1996.

Lin, M.C. y Gottschalk, S., "Collision detection between geometric models: a survey", *Proceedings of IMA Conference on Mathematics of Surfaces VIII*, Birmingham, UK. 31 de Agosto – 02 de Septiembre de 1998.

Mark, W.R., Randolph, S.C., Finch, M., Van Verth, J.M. y Taylor II, R.M. "Adding Force Feedback to Graphics Systems: Issues and Solutions", *Proceedings of the ACM SIGGRAPH'96 - Computer Graphics*, Vol. 30, pp. 447-452. New Orleans, USA. Agosto (4-9) 1996.

McNeely, W.A., Puterbaugh, K.D. y Troy, J.J., "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", *Proceedings of the ACM SIGGRAPH'99 - Computer Graphics*, pp. 401-408. Los Angeles, California, USA. Agosto 1999.

Mirtich, B., "V-Clip: Fast and Robust Polyhedral Collision Detection", *ACM Transactions on Graphics*, Vol. 17, No. 3, pp. 177-208. Julio 1998.

Mirtich, B., "Timewarp Rigid Body Simulation", *Proceedings of the ACM SIGGRAPH'00 - Computer Graphics*, pp. 193-200, New Orleans, USA. 2000.

Moore, M. y Wilhelms, J., "Collision Detection and Response for Computer Animation", *Proceedings of the ACM SIGGRAPH'88 - Computer Graphics*, Vol. 22, No. 4, pp. 289-298. Atlanta. Agosto (1-5) 1988.

Naylor, B., Amanatides, J. y Thibault, W., "Merging BSP Trees Yields Polyhedral Set Operations", *Proceedings of the ACM SIGGRAPH'90 - Computer Graphics*, Vol. 24, No. 4, pp. 115-124. Agosto (6-10) 1990.

O'Sullivan, C., Dingliana, J., Ganovelli, F. y Bradshaw, G., "Collision Handling for Virtual Environments", *Eurographics 2001 Tutorial Proceedings*, 2001.

Palmer, I.J. y Grimsdale, R.L., "Collision Detection for Animation using Sphere-Trees", *Computer Graphics Forum*, Vol. 14, No. 2, pp. 105-116. 1995.

Pentland, A.P., "Computational Complexity Versus Simulated Environments", *Computer Graphics Forum*, Vol. 22, No. 2, pp. 185-192. 1990.

Ponamgi, M.K., Manocha, D. y Lin, M.C., "Incremental algorithms for collision detection between solid models", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 1, pp. 51-64. 1997.

Quinlan, S., "Efficient Distance Computation between Non-Convex Objects", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 4, pp. 3324-3329. San Diego, CA, USA. 1994.

Renz, M., Preusche, C., Pötke, M., Kriegel, H.-P. y Hirzinger, G. "Stable Haptic Interaction with Virtual Environments Using an Adapted Voxmap-PointShell Algorithm", *Proceedings of the International Conference Eurohaptics'01*. 2001.

Requicha, A.A.G., "Representations for rigid solids: Theory, methods and systems", ACM Computing Surveys, Vol. 12, pp. 437-464. December 1980.

Ritter, J., "An Efficient Bounding Sphere", *Graphics Gems I*, Academic Press, pp. 301-303, 1990.

Rossignac, J., Megahed, A. y Schneider, B., "Interactive Inspection of Solids: Cross-sections and Interferences", *Proceedings of the ACM*

SIGGRAPH'92 - Computer Graphics, Vol. 26, No. 2, pp. 353-360. Julio (26-31) 1992.

Rozas, O., "Detección de Colisiones en Simulación de Mecanismos con un Modelo de Representación basado en Superficies NURBS", *Tesis Doctoral*. Universidad de Navarra. Mayo 1998.

Rubí, J., "Cinemática, Dinámica y Control de Robots Redundantes y Robots Subactuados", *Tesis Doctoral*. Universidad de Navarra. Octubre 2002.

Sapidis, N.S., y Perucchio, R. "Domain Delaunay tetrahedrization of solid models", *International Journal of Computational Geometry and Applications*, Vol. 1, No. 3, pp. 299-325. 1991.

Sapidis, N.S., y Perucchio, R. "Delaunay triangulation of arbitrarily shaped planar domains", *Computer Aided Geometry Design*, Vol. 8, No. 6, pp. 421-437. Diciembre 1991.

Savall, J., Borro, D., Gil, J.J. y Matey, L., "Description of a Haptic System for Virtual Maintainability in Aeronautics", *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2887-2892, EPFL, Lausanne, Switzerland. Septiembre (30) – Octubre (4) 2002.

Selim, S.Z. y Almohamad, H.A., "Collision computation of moving bodies", *European Journal of Operational Research*, Vol. 119, No. 1, pp. 121-129. 1999.

Sigal, A., Gil, M. y Ayellet, T., "Deferred, Self-Organizing BSP Trees", *Computer Graphics Forum (Eurographics'02)*, Vol. 21, No. 3, pp. 269-278, Saarbrücken, Germany. Septiembre (2-6) 2002.

Shaffer, C.A. y Herb, G.M., "A Real-Time Robot Arm Collision Avoidance System", *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 2, pp. 149-160. Abril 1992.

Smith, A., Kitamura, Y., Takemura, H. y Kishino, F., "A Simple and Efficient Method for Accurate Collision Detection among Deformable Polyhedral Objects in Arbitrary Motion", *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pp. 136-145, North Carolina, USA. Marzo(11-15) 1995.

Thompson II, T.V., Nelson, D.D., Cohen, E. y Hollerbach, J.M. "Maneuverable NURBS Models Within A Haptic Virtual Environment", *Proceedings of the 1997 ASME International Mechanical Engineering Congress and Exhibition*, Vol. 61, pp. 37-44. Dallas. 1997.

Van Den Bergen, G., "Efficient Collision Detection of Complex Deformable Models using AABB Trees", *Journal of Graphics Tools (JGT)*, Vol. 2, No. 4, pp. 1-13. 1997.

Van Den Bergen, G., "Collision Detection in Interactive 3D Computer Animation", Tesis Doctoral. Departamento de Matemáticas y Ciencias de la Computación. Universidad de Tecnología de Eindhoven. 1999.

Van Den Bergen, G., "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects", *Journal of Graphics Tools (JGT)*, Vol. 4, No. 2, pp. 7-25. Julio 1999.

Vemuri, B.C., Cao, Y., y Chen, L., "Fast Collision Detection Algorithms with Applications to Particle Flow", *Computer Graphics Forum (Proceedings of the Eurographics'98)*, Vol. 17, No. 2, pp. 121-134. Junio 1998.

Volino, P., y Magnenat-Thalmann, N., "Interactive Cloth Simulation: Problems and Solutions", *Proceedings of the JWS'97*. 1997.

Wilson, A., Larsen, E., Manocha, D. y Lin, M.C., "Partitioning and Handling Massive Models for Interactive Collision Detection.", *Computer Graphics Forum (Eurographics'99)*, Vol. 18, No. 3, pp. 319-329, Milan, Italia. Septiembre (7-11) 1999.

Zachmann, G., "Real-Time and Exact Collision Detection for Interactive Virtual Prototyping", *Proceedings of DETC'97 (ASME Design Engineering Technical Conferences)*, Septiembre 14-17, Sacramento, California. 1997.

Zachmann, G., "Rapid Collision Detection by Dynamically Aligned DOP-Trees", *Proceedings of VRAIS'98 (Virtual Reality Annual International Symposium)*, Atlanta, Georgia. Marzo 1998.

Zachmann, G., "Virtual Reality in Assembly Simulation – Collision Detection, Simulation Algorithms, and Interaction Techniques", *Tesis Doctoral*. Universidad Técnica de Darmstadt. Julio 2000.

Zhang, D. y Yuen, M.M.F., "A Coherence-based Collision Detection Method for Dressed Human Simulation", *Computer Graphics Forum*, Vol. 21, No. 1, pp. 33-42. 2002.

CAPÍTULO 2

ANÁLISIS DEL PROBLEMA

2.1 INTRODUCCIÓN

La detección de colisiones forma parte de lo que a menudo es llamado *detección de interferencias* o *detección de intersecciones*, la cual puede ser dividida en tres importantes tareas o subproblemas: *detección de colisión*, *determinación del área de contacto* y *respuesta de colisión*. La *detección de colisión* comprueba si dos objetos colisionan entre sí y la determinación del área de contacto encuentra o calcula la intersección entre el par de objetos. La respuesta de colisión es una parte más independiente y se encarga de determinar las acciones que deberían ser aplicadas a los objetos como respuesta a su colisión. Los dos primeros problemas están muy relacionados entre sí pero el último es dependiente de la aplicación y se tratará con detalle en el capítulo 5.

En el presente capítulo, en primer lugar se enunciará el problema al que hay que enfrentarse. El tipo de aplicación es muy concreto: escenarios masivos y muy densos dónde se requiere un nivel de interactividad alto, lo suficiente para simular tareas reales de mantenimiento. Seguidamente se analizan distintas posibilidades de afrontar el problema a partir de la clasificación descrita en el anterior capítulo y se aprovecha para asentar las bases del método de detección de colisiones utilizado en la Tesis. Finalmente, las ideas base del método son validadas mediante la comparativa realizada entre cuatro métodos distintos que combinan técnicas de enumeración espacial con subdivisiones jerárquicas.

2.2 DESCRIPCIÓN GENERAL DEL PROBLEMA

EL presente trabajo se ha centrado en modelos geométricos representados mediante B-Rep. Es decir, para un objeto 3D se posee una lista de triángulos que definen su frontera. No obstante, los formatos de fichero soportados son varios y algunos pueden tener definidos polígonos de más de tres lados, por lo que para generalizar el problema se ha desarrollado un algoritmo de triangularización, el cual se ejecuta en un preproceso antes de mostrar la escena gráfica. Existen numerosos modelos geométricos, ya Foley et al. (1990) recopilaba una interesante clasificación que ha sido extendida por diversos autores. Se ha tomado como punto de partida el modelo poliédrico por razones prácticas: es actualmente el modelo más común disponible en los sistemas CAD para exportar descripciones geométricas.

El problema de mantenibilidad necesita un módulo de colisiones eficaz sobre maquetas de hasta millones de triángulos con el objetivo de simular interactivamente tareas reales de mantenimiento. Además, se necesita ofrecer al usuario una sensación de tacto. Este feedback es la clave para simular tareas lo más realisticamente posible. Para lograrlo, el sistema posee un dispositivo háptico que genera una fuerza dependiente del contacto. Esta clase de entornos es muy diferente a otros ya conocidos en el área de colisiones como son el *path finding*, robótica, vehículos, *virtual walkthrough* y diseño de mecanismos.

Por características propias del tipo de aplicación, en una simulación se pueden distinguir tres tipos de objetos:

- ***Maqueta virtual u objeto estático***: este objeto está compuesto a su vez por miles de objetos y todos ellos componen lo que se conoce como la maqueta virtual del motor aeronáutico. En el ámbito bibliográfico, esta clase de modelos recibe el nombre de modelos masivos (*massive models*) por su gran número de polígonos que pueden alcanzar ordenes de millones. Un ejemplo gráfico de la complejidad que poseen las maquetas típicas se muestra en la Figura 2.1.

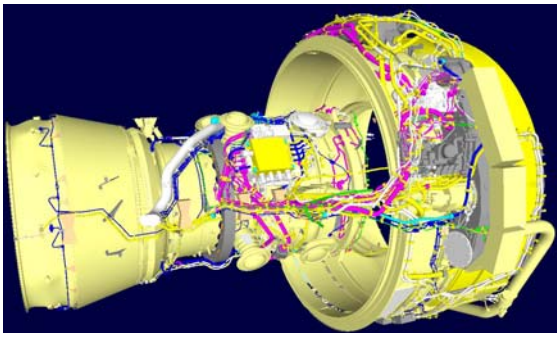


Figura 2.1: Modelo 3D de una maqueta representando un motor aeronáutico.

- ***Herramienta u objeto móvil***: la mano o la herramienta del operario se inserta en la escena virtual a través de un modelo virtual (dependiendo de qué elemento se quiera manipular, la herramienta cambiará). Su modelo está constituido por unos cientos o miles de triángulos.

- ***Objeto desensamblado***: pueden ser muchos pero en un momento dado sólo existe uno. Corresponde a un objeto que se ha desmontando del motor y que permanece unido a la herramienta. Su tamaño depende del objeto en cuestión: se pueden desmontar desde tornillos y tuercas hasta tubos o componentes enteros.

Para resumir, el problema de colisiones que se aborda tiene dos características específicas. La primera concierne al número de objetos en la escena. El entorno estático está compuesto por un número elevado de objetos, mientras que el número de objetos móviles es muy reducido (la herramienta y algún objeto desensamblado). Estos objetos móviles representan aproximadamente sólo un 0.5% del número total de triángulos en la escena. La segunda característica se refiere a la posición relativa entre las geometrías de los objetos estático y móvil: algunos triángulos del objeto móvil estarán siempre muy cercanos al conjunto de triángulos del objeto estático.

Una vez descritos los diferentes objetos, hay que analizar sus intersecciones. El conocido problema de la explosión combinatoria aparece al tener que comparar todos los polígonos de un cuerpo con todos los del otro para encontrar la intersección.

Si se quiere realizar una búsqueda exhaustiva de las intersecciones entre los objetos “estático-móvil”, el método más directo para determinar la intersección es la comparación entre todas las posibles combinaciones de polígonos de la herramienta y polígonos de la maqueta. Esta técnica de fuerza

bruta sólo es viable en aplicaciones con modelos muy pequeños o en métodos no interactivos. Para demostrar su ineficacia en escenarios relativamente más grandes, se puede tomar como ejemplo los órdenes de magnitud en los que se mueve la aplicación considerada (millones de polígonos para el objeto estático y miles para el móvil). Si consideramos que el cálculo de una intersección entre dos triángulos tarda un tiempo aproximado de $10\,\mu s$ [2], entonces el tiempo que se tardaría en realizar un cálculo de colisiones en un momento dado sería:

$$t = 10^6 \times 10^3 \times 10^{-5} = 10^4 s \approx 2.7 horas \qquad (2.1)$$

Es obvio que se necesita alguna técnica para reducir este tiempo. La mayoría de los cálculos son innecesarios ya que en un instante *t* de tiempo dentro de la simulación, el conjunto real de triángulos intersectándose será mucho menor que el escogido para el ejemplo anterior. Algo similar ocurre en el mundo real. Si una persona necesita conocer si dos objetos están colisionando, primero se parte de una observación inicial de las posiciones de los objetos y se analiza qué partes de esos objetos están cercanas entre sí descartando mediante una simple inspección visual las zonas en las que se puede asegurar que no existe colisión. Finalmente se puede observar con más detenimiento las zonas conflictivas.

Algo parecido usan los métodos de detección de colisiones. Se genera una estructura de datos auxiliar a la geometría que ayuda a centralizar la atención en la zona o zonas donde realmente están cercanos los objetos para posteriormente realizar el cálculo exacto en esas zonas. Por lo tanto, y fijándose en la clasificación realizada en el capítulo anterior, el método propuesto tendrá que tener sus bases en una de las técnicas dentro de los tres grandes grupos de métodos: los métodos de subdivisión espacial, los métodos de jerarquía de volúmenes o los métodos de proximidad. Hay que recordar que uno de los objetivos del método era el cálculo de las colisiones sin tener en cuenta ningún tipo de restricción geométrica de los objetos por lo que el método deberá ser de propósito general.

Al final del capítulo anterior se daban las posibles razones por las que las técnicas de partición espacial parecían más atractivas que las jerarquías de volúmenes. Y dentro de la subdivisión espacial, la enumeración uniforme era la técnica más interesante gracias al acceso directo que posee. En el siguiente

[2] Tiempo calculado a partir del algoritmo de Möller usado para detectar la intersección entre dos triángulos (Möller 1997). Para un procesador Pentium 3 Xeon a 866 MHz, el algoritmo ejecuta 10^5 testeos en un segundo.

apartado se analiza esta técnica junto con la más conocida dentro de la subdivisión jerárquica: el octree.

2.3 ANÁLISIS DE TÉCNICAS DE PARTICIÓN ESPACIAL

Partiendo de un modelo de representación de objetos definido por facetas, una estructura de partición espacial brinda la información necesaria para acelerar la detección de interferencias entre distintos objetos. En líneas generales, estas técnicas se encargan de subdividir o particionar el espacio tridimensional y realizar un primer cálculo entre esas particiones. Una vez se tiene el conjunto de particiones colisionando entre sí, se puede pasar a un nivel de precisión más elevado. Un esquema general de la resolución del problema utilizando técnicas de partición espacial se puede ver en la Figura 2.2. Para las dos últimas fases se utiliza la nomenclatura usada por Hubbard (1993).

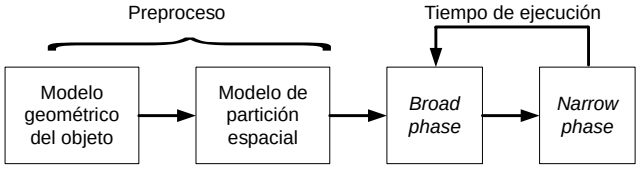


Figura 2.2: Esquema general de la resolución del problema de colisiones utilizando métodos de partición espacial.

A partir de la geometría original del modelo, en este caso los objetos son representados mediante facetas, se genera una estructura auxiliar que será la base de información utilizada por el módulo de colisiones. Este paso sólo es necesario ejecutarlo una única vez y en tiempo de preproceso. Ya en ejecución, el planteamiento es dividir el problema en dos: una primera fase en la que se descartan zonas de trabajo (*broad phase*) y otra en la que se testean los pares de triángulos obtenidos en el paso anterior (*narrow phase*).

Como punto de partida se analizan en detalle dos técnicas pertenecientes a los métodos de subdivisión o partición espacial: una solución enumerativa con voxels y otra jerárquica con octrees. En la bibliografía existen antecedentes en el uso de voxels para simulación de tareas de mantenimiento dentro de escenarios virtuales masivos (McNeely et al. 1999). Para solucionar el inconveniente de la enumeración espacial relativo al consumo de memoria, la alternativa es el uso de una enumeración jerárquica siendo la más extendida el octree, sobre todo en sistemas con un objeto móvil contra un escenario estático como es el caso de la robótica (Hayward 1986 y Shaffer y Herb 1992).

Se pueden considerar dos alternativas al almacenamiento contenido en cada partición: almacenar objetos o almacenar polígonos. Asimismo, en los siguientes apartados que describen las dos técnicas de subdivisión, se valorarán dos aspectos relevantes en este tipo de métodos: el tiempo de ejecución en el cual habrá que optimizar la *broad* y *narrow phase*, y el consumo de memoria requerido.

Para determinar el grado de eficiencia de un algoritmo existen distintas estrategias.

- La aproximación empírica (o a posteriori): consiste en programar los diferentes algoritmos y después probarlos con distintas entradas.

- La aproximación teórica (o a priori): consiste en determinar matemáticamente la cantidad de recursos (tiempo de ejecución, espacio en memoria, etc.) que necesitará el algoritmo en función del tamaño de las entradas consideradas.

En este capítulo se usará la aproximación teórica intentando analizar teóricamente el comportamiento de los diferentes algoritmos. Hay que tener en cuenta las limitaciones de este estudio: los algoritmos son muy dependientes del problema a resolver por lo que es muy difícil realizar un estudio teórico exhaustivo para todos los posibles casos.

Al final de este capítulo y en posteriores, se hará uso de la aproximación empírica para probar y comparar los algoritmos con casos reales.

2.3.1 Técnica basada en enumeración espacial (voxels)

La estructura de voxels consiste en un conjunto de celdas iguales y unidas entre sí, disjuntas y ordenadas en una malla tridimensional regular. La idea básica es muy similar a la ya utilizada en trabajos previos, tanto en colisiones (García-Alonso et al. 1994) como en el problema del cálculo de distancias (Barriuso 1998).

Para el presente análisis, y como primera aproximación, se considerará un espacio de trabajo cúbico de 1 m^3 y se realizará el mismo número de subdivisiones en cada eje. De este modo, la estructura general usada por el módulo de colisiones constará de una matriz tridimensional con $N_i x N_j x N_k$ celdas (en este caso $N_i=N_j=N_k$) que apuntarán a listas de objetos/polígonos contenidos parcial o totalmente en ellas (Figura 2.3).

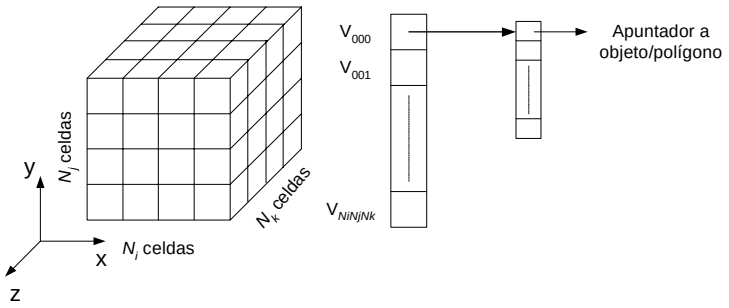


Figura 2.3: Estructura de voxels en 3D y su representación en memoria.

Las distintas subdivisiones se utilizan como una herramienta para reducir la complejidad del problema, pero en ningún momento sustituyen al modelo geométrico original.

Para el consumo de memoria no se ha tenido en cuenta la memoria usada para el almacenamiento de los propios objetos o triángulos como son sus coordenadas 3D, sus cajas contenedoras o sus matrices de trasformación. Estos datos son almacenados por el módulo gráfico y su acceso es compartido por el módulo de Colisiones. De este modo se puede valorar exclusivamente cuál es la cantidad de memoria que se debe consumir para la detección de colisiones.

Como el número de polígonos del objeto estático es tres órdenes de magnitud superior al del móvil, como simplificación tampoco se tendrá en cuenta la memoria consumida por el objeto móvil considerando este objeto como un conjunto de triángulos sin modelo de partición espacial asociado.

2.3.1.1 Voxels de objetos

Los voxels de objetos son usados a menudo en aplicaciones *n*-body (Lawlor y Kalé 2002) dónde interesa conocer en cada momento qué objetos están cercanos entre sí. Pero en problemas *n*-body estáticos como los que se estudian en esta Tesis también pueden implementarse, aunque presenten algún problema que se mencionará enseguida.

Tiempo de ejecución

Para determinar el orden de un algoritmo (tiempo que emplea ese algoritmo en ejecutarse) hay que fijarse en el número de accesos realizados hasta alcanzar el grado de precisión requerido. En este capítulo se analiza de una manera muy teórica el orden de los algoritmos.

En el caso de la solución enumerativa, tal y como se mostraba en la Figura 1.11, el acceso a una celda del mallado es directo (*O(1)*) y conseguir la

lista de objetos asociados a esa celda es un nivel de indirección más, por lo que ambos accesos tienen orden constante. Una vez se tiene el/los objeto/s estático/s en colisión con el objeto móvil, se pasaría al siguiente nivel de precisión en el que se comprueba la intersección entre las diferentes facetas. El paso a este nivel de detalle es necesario siempre y cuando se quiera llegar al máximo nivel de precisión que corresponde al de la geometría. El tiempo de ejecución queda por tanto especificado de la siguiente manera por la ecuación (2.2).

$$O(t_{ejec}) = O(1) + O(v_o) \times O(f_e f_m) = O(v_o f_e f_m) \qquad (2.2)$$

Donde v_o es el número de voxels estáticos ocupados por el AABB del objeto móvil (a partir de ahora se les llamará voxels objetivo, ver Figura 2.4b), f_m es el número de facetas del objeto móvil y f_e es el número medio de facetas de los objetos estáticos contenidos en cada voxel objetivo (Nota: si un objeto tiene 50 facetas y otro tuviese 800, el número de facetas asociado a ese voxel sería 850).

A simple vista puede parecer que el término v_o es muy pequeño en comparación con el resto de factores. Pero no se puede despreciar ya que, como se comprobará más adelante, puede llegar a darse el caso de que este término crezca desmesuradamente (sobre todo si el mallado es muy fino).

Esta solución presenta un problema que se describe a continuación. En la Figura 2.4 (a) se muestra un mallado de voxels conteniendo una escena compuesta de un único objeto **A**. Si se considera un objeto móvil **B,** entonces se calcula el conjunto de voxels objetivo que intersecta con el objeto móvil (Figura 2.4 (b)).

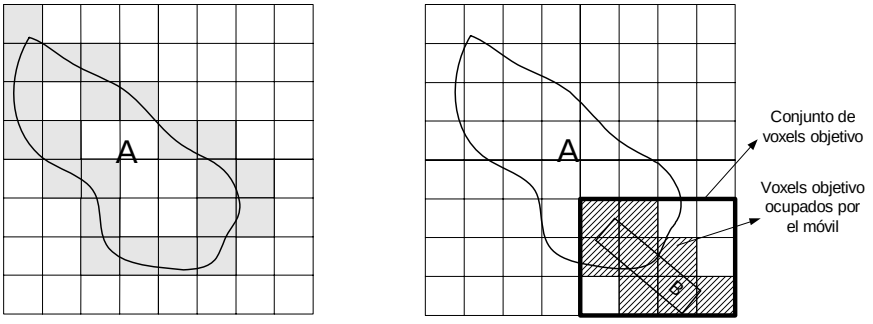


Figura 2.4: En (a) se ilustra un mallado de voxels donde algunos apuntarán al objeto A y en (b) se muestran los voxels objetivo y los ocupados.

El problema surge si se quiere calcular exactamente la zona de contacto. Con esta solución se conoce el par de objetos que intersectan entre sí (hay que

recordar que los voxels únicamente apuntan a objetos), por lo que para calcular la zona de contacto será necesario mirar todos los polígonos de los dos objetos. Con el ejemplo de la figura se aprecia que se realizarían muchos más cálculos de los necesarios ya que lo que realmente está en contacto son los extremos de los objetos. Este problema se puede solucionar con voxels apuntando a conjuntos de triángulos en lugar de a objetos.

Memoria

Si se considera que un puntero ocupa 4 bytes de memoria, N es el número de celdas en cada eje y n_p es el número total de apuntadores a objetos, el coste de memoria C_{mem} viene dado por la fórmula (2.3).

$$C_{mem} = N^3 \times 4 + n_p \times 4 \tag{2.3}$$

El factor n_p no se conoce a priori pero su peso respecto a N^3 es muy pequeño pudiéndose despreciar para analizar la fórmula. Dependiendo de la precisión requerida, se podría tener uno de los siguientes casos:

Precisión fina (1 mm) → $C_{mem} \approx 4 \times 10^9$ bytes

Precisión media (1 cm) → $C_{mem} \approx 4 \times 10^6$ bytes

Precisión gruesa (1 dm) → $C_{mem} \approx 4 \times 10^3$ bytes

Aunque sea una aproximación, sirve de comprobación que con una precisión elevada se manejaría cifras de memoria del orden de Gigabytes, mientras que eligiendo una precisión media o pequeña se tendría Megabytes o Kilobytes respectivamente. La primera opción se rechaza por el consumo de memoria que requiere. Tampoco merece la pena ahorrar memoria eligiendo una precisión muy gruesa. El tamaño de los voxels sería muy grande y se perdería la ventaja de subdividir el espacio con el objetivo de discriminar cálculos innecesarios.

2.3.1.2 Voxels de facetas o triángulos

La Figura 2.5 presenta la estructura de los voxel de facetas y se pueden apreciar los cambios respecto a los voxels de objetos (Figura 2.3).

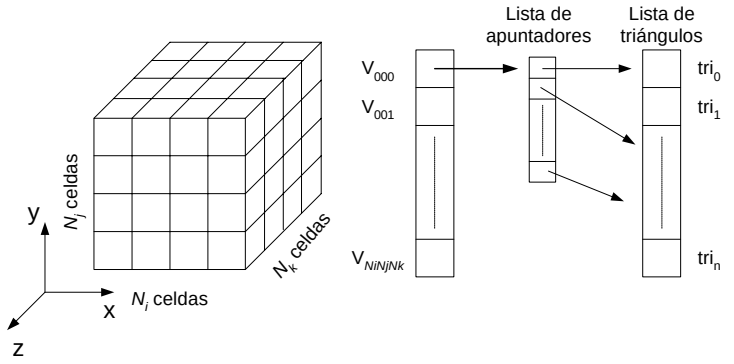


Figura 2.5: Estructura de voxels de facetas en 3D y su representación en memoria junto con la estructura de triángulos propia del módulo de colisión.

Tiempo de ejecución

Como se verá, esta solución consume más memoria pero repercute favorablemente en el tiempo de ejecución ya que ahora únicamente se testean los triángulos que realmente están en colisión o cerca de estarlo (resolviéndose el problema de los voxels de objetos anteriormente comentado). El tiempo de ejecución queda reflejado por la fórmula (2.4).

$$O(t_{ejec}) = O(1) + O(v_o) \times O(f_{ve}\, f_m) = O(v_o\, f_{ve}\, f_m) \tag{2.4}$$

Ahora f_{ve} es el número medio de facetas estáticas presentes en un voxel ocupado. Estas facetas serán únicamente las que estén en colisión o cerca de estarlo por lo que $f_{ve} << f_e$.

Memoria

En este caso el C_{mem} vendría dado por la fórmula (2.5).

$$C_{mem} = 4 \times N^3 + 4 \times n_p + 46 \times n_{tri} \tag{2.5}$$

N es el número de celdas en cada eje, n_p en este caso será el número total de apuntadores a facetas. Como se explicará más adelante, a cada triángulo se le dota de una estructura que consume 46 bytes (las coordenadas 3D están en el módulo de Visualización).

Aunque n_p tampoco es fácil de estimar en este caso, con esta solución se tendrían los mismos casos de precisión que se manejaban antes. El resultado sería el mismo, para precisiones muy altas (del orden de milímetros) el coste de memoria sería excesivo. De todas formas, la precisión utilizada con los voxels

de facetas debería ser mayor que con los voxels de objetos. La razón se debe a que con los objetos no hace falta ajustar tanto ya que al final siempre se mirarán todos los triángulos de ese objeto. Sin embargo, con los voxels de facetas, el objetivo es discriminar al máximo el número de triángulos. Esto quiere decir que el factor N^3 será bastante menor en el caso de tener objetos.

Aunque el uso óptimo de los voxel-faceta requiere más memoria que los voxel-objeto, desaparece el problema mostrado en la sección anterior ya que con voxel-faceta, únicamente se comprobará la intersección entre los triángulos apuntados por los voxels objetivo y no del resto del objeto.

La memoria que ocupa cada elemento de la lista de triángulos, en este caso 46 bytes, puede parecer excesiva pero como se comprobará en el capítulo siguiente, se guarda tanta información por cada triángulo para que la posterior comprobación en tiempo de ejecución sea lo más eficiente posible. Es un claro ejemplo del compromiso tiempo/memoria. Con la solución de voxels apuntadores a facetas, se apuesta por un menor tiempo de ejecución aunque ello implique un mayor coste de almacenamiento.

Resumiendo, las ventajas de utilizar una solución enumerativa basada en voxels para implementar un sistema de colisiones son:

- El nivel de subdivisión del modelo de partición espacial no limita en ningún caso la precisión del cálculo. Únicamente condiciona el tiempo de computación.

- Acceso al espacio objetivo en tiempo constante por lo que el tiempo de ejecución sólo depende del número de voxels y facetas implicadas.

- Al tener fijo el número de celdas en las tres dimensiones, la creación y manejo del array3D que represente el mallado de voxels es muy sencillo.

Como toda solución, también ésta presenta una serie de inconvenientes que se exponen a continuación:

- Al tener fijo el número de celdas en las tres dimensiones, el consumo de memoria puede llegar a ser excesivo si se quiere alcanzar una precisión establecida.

- Además y ligado a lo anterior la mayoría de esas celdas estarán vacías y sin uso.

- El cálculo de un nivel de voxelizado óptimo es una tarea complicada. Si se elige un mallado grueso se puede estar desperdiciando la propiedad de localidad espacial del método, y si por el contrario se elige un nivel muy fino entonces se corre el riesgo de consumir demasiada memoria y además también penalizar la eficiencia del algoritmo.

2.3.2 Técnica basada en subdivisión espacial jerárquica (octrees)

Las representaciones jerárquicas son una variante del anterior caso. También utilizan primitivas para particionar el espacio pero no lo hacen regularmente sino que éste se subdivide recursivamente hasta que se alcanza un nivel de subdivisión (precisión) adecuado.

Una posible estructura en memoria es la mostrada en la Figura 2.6.

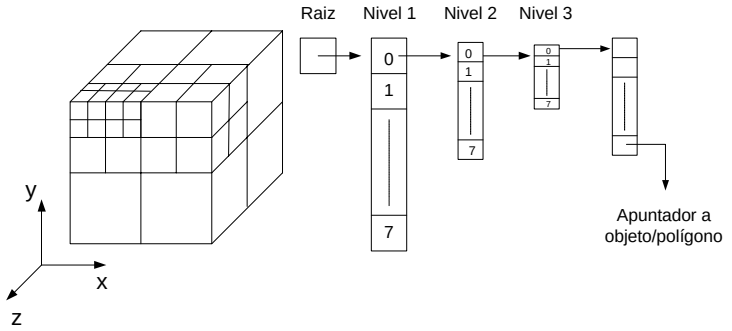


Figura 2.6: Estructura del octree en 3D y su representación en memoria.

Al igual que los voxels, los nodos hoja de un octree mantendrán una lista a los objetos o a los polígonos contenidos en ese nodo (en el caso de polígonos, el octree tendrá un nivel más de indirección apuntando a una lista de triángulos propia del módulo de colisiones).

2.3.2.1 Octree de objetos

Como ya se comenta en la clasificación del capítulo anterior, la ventaja de tener un octree de objetos radica en el menor tamaño y mejor compactación inherente a la estructura árbol permitiendo rápidas actualizaciones si existe movimiento de objetos. En aplicaciones dónde no se requiere tanta precisión, el octree de objetos ofrece buen rendimiento al no tener que pasar al nivel de triángulos; sin embargo en lo que a mantenibilidad respecta, la precisión tiene que ser como

mínimo la de la aproximación poliédrica por lo que siempre habrá que acceder a la misma.

Tiempo de ejecución

Al contrario que la solución enumerativa, el cálculo del conjunto de celdas objetivo ya no es directo. Al tener la estructura representada mediante un octo-árbol, es necesario realizar una búsqueda sobre el mismo cada vez que se quiera localizar en qué celda se encuentra un punto dado. En el peor caso, las búsquedas en un árbol son de orden logarítmico al número de nodos hoja, es decir, la profundidad del árbol. El tiempo de ejecución se refleja en la fórmula (2.6):

$$O(t_{ejec}) = O(p) + O(o_o) \times O(f_e f_m) = O(p + o_o f_e f_m) \qquad (2.6)$$

Siendo p la profundidad del árbol. Los demás factores son los mismos que se tenían con la solución enumerativa (o_o es similar a v_o pero en este caso se refiere a celdas o nodos hoja del octree y no a voxels). Hay que tener en cuenta que el uso de voxels o de octrees afecta sólo al tiempo de búsqueda de los voxels/hojas ocupados por el objeto estático y el móvil. Una vez encontrados, sigue siendo necesario acceder a facetas para calcular un contacto exacto. La ventaja de la solución jerárquica es el menor consumo de memoria. Por otra parte, con la solución orientada a objetos se seguiría teniendo el mismo problema que con los voxel-objetos.

Memoria

La medida del consumo de memoria en el caso de los octrees es un poco más complicada de establecer debido a su carácter recursivo y dependiente del modelo, aunque el consumo de memoria total es mucho mayor en los voxels.

Para dar una estimación del número de celdas usadas en la estructura de árbol, primero hay que definir hasta que nivel se llegará en la recursión. Se pueden utilizar los mismos ejemplos de precisión usados en el caso de la enumeración espacial. En este caso tener una precisión del milímetro no significa tener 1000 hojas en cada dimensión, 10^9 en total, pero en modelos muy densos podría ocurrir que sí.

Numerando el nodo raíz como nivel 0, cada nivel tendrá 8^{nivel} nodos, lo que significa que puede haber como máximo (si el modelo es tan denso que no hay nodos vacíos) 2^p celdas en cada dimensión, siendo p la profundidad del árbol. Por lo tanto, en este peor caso la memoria consumida por los nodos del árbol vendría dada por la fórmula (2.7).

$$C_{mem-nodos} = 4 \times \sum_{i=0}^{p} 8^p \tag{2.7}$$

A priori parece que esta solución es más costosa que el utilizar voxels porque guarda en memoria los nodos hoja (representación equivalente a los voxels) más todo el conjunto que representa a los nodos interiores del árbol (sumatorio desde 0 a *p*-1). Pero este caso es prácticamente imposible que suceda ya que si se opta por la solución de octrees significa que se tiene conocimiento de la existencia de volumen vacío en la escena por lo que nunca se llegará a tener tanta cantidad de nodos. La división del espacio será más parecida a la de la Figura 2.6, es decir, celdas concentradas en partes del modelo y otras celdas grandes representando espacios vacíos.

Ya que el número final de nodos depende del modelo, no se puede dar una expresión más exacta para el consumo de memoria que la ofrecida en (2.8).

$$C_{mem} = C_{mem-nodos} + 4n_p \tag{2.8}$$

Donde $C_{mem\text{-}nodos}$ para una profundidad p venía dada por la ecuación (2.7).

2.3.2.2 Octree de facetas

El octree de polígonos es mejor si lo que se necesita es precisión en el cálculo y se quiere evitar el tener que comparar más polígonos de los necesarios.

Tiempo de ejecución

La fórmula (2.9) del tiempo de ejecución queda de la misma forma que en (2.4) pero modificado el factor correspondiente al tiempo de búsqueda del conjunto de celdas objetivo.

$$O(t_{ejec}) = O(p) + O(o_o) \times O(f_{he} f_m) = O(p + o_o f_{he} f_m) \tag{2.9}$$

Como en el caso de los voxels de facetas, f_{he} es el número de facetas pertenecientes a los nodos hoja objetivo. Estas facetas serán únicamente las que estén en colisión o cerca de estarlo por lo que $f_{he} << f_e$.

Memoria

La expresión (2.10) quedaría de la misma forma que en (2.5) salvo por el factor $C_{mem\text{-}nodos}$ que depende del modelo.

$$C_{mem} = C_{mem-nodos} + 4n_p + 46n_{tri} \tag{2.10}$$

Resumiendo, las ventajas de utilizar una solución jerárquica son:

- Ahorro considerable de memoria al subdividir únicamente las celdas que contienen parte de la frontera del objeto.

- De este modo se puede llegar más fácilmente a un nivel de precisión establecido.

Algunos inconvenientes son:

- El tiempo de ejecución depende de un factor más que corresponde con el tiempo de búsqueda en el árbol para localizar las celdas objetivo.

- Al ser un método jerárquico, las estructuras de datos son más complicadas y difíciles de gestionar.

2.4 COMPARATIVA DE ALGORITMOS

Aunque teóricamente parece que los algoritmos de voxels son más eficientes, computacionalmente hablando, que las organizaciones en octrees, se han desarrollado los dos métodos para probarlos con casos reales. Además, también se ha implementado la posibilidad de utilizar métodos híbridos como es el caso de voxelizar la maqueta estática mientras que se utiliza un octree para el objeto móvil y viceversa: utilizar el octree para la escena estática y un mallado de voxels para el objeto móvil.

Este apartado no se limita meramente a recoger los distintos algoritmos implementados sino que se realizan comparativas de los distintos métodos y se señalan ventajas e inconvenientes de cada uno de ellos.

2.4.1 Consideraciones previas

Antes de pasar a describir y comparar los 4 algoritmos desarrollados, es necesario aclarar un par de conceptos:

- Todos los cálculos de los algoritmos son realizados sobre el sistema de coordenadas del objeto estático (maqueta) para lo cual se utilizan las coordenadas globales de la escena.

- Cuando se crea la estructura de voxels u octree para el objeto móvil, hay que tener en cuenta que este objeto puede estar orientado en cualquier dirección por lo que las celdas “móviles” son celdas

orientadas en el sistema de referencia del objeto móvil, al contrario que las celdas "estáticas" que son paralelas al sistema de referencia global. Como el algoritmo que mira qué celdas intersectan con una faceta es más rápido en el caso de cajas paralelas, es necesario una transformación del sistema de referencia móvil al estático.

- Para simplificar se ignora la memoria consumida por el objeto móvil en su modelo de partición espacial ya que el orden de magnitud de este objeto es 3 veces menor al del objeto estático.

- Las pruebas experimentales se han realizado sobre un modelo de turbina aeronáutica con 1615172 polígonos y un modelo de mano-herramienta de 2687 polígonos. Los resultados son extrapolables a cualquier otro modelo estático y móvil.

- Los tiempos han sido calculados como la media de tiempos de 10 posiciones diferentes de trabajo.

2.4.2 Método puro de voxels (MVV)

En primer lugar se analizan los resultados obtenidos con un método de voxels parecido al implementado por García-Alonso et al. (1994) el cual ofrece buenos resultados para el análisis de mecanismos.

Este primer algoritmo de voxels que se ha implementado utiliza sendos modelos de partición espacial para voxelizar tanto la maqueta estática como la herramienta (objeto móvil). Por lo tanto, el orden del algoritmo que calcula las colisiones tiene un pequeño cambio respecto a la fórmula teórica que se daba en (2.4) que se analizará a continuación.

Una vez se tiene generadas las dos mallas de voxels, para calcular las colisiones hay que pasar por varios pasos o niveles de precisión: el nivel de voxel y el nivel de triángulo. Los autores hacían un test previo a los dos mencionados que consistía en comprobar la interferencia entre los AABBs de los dos objetos. Este test es muy necesario en aplicaciones de tipo *n*-body para descartar pares de objetos pero en el presente caso no se aplica ya que en cualquier operación el resultado va a ser siempre positivo.

En el primer nivel se busca las interferencias entre las dos mallas de voxels y como resultado se genera la lista de parejas de voxels intersectándose entre sí (lo que antes se llamaba v_o). Después, en un segundo paso, se va mirando cada pareja de voxels cogiendo sus dos listas de polígonos asociadas y comprobando la intersección entre parejas de triángulos.

Por tanto, el orden del algoritmo MVV queda de la siguiente forma:

$$O(t_{ejec}) = O(v_m) + O(v_o) \times O(f_{ve} f_{vm}) = O(v_m + v_o f_{ve} f_{vm}) \qquad (2.11)$$

Donde v_m es el número de voxels móviles y f_{ve} y f_{vm} son el número medio de facetas que pertenecen a cada pareja de voxels objetivo. En este caso v_o es diferente al parámetro usado en el análisis teórico del anterior apartado. En esa ocasión correspondía a todo el conjunto de voxels cubiertos por el volumen contenedor del objeto móvil, mientras que en la fórmula (2.11) representa al número medio de voxels cubierto por cada voxel móvil.

En la Figura 2.7 se puede observar los tiempos conseguidos con este método para diferentes voxelizaciones del objeto móvil.

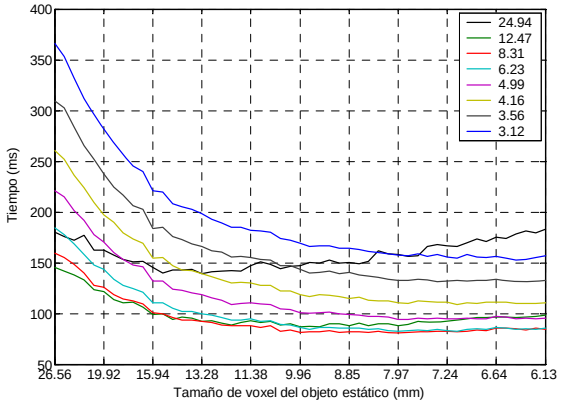


Figura 2.7: Relación de tiempos con el método MVV entre distintas voxelizaciones del objeto móvil en un rango de discretización estática.

Se puede apreciar que las curvas mínimas aparecen para el rango de tamaños del voxel móvil entre 6.23 y 12.47 mm. Para tener una perspectiva más amplia, en la Figura 2.8 se representa el comportamiento global del método mediante una superficie de tiempos.

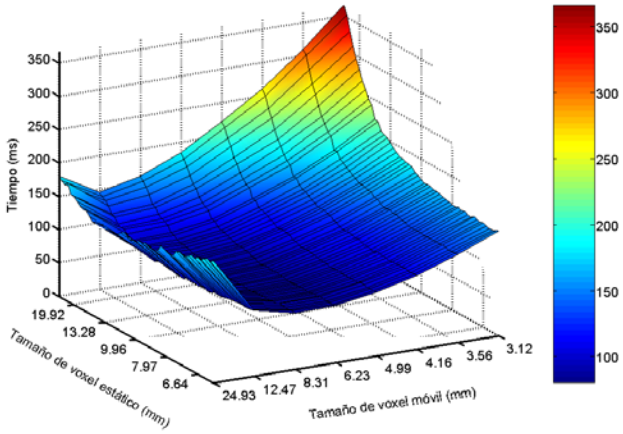


Figura 2.8: Tiempos usando diferentes niveles de discretización tanto del objeto estático como del móvil con el método MVV.

Para poder observar mejor dónde se encuentra el óptimo, en la Figura 2.9a se ha representado los isotiempos de la superficie resultante. Se ha modificado el mapa de colores para localizar más fácilmente la zona óptima (colores fríos para tiempos más pequeños). Según aparece, los cambios más bruscos se suceden en los niveles del objeto móvil. Para poder centrarse en ese detalle, en la Figura 2.9b se representan las medias de tiempos de todo un intervalo estático de discretización.

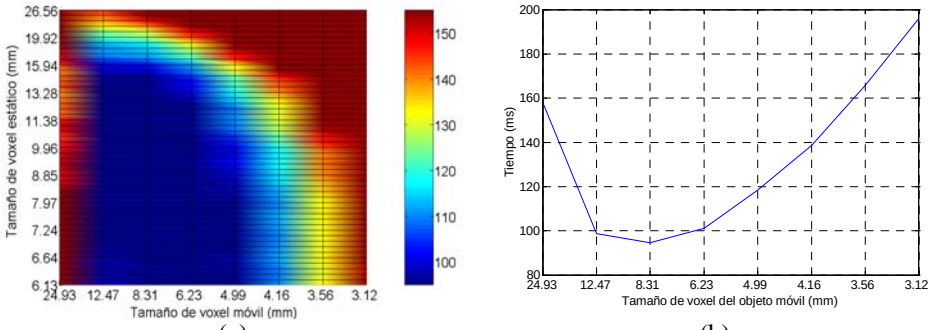


Figura 2.9: Tiempos con el método MVV. En (a) se representa el mapa de isotiempos comparando niveles de voxelización estática y móvil. En (b) se presentan las medias de tiempos para diferentes voxelizaciones del objeto móvil.

Tal y como se esperaba, existe un dominio de niveles en los que se alcanza un óptimo o rangos mínimos en los tiempos de análisis de colisiones. Más allá de ese dominio, la memoria consumida es mayor y además el rendimiento empieza a caer otra vez. Fijando la atención en esta última gráfica

se puede deducir qué nivel es el óptimo (en este caso coincide con el tamaño de voxel móvil 8.31 mm). Al final del capítulo se compararán los 4 métodos considerando el tiempo mínimo en cada caso.

En cuanto a la memoria consumida por este método se mostrará también al final del capítulo al compararlo con el resto de métodos propuestos.

2.4.3 Método puro de octrees (MOO)

Una vez se elige el número de niveles de cada octree, tanto para el estático como para el móvil, existen dos formas de realizar el preproceso de generación de los árboles.

En la primera de ellas se iría subdividiendo recursivamente el volumen contenedor del objeto. En cada subdivisión se mira si existen triángulos dentro de cada celda en cuyo caso afirmativo se seguiría subdividiendo hasta que se alcanzase la precisión elegida. Al mismo tiempo se va construyendo una matriz de punteros en la que en los nodos terminales se guarda la lista de punteros a los triángulos correspondientes a esa celda, y en los nodos parciales se guarda la dirección de los octrees hijos.

Esta forma de realizar la construcción del octree aunque es válida, resulta extremadamente lenta para el caso de objetos con un gran número de polígonos. Esto es así porque es necesario recorrer la lista de triángulos del objeto para cada octante de todas y cada una de las subdivisiones que se realizan. Por ello se ha desarrollado otra forma de efectuar el preproceso mucho más rápido aprovechando el modelo de voxels.

En este caso, se partiría de un mallado de voxels como si se fuera a utilizar el método MVV descrito en el apartado anterior. La diferencia viene a continuación: se empezaría a construir el octree de la misma forma que antes, con subdivisiones recursivas. En cada subdivisión, se tomaría el rango de voxels contenido en cada octante. Si todos los voxels están vacíos, ese octante ya no continua subdividiéndose. Si por el contrario alguno de esos voxels no está vacío, entonces se sigue subdividiendo ese octante.

La subdivisión continua hasta que los octantes alcanzan la precisión de los voxels, momento en el cuál los octantes copiarían las direcciones apuntadas por los voxels y se liberaría la memoria usada para el mallado de voxels.

Para determinar si existe colisión, se comparan los dos árboles entre sí empezando por los nodos raíces. Si existe colisión entre los dos octantes, se pasaría a mirar recursivamente los nodos hijos de esos dos octantes hasta que la pareja de octantes ya no tenga más subdivisiones, momento en el cual se

insertará en la lista de parejas de octantes intersectantes (similar a la lista de parejas de voxels del método anterior).

Si se analiza el algoritmo, se puede comprobar que se necesita pasar por todo el árbol del objeto móvil mientras que sobre el octree del objeto estático lo que se está haciendo son búsquedas. El orden del algoritmo se refleja a continuación:

$$O(t_{ejec}) = O(p_e o_m) + O(o_o) \times O(f_{he} f_{hm}) = O(p_e o_m + o_o f_{he} f_{hm}) \qquad (2.12)$$

Donde p_e es la profundidad del octree del objeto estático, o_o es el número de octantes objetivo (similar a v_o), o_m es el número de nodos del árbol móvil y los demás factores son los mismos que los descritos en el apartado anterior.

La penalización introducida por este algoritmo queda reflejada en el factor p_e que ahora multiplica al recorrido de la estructura del objeto móvil.

Como en el método anterior (MVV), en la Figura 2.10 se muestran diferentes tiempos dentro de un rango de niveles tanto del objeto estático como del móvil. Los tamaños de los niveles corresponden con el tamaño que alcanzarían los octantes hoja. En este caso, este parámetro es exponencial por lo que no se puede conseguir un conjunto de discretizaciones tan numeroso como ocurría en el método de voxels

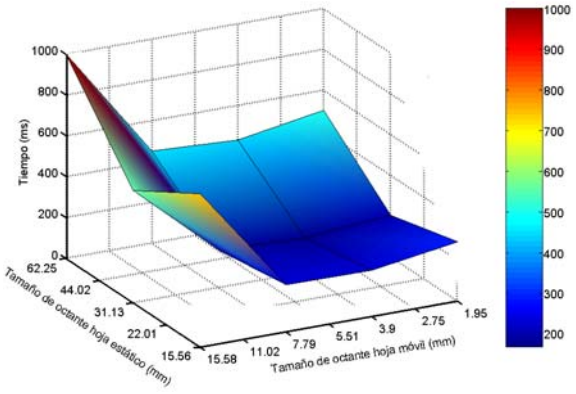
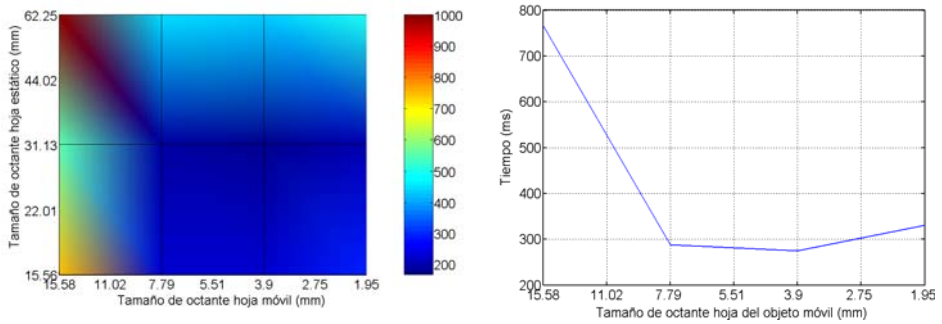


Figura 2.10: Relación de tiempos con el método MOO en un rango de subdivisiones del octree estático entre distintas subdivisiones del octree móvil.

Al igual que en el método anterior, en la Figura 2.11 se representan tanto el mapa de isotiempos como la media de tiempos para cada octree móvil.

Se comprueba que los tiempos para el método basado en octrees (MOO) son mayores que para el basado en voxels (MVV). Aunque con el estudio

teórico ya se intuía el resultado, al final del capítulo se muestra experimentalmente cuán peor es este método en comparación con los otros tres.



(a) (b)

Figura 2.11: Tiempos con el método MOO. En (a) se representa el mapa de isotiempos comparando niveles de subdivisión estática y móvil. En (b) se presentan las medias de tiempos para diferentes subdivisiones del octree móvil.

2.4.4 Método híbrido de voxels-octree (MVO)

Los dos métodos que se describen a continuación tratan de combinar las dos técnicas anteriores. En el método MVO se utiliza una estructura de voxels para toda la maqueta estática y una estructura octree para particionar la herramienta móvil.

La forma de generar ambas estructuras ya ha sido explicada en los dos métodos anteriores por lo que sólo hace falta analizar el orden del algoritmo. En este caso se parte del nodo raíz del octree móvil. Mientras exista colisión entre un octante y el entorno se va descendiendo en la jerarquía recursivamente. Como ocurría en el método anterior, al final es necesario recorrerse todo el árbol para llegar a todos los nodos hoja. Una vez se tienen los nodos terminales, se calcula con qué voxels del objeto estático están colisionando y se genera la lista de parejas voxel-hoja. La expresión matemática se muestra a continuación:

$$O(t_{ejec}) = O(o_m \times 1) + O(v_o) \times O(f_{ve} f_{hm}) = O(o_m + v_o f_{ve} f_{hm}) \qquad (2.13)$$

Como en esta ocasión se utiliza un mallado de voxels para el objeto estático, la búsqueda en él es directa como queda reflejado en la fórmula. Ésta queda idéntica a la (2.11) salvo en el factor que indica el recorrido por el objeto móvil. En un caso depende del número de voxels mientras que en este caso depende del número de nodos del árbol generado.

Pasando a los resultados experimentales, en la Figura 2.12 se muestra la superficie de tiempos, mientras que en la Figura 2.13a se representa el mapa de

isotiempos enfrentando diferentes niveles de voxelización estática y diferentes niveles de subdivisión del octree móvil. Además, en la Figura 2.13b aparece la media de tiempos para cada octree del objeto móvil.

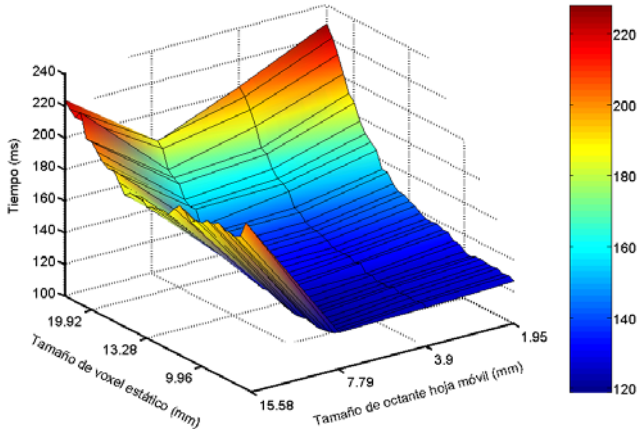
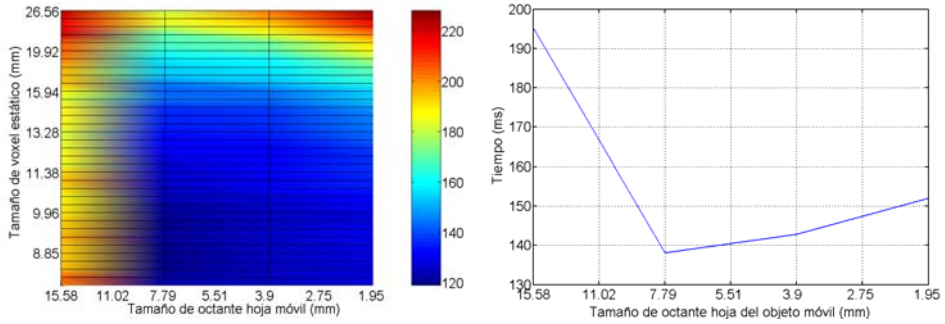


Figura 2.12: Relación de tiempos con el método MVO en un rango de voxelización del objeto estático entre distintas subdivisiones del octree móvil.



(a) (b)

Figura 2.13: Tiempos con el método MVO. En (a) se representa el mapa de isotiempos comparando niveles de voxelización estática y subdivisión de octree móvil. En (b) se presentan las medias de tiempos para diferentes subdivisiones del octree móvil.

Este método se comporta mejor que el MOO por el uso de la enumeración espacial en el objeto estático en lugar de la subdivisión jerárquica. Aunque en rendimiento sigue sin superar al MVV, los tiempos no se disparan tanto como en el método puro de octrees. Al final del capítulo se verá la comparación realizada y las ventajas que puede tener la elección de este método.

2.4.5 Método híbrido de octree-voxels (MOV)

Este caso es el simétrico del anterior. Para el objeto estático se utiliza una estructura octree mientras que para el objeto móvil se crea un mallado de voxels.

Como ocurría en el método MVV, primero hay que realizar un recorrido por el mallado de voxels móviles calculando su intersección con la estructura del objeto estático. La diferencia es que en lugar de ser una búsqueda directa, por cada voxel móvil hay que realizar una búsqueda por el árbol hasta llegar al nodo terminal objetivo. Por consiguiente la fórmula queda como se muestra a continuación:

$$O(t_{ejec}) = O(p_e v_m) + O(o_o) \times O(f_{he} f_{vm}) = O(p_e v_m + o_o f_{he} f_{vm}) \qquad (2.14)$$

Como en el resto de métodos, en la Figura 2.14 se muestra la superficie de tiempos enfrentando esta vez subdivisiones del octree estático contra niveles de voxelización del objeto móvil.

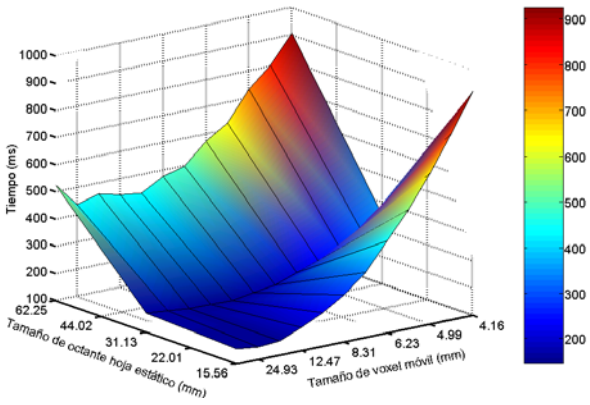
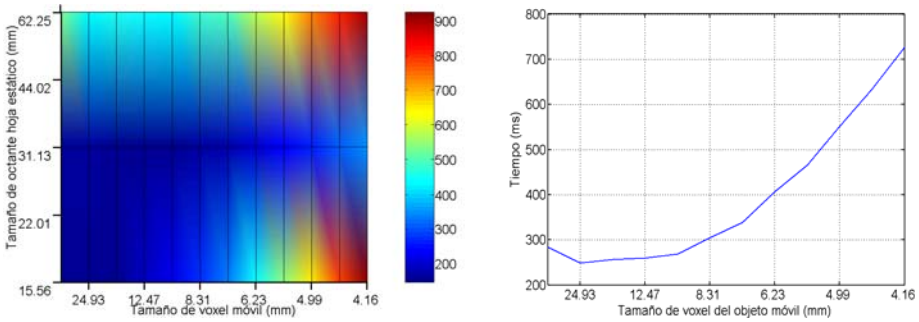


Figura 2.14: Relación de tiempos con el método MOV en un rango de subdivisiones del octree estático entre distintas voxelizaciones del octree móvil.

Como en los métodos anteriores, en la Figura 2.15 se representa el mapa de isotiempos y la media de tiempos, esta vez dependiente del tamaño de voxel en el objeto móvil.



(a) (b)

Figura 2.15: Tiempos con el método MOV. En (a) se representa el mapa de isotiempos comparando niveles de voxelización estática y subdivisión de octree móvil. En (b) se presentan las medias de tiempos para diferentes subdivisiones del octree móvil.

En este método vuelven a aparecer tiempos elevados debidos al uso otra vez de una estructura jerárquica octree en el objeto estático.

2.4.6 Comparación global

A continuación se muestra una serie de gráficas en las que se puede ver la comparación realizada entre los 4 métodos centrando la atención en los tiempos de cálculo y en la memoria consumida.

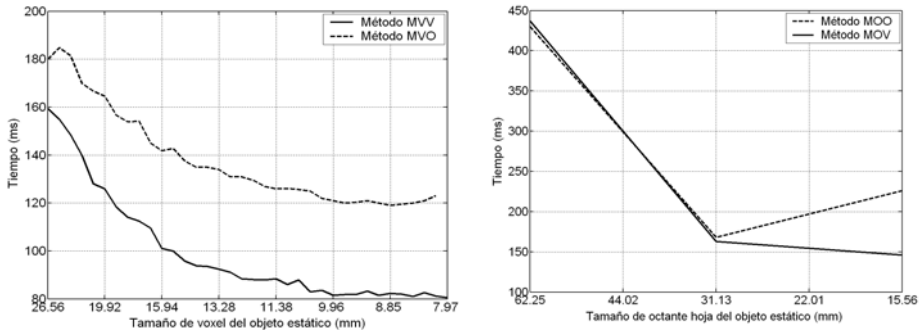
En la Tabla 2.1 se compara el rendimiento de los cuatro métodos. Al tener diferentes tamaños los voxels y los octantes, es difícil expresarlo en una única gráfica por lo se ha preferido poner los datos en una tabla. En cada caso, los tiempos corresponden al óptimo de cada método, es decir, se ha elegido el mínimo de cada una de las superficies de tiempos mostradas en los anteriores apartados.

Método	Tamaño del voxel/octante estático	Tamaño del voxel/octante móvil	Tiempo (ms)
MVV	7.97	8.31	80.6
MOO	31.13	3.9	168
MVO	8.85	7.79	119
MOV	15.56	24.93	146

Tabla 2.1: Tiempos mínimos de cada uno de los métodos. En cada método se muestra los tamaños (mm) de voxel o de octante que hacen mínimo el tiempo mostrado.

Según lo mostrado, el método más rápido sería el MVV y el que peor se comporta correspondería al método que posee sendos octrees en ambos objetos. Para tener una idea de cómo se comporta el método a lo largo de un intervalo de

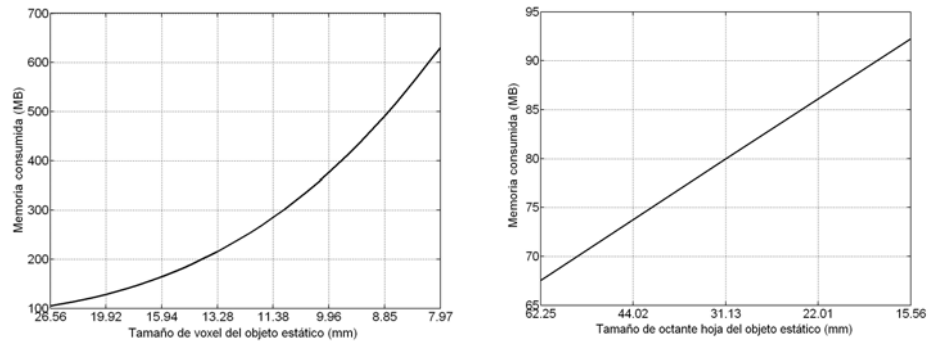
discretización estático, sí que se puede poner la comparativa en dos gráficas por separado, una comparando los dos métodos que presentan subdivisión uniforme en el estático (Figura 2.16a) y la otra con los otros dos métodos de octrees (Figura 2.16b). Para estas gráficas se han escogido los niveles de subdivisión óptimos del objeto móvil vistos en la tabla anterior.



(a) (b)

Figura 2.16: Comparativa de tiempos entre los diferentes métodos en un intervalo de discretización estático. En (a) se comparan los métodos MVV y MVO y en (b) se comparan MOO y MOV.

Además del rendimiento es necesario también comparar la memoria consumida por cada uno de ellos. Al igual que antes, en la Figura 2.17a aparece la curva de la memoria que consumirían los métodos que poseen enumeración espacial uniforme en el objeto estático (MVV, MOO), que lógicamente es coincidente por ser despreciable la memoria consumida por el objeto móvil. En la Figura 2.17b se representa la memoria consumida por los métodos que optan por una subdivisión jerárquica (MOO, MOV).



(a) (b)

Figura 2.17: Comparación de la memoria consumida entre diferentes métodos. En (a) se compara el MVV y el MVO y en (b) se compara el MOO y el MOV.

Se pueden extraer varias conclusiones del análisis y comparativa realizados en este capítulo:

- El método de voxels MVV es el que da mejores resultados en cuanto a su rendimiento de tiempos.

- El peor resultado corresponde con el método puro de octrees, MOO, debido a los recorridos que deben hacerse por sus estructuras arbóreas.

- Sin embargo, en cuanto a la memoria consumida sucede al revés. El MOO consume aproximadamente 8 veces menos que el método de voxels puro MVV. Aún cuando el consumo de memoria de este método sea mucho menor que el consumido por el método MVV, este ahorro de memoria no compensa la pérdida en el rendimiento.

- El método MVO carece de interés ya que el rendimiento conseguido es parecido al MVV (ligeramente menor) pero la memoria consumida sigue siendo la misma ya que el objeto estático también es voxelizado con este método.

- Una solución de compromiso podría ser el método MOV. El rendimiento es menor que el MVV pero no llega a caer tanto como el método puro de octrees. Además, este método tiene la ventaja del ahorro inherente en la estructura octree al utilizarse ésta para particionar el objeto estático.

En este capítulo se han presentado e implementado varios métodos de subdivisión espacial enumerativos y jerárquicos. El objetivo ha sido justificar la elección de uno de ellos como base para el método que se propone en el siguiente capítulo.

Vistas las conclusiones, se podría pensar en la elección del método MVV o MOV como buenos candidatos para la detección de colisiones en el tipo de aplicaciones que se quieren manejar. Sin embargo, según las gráficas de tiempos de uno y otro, los tiempos están muy cerca del límite impuesto para una buena interactividad de la aplicación por lo que se hace necesario un nuevo planteamiento del problema y modificar los algoritmos vistos en este capítulo. Como se ha podido comprobar, el único método que ha dado algún tiempo por debajo del límite de interactividad ha sido el MVV por lo que en el capítulo siguiente se partirá de este método para la propuesta final. El método MOV, aunque bueno en memoria se rechaza por dos razones: primero porque los tiempos son superiores a los 100 ms y segundo, como se verá en el capítulo

siguiente, el método óptimo nos ha llevado incluso a prescindir de la voxelización del objeto móvil.

2.5 REFERENCIAS

Barriuso, J.R., "Cálculo de la Distancia entre Objetos representados con precisión mediante Poliedros Cóncavos durante una Simulación Mecánica", *Tesis Doctoral*. Universidad de Navarra. Octubre 1998.

Foley, J.D., Dam, A. Van, Feiner, S.K., y Hughes, J.F., "*Computer Graphics principles and practice*", Second Edition, Addison-Wesley, 1990.

García-Alonso, A., Serrano, N. y Flaquer, J., "Solving the Collision Detection Problem", *IEEE Computer Graphics and Applications*, Vol. 13 (3), pp. 36-43. Mayo 1994.

Hayward, V., "Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Workspace", *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 2, San Francisco, CA. 1986.

Hubbard, P.M., "Interactive collision detection", *In Proceedings of the 1993 IEEE Symposium on Research Frontiers in Virtual Reality*, pp. 24-31. Octubre 1993.

Lawlor, O.S. y Kalé, L.V., "A Voxel-Based Parallel Collision Detection Algorithm", *Proceedings of the ICS'02 (International Conference on Supercomputing)*, New York, New York, USA. Junio 22-26, 2002.

McNeely, W.A., Puterbaugh, K.D. y Troy, J.J., "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", *Proceedings of the ACM SIGGRAPH'99 - Computer Graphics*, pp. 401-408. Los Ángeles, California, USA. Agosto 1999.

Möller, T., "A Fast Triangle-Triangle Intersection Test", *Journal of Graphics Tools (JGT)*, Vol. 2, No. 2, pp. 25-30. 1997.

Shaffer, C.A. y Herb, G.M., "A Real-Time Robot Arm Collision Avoidance System", *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 2, pp. 149-160. Abril 1992.

CAPÍTULO 3

EVOLUCIÓN Y DESCRIPCIÓN DEL MÉTODO PROPUESTO

3.1 INTRODUCCIÓN

En este capítulo se describen y justifican los algoritmos utilizados para resolver el problema de la detección de colisiones entre objetos representados mediante facetas planas tras analizar diversas mejoras en el algoritmo MVV. Este nuevo método cumple con las condiciones y objetivos propuestos en el capítulo introductorio de la Tesis. El algoritmo final está basado en las discusiones planteadas en el capítulo anterior donde se describía y planteaba el problema a resolver.

Antes de llegar a la solución finalmente adoptada, se ha partido de soluciones ya implementadas y probadas en el capítulo anterior. En este capítulo se describirá el método finalmente adoptado y se mostrarán resultados experimentales que prueban el mejor rendimiento del método presentado en esta Tesis.

Inicialmente, se introduce el esquema general utilizado por el método. En apartados posteriores se procede a la descripción detallada de cada uno de los pasos enunciados en el esquema. En estos apartados se analiza y compara el método propuesto con el algoritmo MVV del capítulo anterior y con otros existentes en la bibliografía. Este método lo denominaremos MVF y la razón de este nombre aparecerá más adelante (apartado 3.2.1).

3.2 ESQUEMA GENERAL DEL MÉTODO

En la Figura 2.2 se mostraba un esquema muy general de la resolución del problema utilizando métodos de partición espacial. En este apartado se expande y se adapta ese esquema para mostrar todos los pasos seguidos por el método propuesto. Esta adaptación queda reflejada en la Figura 3.1 que se muestra a continuación.

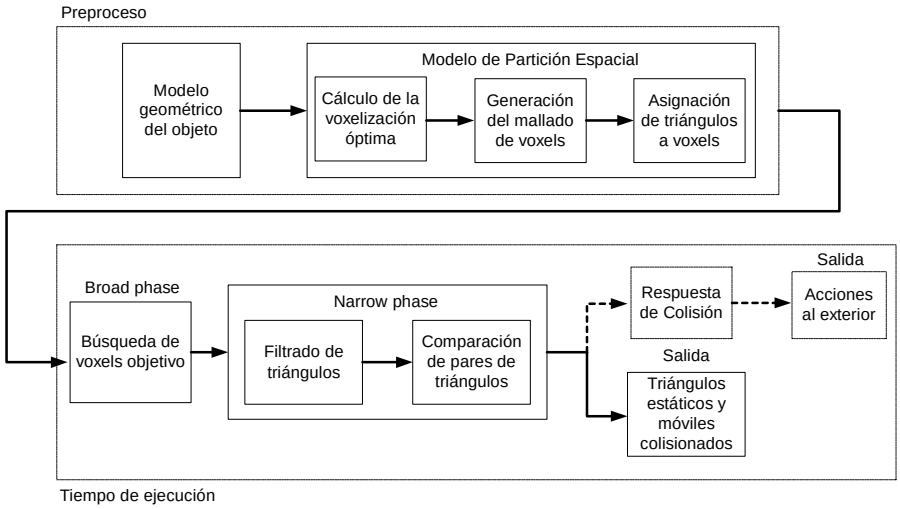


Figura 3.1: Esquema general del método propuesto.

El esquema está dividido en dos grandes bloques. El primer bloque, que se ejecuta una única vez y en tiempo de preproceso, es el encargado de generar el modelo de partición espacial; y el segundo, que se ejecuta en cada periodo o ciclo de ejecución del proceso de colisiones, se encarga de calcular las intersecciones entre los objetos de la escena. El segundo bloque se ejecuta en un único procesador dedicado exclusivamente a este cálculo por lo que la visualización de la escena gráfica no afecta en absoluto. Esta arquitectura será descrita en detalle en el capítulo 6.

Además de los dos grandes bloques, en el esquema aparece otra zona dibujada con línea discontinua. Esta zona pertenece al cálculo de la respuesta de colisión. Este problema se analizará en el capítulo 5.

En los dos siguientes apartados se explica y analiza cada uno de los dos grandes bloques mostrados en el esquema del método. Además, en cada uno de ellos se comparan diferentes alternativas y se realiza un análisis más profundo de determinados aspectos.

3.2.1 Bloque del Preproceso

Este bloque genera una estructura de datos para que el bloque de tiempo de ejecución sea capaz de calcular todas las colisiones de la escena (y otras informaciones como la respuesta de colisión) en tiempo real. Como entrada únicamente tiene la representación geométrica de los objetos que en el caso de la presente Tesis corresponde con la representación B-Rep generada por sistemas CAD.

Al contrario que los algoritmos vistos hasta ahora, este método sólo particiona espacialmente la maqueta estática, el objeto móvil no se particiona y sólo se usará su descripción poligonal. Por eso, y siguiendo con la terminología utilizada en el capítulo anterior, al método propuesto se le llama MVF, método de voxels-facetas.

Este bloque se subdivide en tres fases:

- Cálculo de la voxelización óptima: se trata de calcular el nivel de discretización más adecuado en cada dimensión para que el cálculo de las colisiones en tiempo de ejecución sea el mínimo posible. Este es un problema complejo cuyo análisis y resolución queda reflejado en el capítulo siguiente.

- Generación del mallado de voxels: una vez calculado el voxelizado óptimo, esta fase sólo tiene que reservar la memoria necesaria para albergar el mallado de voxels (de tamaño igual al calculado en la anterior fase) y otras estructuras necesarias.

- Asignación de triángulos a voxels: en las dos fases previas se calcula y genera la partición espacial vacía. Esta fase es la encargada de asignar a cada voxel los triángulos contenidos en él.

Como se ha mencionado, la primera fase será fruto del análisis realizado en un capítulo posterior, por lo que a continuación se explican las dos fases siguientes. Previamente se realizaran unas consideraciones de tipo práctico sobre la memoria.

3.2.1.1 Consideraciones sobre la memoria RAM

Antes de abordar los algoritmos conviene hacer una consideración acerca de la memoria disponible en el sistema. Estas consideraciones son muy importantes desde el punto de vista práctico: las aplicaciones actuales de los resultados de esta Tesis. Sin embargo, no condicionan a los algoritmos, resultados y conclusiones obtenidos.

En el capítulo 6 se describe minuciosamente la arquitectura del sistema, pero antes de estudiar cuánta memoria será necesaria para el módulo de Colisiones, se deben considerar los recursos que se poseen (en este caso de memoria física). Por ejemplo, el Sistema Operativo (SO) es un factor a tener en cuenta. Todo el sistema está desarrollado en la plataforma de Microsoft Windows 2000. Este SO limita la memoria utilizada por un proceso a 2 GB. Aunque los dos factores (tiempo-memoria) son importantes, desde el principio se ha apostado por un mayor rendimiento aunque signifique mayor consumo de memoria, por lo tanto se ha dispuesto de un PC con esos 2 GB de memoria física a compartir entre el módulo de Visualización, que deberá guardar toda la información geométrica de la escena, y el módulo de Colisiones.

Si se poseen los 2 GB de memoria física, como es el caso, la teoría dice que el SO debería permitir a un proceso el uso total de esos 2 GB. Pero en la práctica no es así. El SO siempre se reserva un porcentaje de la memoria física para uso personal. La cantidad reservada dependerá del SO y del número de aplicaciones residentes en memoria desde el arranque del computador. En el presente caso el SO se reserva 200 MB para sí.

Esto significa que los módulos de visualización y de colisiones tienen que compartir los 1.8 GB restantes. A priori no se conoce cuánta memoria necesita uno u otro módulo pero una primera aproximación puede ser dejar 1 GB para la detección de colisiones y 800 MB para el módulo visual, más que suficiente para almacenar la información geométrica de la maqueta virtual. Aunque no es objetivo de esta Tesis, se puede hacer un rápido análisis de la memoria necesitada por el módulo visual ya que puede afectar a la división de memoria realizada para ambos módulos.

El sistema visual se organiza jerárquicamente en una estructura arbórea que almacena un nodo raíz, que contiene a toda la escena; nodos hojas, los cuales contienen la geometría; y nodos intermedios que contienen información relacionada con objetos enteros como pueden ser sus volúmenes contenedores (cajas, esferas, etc), matrices de transformación, color, etc. Sin embargo se puede simplificar esa estructura teniendo en cuenta únicamente los nodos terminales que son los que contendrán el mayor peso en el almacenamiento de los datos.

Básicamente, para cada vértice, el módulo visual utiliza los siguientes campos: 6 floats, coordenadas globales y locales, para cada vértice y otros 6 para las normales asociadas al vértice. Para cada triángulo se utilizan 3 índices para los 3 vértices, y otros 3 índices para las normales. Si cada tipo de datos ocupa 4 bytes (floats e índices), y casi siempre se mantiene que el número de vértices es aproximadamente el doble que el de triángulos; en total se llega a consumir una cantidad de memoria aproximada a:

$$48n_v + 24n_t = 96n_t + 24n_t = 120n_t \; bytes \tag{3.1}$$

Siendo n_v el número de vértices y n_t el número de triángulos de la escena. Eso significa que para modelos de turbinas aeronáuticas de 2 millones de triángulos, el módulo visual requeriría aproximadamente 240 MB.

En el caso peor, para gastar los 800 MB de memoria, el módulo visual estaría guardando información de un modelo de aproximadamente 6.5 millones de triángulos, cifra que parece lejana a los modelos con los que se está trabajando actualmente en el campo aeronáutico.

Limitado a 1 GB, el módulo de colisiones necesita gestionar ese espacio para almacenar diferentes estructuras de datos que se describen en este capítulo. Una vez vista la memoria disponible se puede pasar a describir la fase de la generación del mallado de voxels.

3.2.1.2 Generación del mallado de voxels

En la fase de generación del mallado de voxels, se generan varias estructuras de datos útiles para el cálculo de las colisiones: una es la malla 3D de voxels con un tamaño preestablecido. Este tamaño se calcula previamente en la fase del voxelizado óptimo de la escena. A esta estructura se le llamará *estructura Voxel*.

A partir de la representación poligonal de la superficie frontera del objeto, esta fase divide el volumen del AABB inicial en volúmenes menores, iguales y disjuntos entre sí, de modo que la suma de todos ellos resulte ser el volumen inicial.

Se han desarrollado dos métodos cuya diferencia radica en la forma geométrica de las celdas o voxels generados.

- Voxels paralelepípedos: una vez elegido un nivel de voxelizado, se aplica éste a las tres dimensiones del objeto, es decir, cada dimensión se divide por el mismo número *N*. El resultado es una malla de paralelepípedos con las dimensiones proporcionales a las del volumen inicial.

- Voxels cúbicos: el nivel de voxelizado elegido en este caso es un grano de celda δ. Sólo resta aplicar ese tamaño de celda a las tres dimensiones para obtener unos voxels homogéneos.

La comparación en el rendimiento entre los dos métodos se mostrará al final del capítulo. Conviene recordar que esta etapa sólo se ocupa de realizar

reservas de bloques de memoria, y la siguiente los rellena con apuntadores a información geométrica. Se verán dos métodos: el clásico y el basado en *hashing*.

3.2.1.2.1 Método clásico

Para reservar la memoria necesaria la solución más eficiente sería usar un array 3D de dimensiones N_i, N_j y N_k, pero sólo es válido para soluciones en las que se conoce el tamaño antes de compilar el programa. Para soluciones dinámicas, el array 3D se aloca mediante punteros pero tiene dos inconvenientes. Por un lado implica tener 3 niveles de indirección y por otro se desperdicia memoria con los punteros. Más concretamente, si se tiene un mallado de por ejemplo 100x100x100, la memoria consumida superará los 100^3 x 4 bytes, pues hay que tener en cuenta que las dos primeras dimensiones actúan como apuntadores. La expresión matemática para la memoria consumida queda reflejada por (3.2).

$$Memoria = 4 \times \sum_{i=1}^{3} N^i \tag{3.2}$$

Para evitar esto se suele alocar un bloque único como array unidimensional de tamaño N_ixN_jxN_k y se gestionan internamente las tres dimensiones con fórmulas inmediatas como las expresadas por (3.3).

$$clave = x \times N_j \times N_k + y \times N_k + z$$
$$x = \left\lfloor \frac{clave}{N_j \times N_k} \right\rfloor, \quad y = \left\lfloor \frac{clave\%\left(N_j \times N_k\right)}{N_k} \right\rfloor, \quad z = \left(clave\%\left(N_j \times N_k\right)\right)\%N_k \tag{3.3}$$

Donde *clave* hace referencia a la posición en el array unidimensional. Con soluciones sencillas y de poca memoria esto no presenta ningún problema, sin embargo, cuando se quiere reservar grandes cantidades de memoria se puede dar un problema de fragmentación. Para hacerse una idea, es frecuente que la memoria requerida por la estructura de voxels alcance los cientos de Mbytes. Sin embargo, el problema se muestra con un ejemplo más sencillo en la Figura 3.2. El bloque de tamaño 10 cabe perfectamente en la memoria física del ordenador pero tal y como está fragmentada la memoria, el SO no puede reservar el bloque en posiciones consecutivas de memoria.

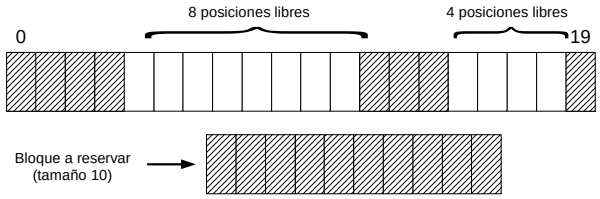


Figura 3.2: Problema de fragmentación interna de la memoria.

La solución ofrecida es la de realizar intentos dicotómicos de reserva de memoria. Se encapsula de forma transparente esa gestión de memoria de tal forma que si la reserva diese error, automáticamente se intenta reservar dos bloques de tamaño la mitad que el que ha dado error, y así sucesivamente.

Como se ha dicho anteriormente, la estructura resultante se llamará *estructura Voxel*. En la Figura 3.3 aparece esta estructura de datos junto con las demás usadas para voxelizar al objeto estático. Como el problema de direccionamiento de voxels queda encapsulado y con un consumo mínimo de memoria, a partir de ahora se considerará la *estructura Voxel* como un único array continuo.

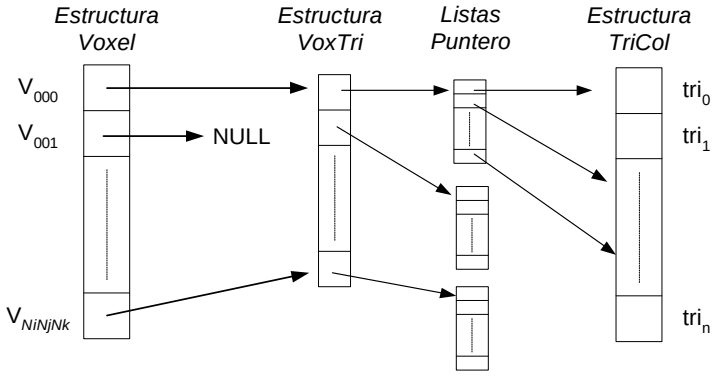


Figura 3.3: Representación en memoria de las estructuras de datos utilizadas en el método MVF.

Durante la etapa de preproceso, la ***estructura Voxel*** se construye como un array de punteros a objetos *VoxTri* (Voxel-Triángulo). Esta representación asegura la mínima información por voxel (4 bytes). Si es un voxel vacío el puntero será NULL y en caso contrario apuntará a un objeto *VoxTri*.

La ***estructura VoxTri*** se construye como un array de objetos *VoxTri*. De este modo la memoria consumida por los objetos *VoxTri* es menor que si se alocan independientemente. A pesar de crear este array, la *estructura Voxel* direcciona directamente cada *VoxTri*; es decir, no se usan índices a la estructura

VoxTri. Cada objeto de la *estructura VoxTri* tiene una lista de apuntadores a información de triángulos (*estructura TriCol*). El tamaño de la *estructura VoxTri* es proporcional al número de voxels no vacíos, factor n_p de la ecuación (2.5). El orden de n_p es el mismo que el del porcentaje de ocupación, que en este tipo de solución enumerativa es muy bajo (alrededor del 5%).

La ***estructura TriCol*** es un array de objetos *TriCol*. Cada objeto contiene información acerca de un triángulo de la escena. La información de las coordenadas de cada triángulo es compartida con el módulo visual y si sólo fuera necesario disponer de información geométrica, entonces no haría falta reservar esta estructura adicional. Pero como se verá, el cálculo de colisiones se acelera si se guarda información adicional por cada triángulo. En esa información se encuentran los siguientes campos:

- Coordenadas globales de los vértices del triángulo: en realidad no se almacenan las coordenadas sino que se almacenan los 3 punteros que apuntan a los vértices almacenados por el módulo visual.

- Coeficientes de la ecuación del plano: corresponde al plano que contiene al triángulo. En el caso de la maqueta estática, este dato se calcula una única vez en tiempo de preproceso.

- Identificador del triángulo.

- Apuntador al objeto del que forma parte el triángulo.

- Flag de colisionable.

- Número de *frame* e identificador de triángulo colisionando en ese frame.

La descripción de la escena estática la componen las estructuras de datos *Voxel*, *VoxTri* y *TriCol*. En cambio, la descripción del objeto móvil sólo contiene la *estructura TriCol*. El objeto móvil no usa las otras estructuras porque, como se verá, no son usadas por el algoritmo propuesto MVF. En el algoritmo MVV sí que eran necesarias. Su eliminación no reporta ahorros significativos de memoria.

En total, para cada triángulo de la escena, el módulo de colisiones almacena 46 bytes. Sin embargo, el mayor peso en el consumo de memoria corresponde a la *estructura Voxel* del objeto estático. En la Figura 3.4 se representa el consumo de memoria con esta solución, construyendo mallados de voxels de diferentes tamaños para un mismo modelo. El modelo de maqueta y

herramienta son los mismos que en el capítulo anterior (1615172 polígonos para el primero y 2687 para el segundo). Para este ejemplo se ha usado la solución paralelepípeda de voxels pero con la cúbica las conclusiones serían similares.

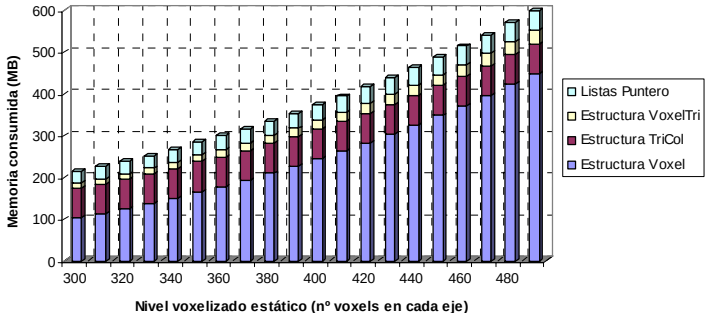


Figura 3.4: Memoria reservada para las estructuras *Voxel*, *VoxTri* y *TriCol* con diferentes niveles de voxelización.

Plantear estas estructuras de datos para resolver el problema de colisiones no es inmediato. Lo primero que se podría pensar es en tener una lista en cada voxel, es decir, una lista por cada elemento de la *estructura Voxel*. Esta alternativa ahorraría el uso de la estructura intermedia *VoxTri* pero el consumo de memoria sería mucho más elevado al tener más información en cada voxel (en nuestra implementación una lista necesita 12 bytes de almacenamiento: 4 bytes para el puntero a los elementos de la lista y otros 8 para información adicional como el tamaño de la lista y el número de elementos). La Figura 3.5 muestra la diferencia entre las dos soluciones.

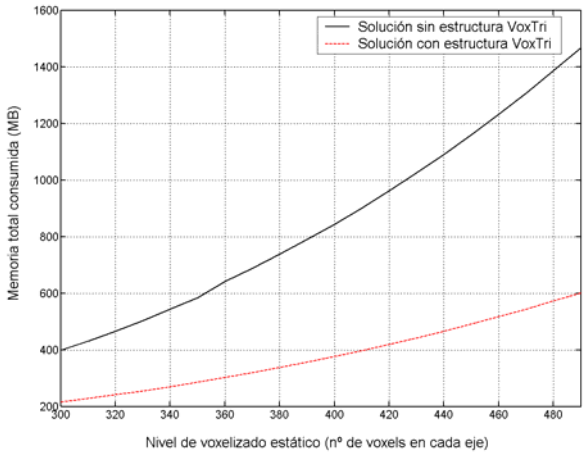


Figura 3.5: Comparación de la memoria total consumida utilizando la estructura de datos *VoxTri* y sin ella.

Gracias a la solución finalmente adoptada (utilizando la *estructura VoxTri*), se puede llegar a tener niveles de voxelizado de hasta 500x500x500 sin acercarse al límite de memoria establecido. Sin embargo, como se verá en el capítulo 4 donde se analiza la búsqueda del voxelizado óptimo, es necesario alcanzar niveles de voxelización superiores para poder analizar el rendimiento del algoritmo en función del grano de voxelización. Por ello, es necesario incorporar otra solución para ahorrar todavía más memoria. Esta solución se presenta en el siguiente apartado.

3.2.1.2.2 Técnicas de Hashing

Algunos autores (Gregory et al. 2000) para ahorrar memoria en métodos de enumeración espacial usan técnicas de *hashing* (de dispersión).

Este tipo de técnicas se basan en establecer una función *H*, llamada función *Hash*, que aplicándose a un valor de clave (posición en una tabla) produce un nuevo valor que se corresponde con una posición en el nuevo espacio de direccionamiento. Conceptualmente, el *hashing* transforma una tabla de almacenamiento en otra más pequeña. Las posiciones de la nueva tabla se consiguen con la función *H*.

Antes de continuar es mejor definir una serie de términos que serán utilizados en la descripción de la solución *Hashing*.

- *Estructura Voxel-virtual*: será la estructura que se ha usado en el método clásico (apartado anterior). Se llama virtual porque realmente ese espacio de direcciones no existe. A las posiciones de esta estructura (posiciones virtuales) se les aplica *H* para obtener las posiciones reales.

- Estructura *Voxel-real*: se llamará de este modo al mallado de voxels que sí se reserva en memoria. Por tanto las posiciones resultantes de aplicar *H* serán posiciones reales de esta estructura.

- El tamaño de esta nueva estructura será $numVoxels_r$.

Hay que tener en cuenta que la transformación *Hash* queda encapsulada en la gestión de la *estructura Voxel*. Es decir, los algoritmos de colisiones "direccionan" los voxels virtuales.

Este tipo de técnicas son usadas también en el ámbito de las Bases de Datos (BD) y direccionamiento de ficheros (Elmasri y Navathe 1997). En estas áreas se busca una buena función *Hash* que cumpla objetivos que también son aplicables en este contexto. Estas propiedades deberían ser:

- Las nuevas posiciones generadas deberían estar uniformemente distribuidas.

- La función *Hash* debería aplicarse a un campo clave que sea único para cada valor, es decir, que no haya dos posiciones con la misma clave.

- Se debe minimizar los *sinónimos*. En el presente contexto, dos sinónimos serían dos voxels virtuales y distintos, que contienen triángulos en su interior, y que al aplicarles la función *Hash*, el resultado es una misma posición en la *estructura Voxel-real*.

- La *estructura Voxel-real* debería tener la mayor parte de sus voxels ocupados con triángulos (alto porcentaje de ocupación).

En nuestro trabajo, el valor de la clave coincide con el índice o posición dentro de la *estructura Voxel*. Ahora hace falta crear una función *Hash* que convierta esa posición de la *estructura Voxel-virtual* a una posición dentro de la *estructura Voxel-real*. Una función *Hash* común es la reflejada por (3.4).

$$H(clave) = clave\%(numVoxels_r) \tag{3.4}$$

Algunos trabajos en el área de las bases de datos afirman que si el tamaño de la nueva tabla (en este caso la *estructura Voxel-real*) es un número primo, las nuevas posiciones generadas por *H* se distribuyen mejor dentro del espacio de direcciones cuando la función utilizada es el resto (Elmasri y Navathe 1997). Otros, en nuestra área, afirman que para evitar demasiados voxels con el mismo patrón, también puede ser útil aplicar una función *random()* a la clave antes de calcular el resto (Gregory et al. 2000). El introducir un factor aleatorio a la función puede tener sentido ya que deriva en un resultado más disperso, pero tiene un inconveniente. Si a cada clave se le aplica un factor de aleatorización, significa que cada posición virtual (voxel virtual) debería tener un campo más almacenando ese factor ya que en tiempo de ejecución hará falta volver a conseguir las mismas posiciones reales a partir de las posiciones virtuales. Esto traería consigo otra vez un incremento importante en la memoria que anularía el ahorro de memoria del *hashing*.

En las siguientes gráficas se muestra el impacto, a nivel de tiempos y de memoria consumida, que supone la utilización de la técnica *Hash*. En cada figura varía en un eje los diferentes tamaños de voxel y en el otro eje los distintos niveles de *hashing*. Este nivel de *hashing* hace referencia al porcentaje en que se ve reducida la *estructura Voxel-real* (un nivel de *hashing* de 50%

significa un ahorro de memoria en la *estructura Voxel* del 50%). En la (Figura 3.6) se representan los tiempos del análisis de colisiones en función de esas dos variables.

En lo que respecta al rendimiento, gracias a los experimentos presentados en esta Figura 3.6 y a otros similares, se pueden sacar una serie de importantes conclusiones. En primer lugar, hay un pequeño incremento en el coste computacional al pasar del nivel de *hashing* cero (no usar *hashing*) a los demás niveles. Esto es razonable ya que el uso de técnicas de *hashing* conlleva un cálculo adicional cada vez que se quiere acceder a la estructura.

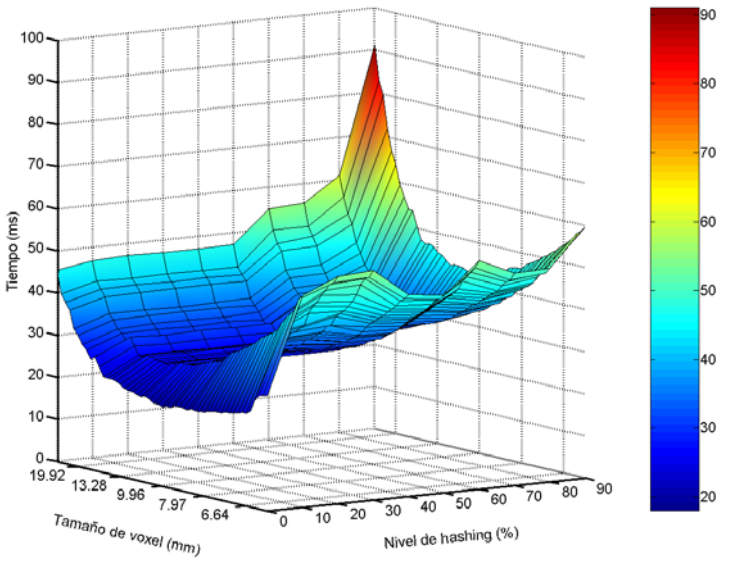


Figura 3.6: Rendimiento del método aplicando diferentes niveles de *hashing.*

También se nota en los niveles más elevados (60% - 90%) del *hashing* un escalón en el rendimiento, más significativo en el último nivel. Esto se debe al *fenómeno de los sinónimos*. Sin embargo, en el rango de niveles medios y bajos (10% - 50%), los tiempos se mantienen más o menos constantes, y aunque se aplique niveles más altos, el rendimiento no empeora de modo alarmante salvo quizás en el nivel del 90%.

Aunque a primera vista parezca un hecho sino erróneo si sorprendente, estos resultados se pueden explicar fijándose en el número de pares de triángulos que se analizan en cada caso (recuérdese que los voxels son un instrumento para determinar qué pares de triángulos estático-móvil deben evaluarse para ver si interfieren). En la Figura 3.7 se representa el número de pares de triángulos que en el ejemplo estudiado se han tenido que evaluar en la

narrow phase (colisión triángulo-triángulo). Este número es el original, es decir, antes de aplicar ningún tipo de filtrado como los que se describirán en el apartado 3.2.2.3.

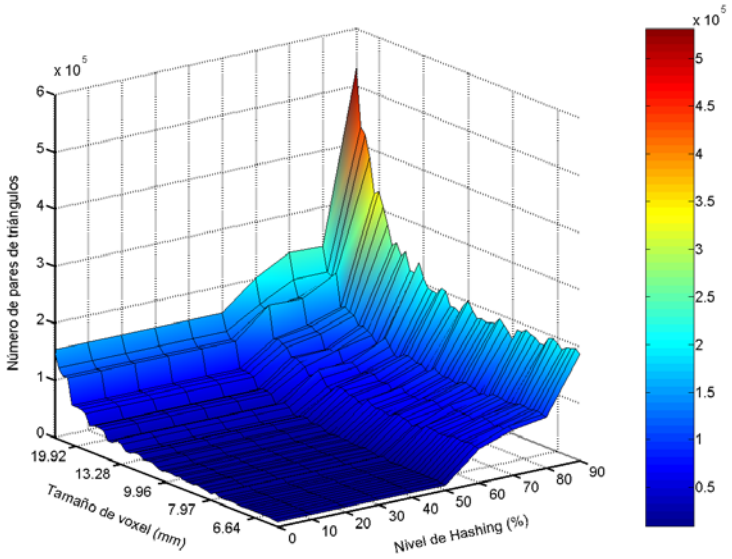


Figura 3.7: Número de pares de triángulos implicados en diferentes niveles de *hashing* y diferentes tamaños de voxel.

Como se puede apreciar, en el rango de niveles [0 – 50], el número de pares de triángulos implicados es el mismo. Esto quiere decir que reduciendo la *estructura Voxel* por debajo de un 50%, el número de triángulos que contiene cada voxel es el mismo que si no se aplica ninguna función *Hash*, o lo que es lo mismo, no ha aparecido ningún *sinónimo* en la estructura. La diferencia de tiempos entre el nivel 0 y los niveles [10-50] vista en la Figura 3.6 se debe únicamente al coste de la función *H* que se tiene que aplicar cada vez que se quiere acceder a la estructura.

En la Figura 3.6 también se puede apreciar que los tiempos vuelven a incrementarse cuando la malla de voxels se vuelve muy fina, sin embargo en la Figura 3.7 se observa que el número de triángulos sigue disminuyendo contra más pequeño sea el voxel. Este fenómeno se debe a que el rendimiento depende del número de triángulos pero también del número de voxels involucrados en la zona de colisión. Esta dependencia del tiempo respecto de los triángulos y los voxels es la base del cálculo de la voxelización óptima. Así en el apartado 3.2.2 se describe detalladamente cómo se forman estos conjuntos y en el capítulo 4 se analiza cuánto influye cada conjunto en el rendimiento total.

En la siguiente figura (Figura 3.8) se puede comprobar el porcentaje de ocupación de voxels que poseen los modelos típicos de motores aeronáuticos. Esta figura es un claro ejemplo de los problemas subyacentes a la elección de un modelo de enumeración espacial. El porcentaje de voxels ocupados es realmente bajo incluso al aplicar niveles de *hashing* de 80% ó 90%. Con el método clásico (nivel de *hashing* 0) el porcentaje no sobrepasa el 10%, o lo que es lo mismo, se está desperdiciando más del 90% de la memoria dedicada a los voxels.

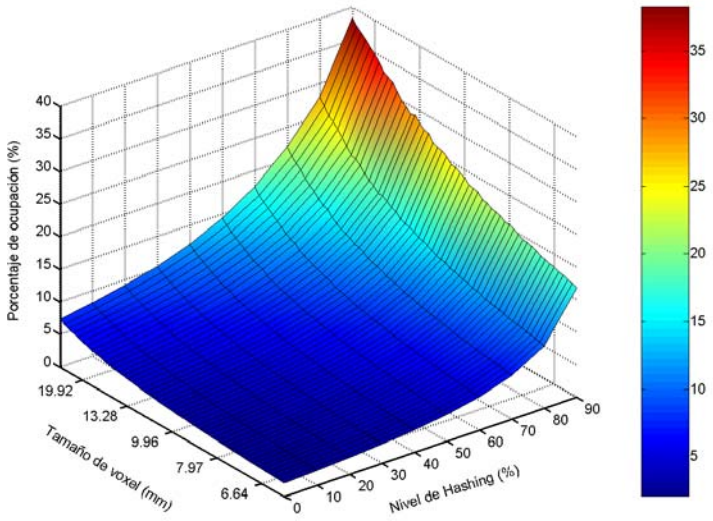


Figura 3.8: Porcentaje de ocupación (% de "voxels reales" que contienen geometría) aplicando diferentes niveles de *hashing.*

Por último, la Figura 3.9 muestra el consumo de memoria total en cada nivel de *hashing*. En lo que respecta a la memoria no hay lugar para la confusión. El nivel de *hashing* indica cuánto se va a reducir la *estructura* V*oxel*. Como esta estructura es la que mayor impacto tiene en el consumo total, un nivel de *hashing* de 50 % significará que el ahorro de memoria es aproximadamente la mitad comparando con el método clásico sin *hashing*.

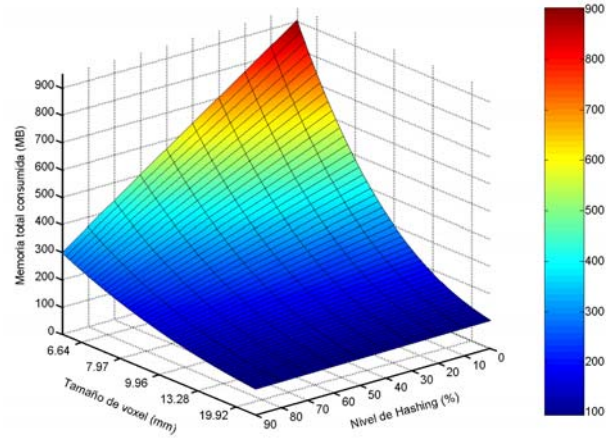


Figura 3.9: Memoria total consumida aplicando diferentes niveles de *hashing.*

Se ha mostrado la razón por la cual el *hashing* prácticamente tiene un coste nulo, es decir, no disminuye el rendimiento del análisis de colisiones. Esta razón es la ausencia de *sinónimos*. Sin embargo, todavía queda por justificar o explicar por qué ocurre este fenómeno en el *hashing* y no ocurre lo mismo al variar el tamaño del voxel. Realmente, con el *hashing* también se varía de cierta manera el volumen del voxel real pero esta variación no es uniforme en el espacio. Una descripción gráfica de lo que ocurre en la generación del voxelizado con y sin *hashing* se puede observar en las siguientes figuras. En la Figura 3.10 se representan diferentes tamaños de voxel (δ) para representar voxelizados más o menos finos.

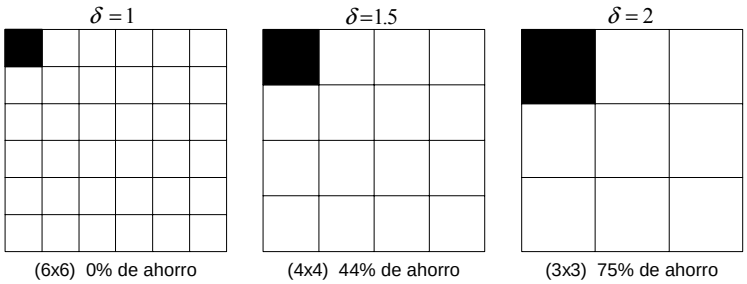


Figura 3.10: Diferentes tamaños de voxel para voxelizar un mismo espacio 2D.

Sin embargo, en la Figura 3.11 se puede observar cómo actuaría el método aplicando diferentes niveles de *hashing* sobre un mismo voxelizado ($\delta = 1$). Si se elige una estructura voxel real con la mitad de posiciones (*hashing* de 50%), entonces la función *Hash* agrupará en una única celda real (en memoria) dos voxels virtuales (los direccionados por los algoritmos de

colisiones), es decir, que la función *H* dará el mismo resultado para dos voxels virtuales distintos.

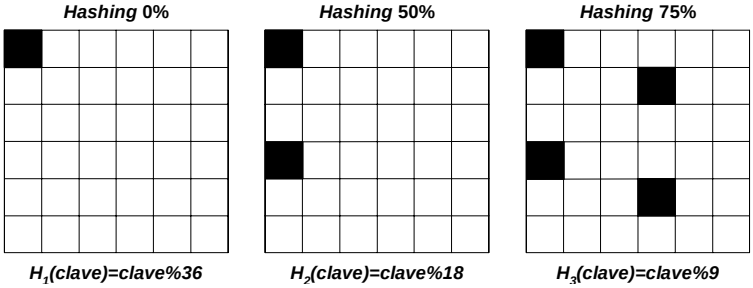


Figura 3.11: Diferentes niveles de *hashing* aplicado a un tamaño fijo de voxel.

El hecho de que varios voxels compartan la misma posición en niveles de *hashing* elevado, como en el ejemplo anterior con un *hashing* 75%, puede parecer arriesgado a la hora de ver el impacto que tendría en el rendimiento del método. Sin embargo, el inconveniente del bajo porcentaje de ocupación juega en este caso un papel favorable.

La fase descrita en este apartado se ejecuta en tiempo constante ya que únicamente se encarga de reservar memoria para las estructuras de datos previamente explicadas. El orden del algoritmo del bloque del preproceso, por tanto, vendrá impuesto por la siguiente fase que se describe a continuación.

3.2.1.3 Asignación de triángulos a voxels

Una vez alocada la memoria requerida por la *estructura Voxel* a partir del volumen inicial de la escena, en esta fase se identifica en qué voxel está contenido, parcial o totalmente, cada uno de los polígonos pertenecientes a la frontera del objeto. Si un triángulo perteneciera a varios voxels a la vez, como se muestra en la Figura 3.12, entonces esos voxels tendrán sendos apuntadores al triángulo en cuestión (Barriuso 1998).

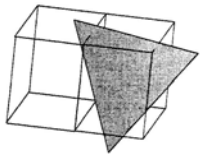


Figura 3.12: Ejemplo de un triángulo contenido en varios voxels a la vez.

La forma más sencilla de realizar este paso es la de comparar cada triángulo de la escena con cada voxel del voxelizado (Held et al. 1995). Sin embargo este procedimiento es muy lento y daría un orden de $O(nN_iN_jN_k)$

siendo n el número de triángulos y $N_i x N_j x N_k$ el número total de voxels. El método se puede acelerar si en lugar de mirar todos los voxels para cada faceta, se mira únicamente aquellos que estén en colisión con el AABB del triángulo en curso. La obtención de este conjunto de voxels es inmediata y la fase de asignación se acelera considerablemente. En este caso la complejidad del algoritmo sería $O(nv_m)$ donde v_m será muy pequeño y corresponde con el número medio de voxels intersectados con el AABB del triángulo.

Existen varias soluciones al problema de averiguar si un triángulo intersecta con una caja. Por ejemplo, Voorhies (1992) presentaba un algoritmo de intersección triángulo-cubo. Posteriormente, Greene (1994) ofrecía otro método para polígonos convexos, y también Green y Hatch (1995) generalizaban y mejoraban el algoritmo para el caso de cualquier polígono.

En la presente Tesis se utiliza un método implementado por Akenine-Möller (2001) que da mejores resultados. El método se basa en el teorema del eje separador (explicado en el apartado 1.4.1.2). La implementación de Barriuso (1998) se basaba en la comparación del triángulo con cada una de las caras del voxel. Sin embargo, experimentos realizados (Figura 3.13) demuestran que el algoritmo de Akenine-Möller es más eficiente que el de Barriuso. Estos tiempos corresponden al tiempo que tardaría en ejecutarse el bloque de preproceso.

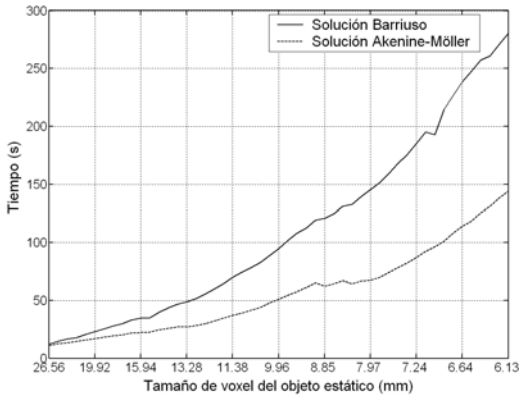


Figura 3.13: Comparación en un rango de niveles de voxelizado del algoritmo de Akenine-Möller y el utilizado por Barriuso.

Esta fase no tiene mayor problema. Una vez asignados los triángulos a cada uno de sus voxels correspondientes, el mallado de voxels creado está preparado para la detección de las colisiones en tiempo de ejecución.

3.2.2 Bloque del Tiempo de Ejecución

Un algoritmo de detección de colisiones depende de las características específicas de cada aplicación. Sin tener en cuenta el consumo de memoria que se ha visto mejorado con las soluciones propuestas en el apartado anterior, en el apartado 2.4 se demostraba que el algoritmo que mejor se comportaba en cuanto a rendimiento era el MVV.

Como ya se comentaba, este algoritmo da buenos resultados para el análisis de mecanismos (García-Alonso et al. 1994), sin embargo, las características específicas que posee el problema de matenibilidad (densidad, número de objetos, número de polígonos) obligan a cambiar de técnica y tener en cuenta otras investigaciones en esta área. Gracias a la Figura 2.16 se podía comprobar que los tiempos obtenidos en simulaciones de mantenibilidad (usando los algoritmos vistos en el apartado anterior) no cumplen los objetivos impuestos para la interactividad. Esto es debido a una serie de razones que se exponen a continuación.

- En el análisis de mecanismos, los objetos implicados están compuestos de miles de polígonos a diferencia de las simulaciones de mantenibilidad que poseen modelos CAD muy complejos y precisos en los que se puede llegar a tener millones de triángulos. Este salto en el orden de magnitud implica un mayor coste computacional de pares de primitivas.

- En el análisis de mecanismos, el estado más habitual es el de no colisión, en cambio en mantenibilidad ocurre justo lo contrario. Analizando con más detalle el algoritmo, en la primera etapa se recorría la estructura de voxels del objeto móvil y para cada voxel se miraba qué voxels estáticos estaban intersectando con él. Sin embargo, este filtro, en análisis de mantenibilidad, genera una explosión de pares de voxels en todas las posiciones ya que la herramienta siempre está dentro del volumen de la escena por lo que cada uno de sus voxels siempre dará un conjunto de voxels estáticos.

- Enlazando con el anterior punto, en el trabajo de García-Alonso et al., sí que tenía sentido voxelizar todos los objetos ya que estos por defecto están separados, y si se intersectan el número de voxels implicados será pequeño. Sin embargo en las simulaciones de mantenibilidad, esta etapa puede ser obviada y pasar directamente a manejar los triángulos del objeto móvil. Esta idea es la base del método que se propone a continuación.

3.2.2.1 Método híbrido de voxels-facetas (MVF)

La principal diferencia entre el método propuesto (MVF) y el algoritmo MVV radica en decidir si el objeto móvil debe poseer o no una *estructura Voxel* asociada (en ambos métodos, el objeto estático es voxelizado de igual manera).

Si se prescinde de la *estructura Voxel* para el objeto móvil, tampoco hace falta las estructuras asociadas, *estructura VoxTri* y las *listas Puntero*, que identifican qué voxels contienen a qué facetas, por lo que la única estructura asociada al objeto móvil es la *estructura TriCol*. La memoria ahorrada con esta solución no es significativa en comparación con la memoria consumida por la maqueta estática.

Con este método, únicamente hay que recorrer la *estructura TriCol* del objeto móvil y comparar cada triángulo con la malla de voxels del objeto estático. Los pasos a seguir se detallan a continuación y tal y como se veía en el esquema general del método de la Figura 3.1, la etapa de ejecución se puede dividir en dos fases: la *broad phase* y la *narrow phase*.

3.2.2.2 Broad phase

La finalidad de la *broad phase* es la de encontrar aquellos objetos o primitivas con probabilidades de colisión, descartando todas las zonas de trabajo sin interés. En los problemas *n*-body, gracias a esta fase se consigue evitar en la mayoría de casos la complejidad cuadrática de la comprobación de todas las parejas de objetos. En los problemas *n*-body estáticos y que utilizan técnicas de partición espacial, lo que se consigue es reducir el volumen de interés.

Partiendo de la representación geométrica de los objetos y del modelo de partición espacial del objeto estático, esta fase consigue reducir el problema de buscar la colisión a un conjunto reducido de voxels. A este conjunto reducido se le llamará voxels objetivo o V_o.

Este conjunto V_o se puede conseguir de varias formas. Por ejemplo, García-Alonso et al. (1994) encuentran el conjunto de pares de voxels intersectados entre sí entre cada par de objetos de la escena. Por el contrario, Held et al. (1995) no utilizan voxels para el objeto móvil y buscan el conjunto de voxels que intersectan con el AABB del objeto móvil de la forma:

$$V_o = \{v \in V \mid v \cap AABB_m \neq \varnothing\} \tag{3.5}$$

donde V es el conjunto total de voxels de la escena y $AABB_m$ es la caja paralela a los ejes que engloba al objeto móvil.

Aunque esta solución consigue reducir el problema considerablemente, el conjunto de voxels objetivo se puede minimizar todavía más ignorando este paso. En lugar de tomar el objeto móvil en su totalidad y mirar los voxels intersectados con su volumen contenedor, se puede pasar directamente a recorrer su *estructura TriCol*. De esta forma se testea qué conjunto de voxels intersectan con cada triángulo, generándose un conjunto más pequeño de voxels al que se llamará $V_{o,t}$. Una vez que un triángulo tiene su propio conjunto de voxels, se pasaría a la *narrow phase* para el testeo entre ese triángulo y todos los asociados con su $V_{o,t}$.

En este caso, si T_m es el conjunto total de triángulos del objeto móvil y $AABB_t$ es la caja contenedora del triángulo t, el V_o total viene definido por la unión de todos los $V_{o,t}$.

$$V_{o,t} = \{v \in V \mid v \cap AABB_t \neq \varnothing\}$$
$$V_o = \bigcup_{t \in T_m} V_{o,t} = \{v \in V \mid \exists t \in T_m, v \cap AABB_t \neq \varnothing\} \quad (3.6)$$

Este nuevo V_o tendrá muchos voxels repetidos ya que varios triángulos pueden compartir varias celdas pero este número es menor al número de pares de voxels generado con el método MVV (García-Alonso et al.) o al método de Held et al. que utiliza un paso previo para calcular los voxels intersectados con el AABB del objeto móvil. Esta comparación puede verse en la Figura 3.14.

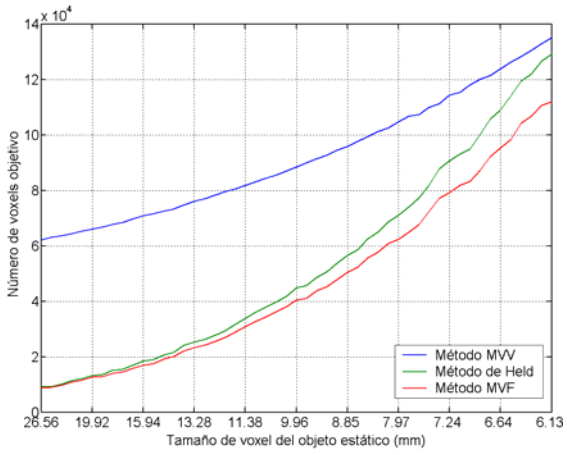


Figura 3.14: Comparación del número de voxels objetivo obtenido con los métodos MVV, MVF y el método de Held et al.

La idea básica de este método y que se diferencia de los métodos comentados anteriormente es la de pasar directamente a la geometría del objeto

móvil. La pregunta que ha llevado a utilizar este método finalmente podría ser: "¿Por qué subdividir el objeto móvil o realizar cálculos con su caja contenedora si al final siempre hay que llegar al nivel de precisión más bajo?".

Unas últimas consideraciones son realizadas antes de pasar a describir la siguiente fase:

- El conjunto V_o del objeto móvil se consigue reducir ya que ahora sólo se tiene en cuenta el AABB de cada triángulo, al contrario que otras soluciones que además de este conjunto, calculan también el correspondiente al AABB del objeto móvil en su totalidad (Held et al. 1995).

- Para reducir todavía más el conjunto V_o, se podría coger cada triángulo y testear exactamente qué voxels intersectan con él pero esta solución sería más costosa computacionalmente en comparación con la solución actual que utiliza primitivas simples como el AABB que engloba a cada triángulo.

3.2.2.3 Narrow phase

En la *narrow phase*, como entrada se tiene el conjunto de triángulos estáticos pertenecientes a los conjuntos $V_{o,t}$ y el triángulo t que intersecta con ese conjunto. Entonces se ejecuta el chequeo entre pares de triángulos para detectar su intersección o no. Aunque para simplificar, en el diseño aparecen las dos fases (*broad* y *narrow*) secuenciales, en realidad se ejecutan en un bucle con tantos ciclos como triángulos tenga el objeto móvil. En la Figura 3.15 aparece el flujo de ejecución de estas dos fases.

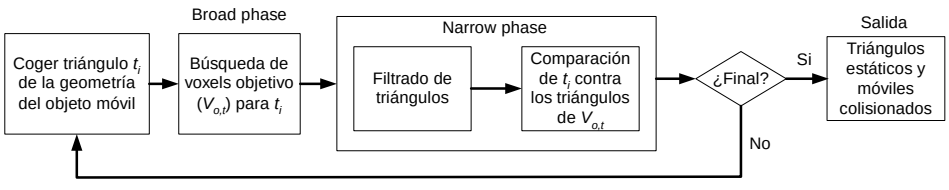


Figura 3.15: Flujo de ejecución de la *broad phase* y la *narrow phase.*

Como se puede apreciar, por cada triángulo t_i se calcula su $V_{o,t}$ y se pasa a la *narrow phase* de ese triángulo t que consiste en chequear su posible colisión con todos los triángulos contenidos en $V_{o,t}$. Una vez finalizado el cálculo de t, se volvería al inicio del ciclo y a calcular otro conjunto de voxels objetivo para el siguiente triángulo.

El método usado para detectar la intersección entre dos triángulos es el implementado por Möller (1997). Este algoritmo ejecuta primero una serie de tests simples para descartar rápidamente la no colisión. Estos tests se basan en mirar si todos los puntos de un triángulo se encuentran en un mismo lado del plano que contiene al otro triángulo y viceversa. Si los triángulos pasan este test, entonces el algoritmo resuelve el problema computando las proyecciones de los triángulos en la intersección de sus planos (para un procesador Pentium 3 Xeon a 866 MHz, el algoritmo logra ejecutar 10^5 testeos en un segundo).

Antes de probar experimentalmente los rendimientos de los algoritmos, se puede expresar el orden del método MVF como ya se hizo para el resto de métodos implementados en el capítulo anterior. Si se considera que en este caso el número de voxels objetivo, v_o, es el cardinal del conjunto $V_{o,t}$, entonces la expresión (3.7) ofrece el orden del método propuesto.

$$O(t_{ejec}) = O(n_m) \times O(1) \times O(v_o) \times O(f_e) = O(n_m v_o f_e) \tag{3.7}$$

Esta expresión será el punto de partida para la discusión ofrecida en el siguiente capítulo sobre la búsqueda del voxelizado óptimo de una escena.

Para ver el rendimiento que da el método propuesto MVF frente al algoritmo MVV, en la Figura 3.16 se muestra el tiempo de cálculo para distintos niveles de subdivisión. Para el MVV se ha escogido varios niveles de subdivisión del objeto móvil que ofrecían resultados óptimos en los análisis recogidos en el capítulo anterior.

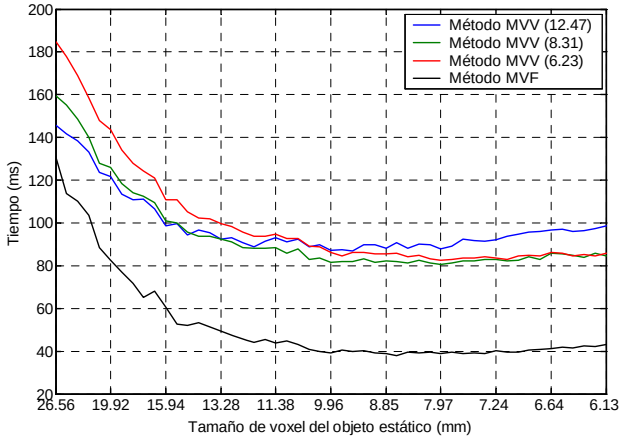


Figura 3.16: Comparación de rendimiento entre los métodos MVF y MVV.

Sin embargo, todavía se puede realizar una serie de optimizaciones antes de pasar al chequeo final entre pares de triángulos. Estas optimizaciones del

método MVF se corresponden con la etapa del filtrado de triángulos que aparece en el esquema de la Figura 3.1 y para ello se han diseñado dos filtros: la detección de cálculos repetidos y la discriminación por AABBs de triángulos.

Detección de cálculos repetidos

Añadiendo dos flags (identificador de *frame* e identificador de triángulo) en la *estructura* T*riCol*, se puede detectar en cada *frame* si una pareja de triángulos ya ha sido testeada o no dentro de ese mismo *frame*. Este par de flags aumenta en un 17.4% la memoria consumida por esta estructura. Como ejemplo, para un modelo de 1615172 polígonos, el consumo se incrementaría en unos 12 MB. Este aumento no es significativo y ya se tuvo en cuenta en el apartado del bloque del preproceso cuando se mostró la memoria consumida por el método.

En la Figura 3.17 se puede comprobar el buen funcionamiento de este filtro. La gráfica representa el número total de pares de triángulos chequeados y colisionando. Con el método original y sin filtrado, el número va creciendo según se aumenta el nivel de voxelización, algo esperado ya que conforme más fina sea la malla de voxels, un mismo triángulo estará contenido en más voxels y por tanto más cálculos repetidos aparecerán. Sin embargo, aplicando el filtro de detección de cálculos repetidos, el número de pares permanece constante en cualquier nivel de voxelización.

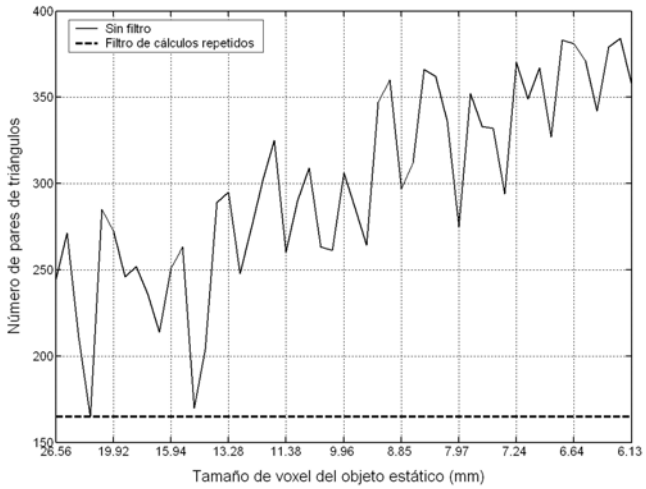


Figura 3.17: Comparación del número de pares de triángulos testeados y en colisión con y sin el filtrado de la detección de cálculos repetidos.

La variación tan alta en el número de pares si se considera la opción sin filtrado se debe a que dependiendo del tamaño del voxel, el número de pares de

triángulos repetidos puede variar. De todas maneras se puede observar que la tendencia es a crecer según se afina el mallado, algo que ya se esperaba.

Discriminación por AABBs de triángulos

Antes de chequear la interferencia entre un par de triángulos mediante el algoritmo de Möller, primero se miran sus volúmenes contenedores. Si sus AABBs no están solapándose, los triángulos tampoco lo estarán. Este test es muy rápido y puede discriminar una gran cantidad de cálculos innecesarios. Al contrario que el anterior, este filtrado está destinado a discriminar pares de triángulos no colisionables, por lo tanto para validar el filtro, en la Figura 3.18 se representa el número total de pares de triángulos testeados.

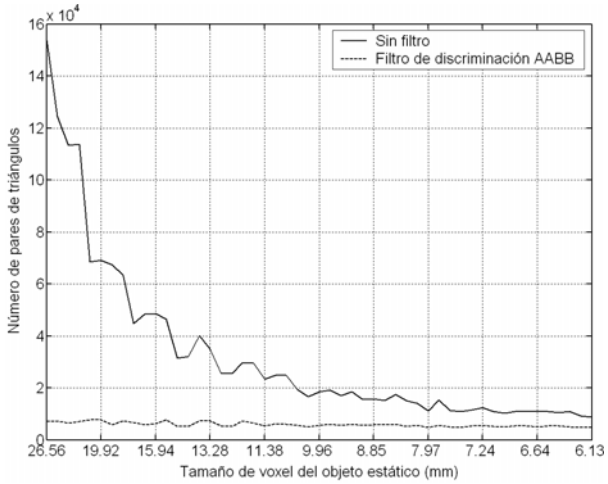


Figura 3.18: Comparación del número final de pares de triángulos chequeados con y sin el filtrado de discriminación por AABBs de triángulos.

Este filtro reporta mayor porcentaje de ganancia que el anterior llegando incluso, con algunos tamaños de voxel, a bajar en un orden de magnitud el número total de pares testeados. Por otro parte es lógico ya que es conocido que en la detección de colisiones la mayor parte del tiempo se consume en testear pares de triángulos que realmente no están intersectándose. Por lo tanto, todos los chequeos orientados a detectar y discriminar esta situación reportarán ganancias significativas.

Como se ha descrito en los párrafos anteriores, cada filtro va destinado a mejorar aspectos diferentes del algoritmo. Mientras que el primer filtro de los pares repetidos actúa sobre todos los triángulos contenidos en los conjuntos $V_{o,t}$; el segundo filtro actúa únicamente sobre los triángulos que no están en colisión.

En la siguiente gráfica (Figura 3.19) se puede apreciar los diferentes rendimientos dependiendo de qué tipo de filtrado se esté haciendo uso.

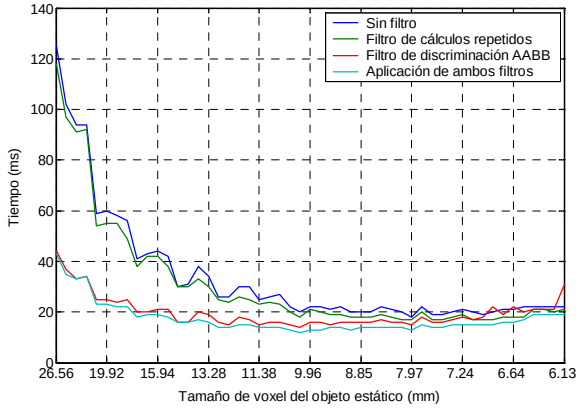


Figura 3.19: Diferencias en el rendimiento del método utilizando diferentes filtrados.

Para que el estudio sea más real, al analizar esos resultados nos restringiremos al rango de voxelización situado en la zona óptima para este ejemplo (12.41 a 8.85; confrontar con el capítulo 4). El primer filtro mejora en un 10.5% al método original mientras que el segundo filtro mejora en un 33.63%. Aplicando los dos filtros, la mejora en el rendimiento asciende un 42.34%.

Los rendimientos arriba mostrados están influidos por el tiempo consumido de la *broad phase* por lo que para analizar de forma correcta el rendimiento de los filtros es mejor fijarse solamente en el número de pares testeados. Si se coge la media de estos pares en el rango definido en el párrafo anterior, se puede mostrar el número total de pares testeados dependiendo del tipo de filtro utilizado. En la Tabla 3.1 se recoge este número y además se muestra el porcentaje real de los pares encontrados en colisión y los no colisionables.

Filtro	Pares testeados	Pares en colisión	Pares en no colisión
Sin filtro	21324	1.4 %	98.6 %
F. cálculos repetidos	17561	0.94 %	99.06 %
F. discriminación AABB	5837	5.1 %	94.9 %
Ambos filtros	3949	4.18 %	95.82 %

Tabla 3.1: Porcentaje de pares de triángulos colisionando y no colisionando respecto del total.

Aunque se apliquen los distintos filtros, es apreciable que el mayor porcentaje del tiempo sigue correspondiendo a pares de triángulos que al final

no están colisionando (96% aproximadamente). Sin embargo, aplicando los dos filtros, el número total de pares disminuye un orden de magnitud lo que se nota en el rendimiento final del método (como se comprueba en la Figura 3.19).

Para resumir, en la Figura 3.20 se compara el rendimiento entre el óptimo del método MVV, el método MVF sin filtros y el MVF finalmente implementado con los dos filtros descritos anteriormente.

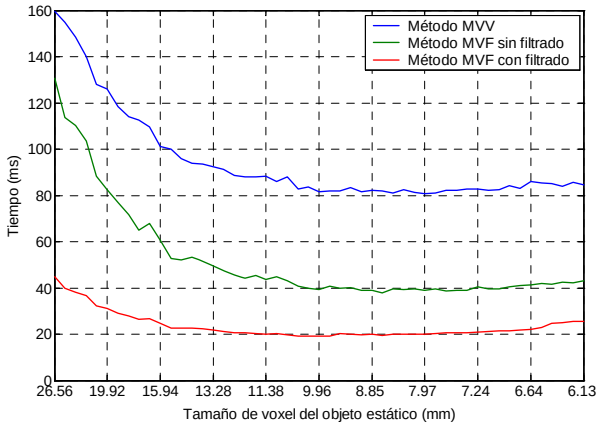


Figura 3.20: Comparación de rendimiento entre los métodos MVV y MVF con y sin filtros.

La salida del método propuesto son dos conjuntos de triángulos en colisión, uno perteneciente al objeto estático y otro al objeto móvil. Con esta información es posible devolver un feedback visual, acústico y táctil al exterior. El feedback visual y acústico es casi inmediato con la información de los triángulos en intersección, sin embargo, dar una respuesta táctil no es un problema trivial por lo que es estudiado en detalle en el capítulo 5.

3.2.3 Geometría del Voxel: cúbica o paralelepípeda

Una última consideración son las dos posibilidades al construir un mallado de voxels: la elección entre geometría cúbica o paralelepípeda. La conclusión a la que se ha llegado una vez estudiado todos los experimentos es que para un consumo de memoria similar, los tiempos obtenidos en cada uno de los métodos no difieren significativamente entre sí, al menos cuando los lados del paralelepípedo tienen un ratio menor de 2. Un ejemplo concreto se muestra en la Figura 3.21 dónde aparecen las dos curvas de tiempos obtenidas con los dos métodos. En una de ellas el eje *x* representa el grano de celda mientras que en la otra curva representa el número de voxels en cada dimensión.

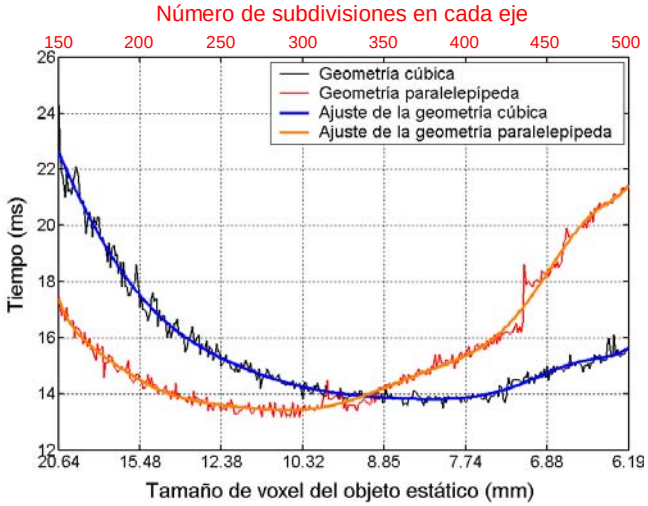


Figura 3.21: Rendimiento del método según la geometría del voxel.

Para tomar un valor de referencia, primero se ajusta cada curva con funciones polinómicas por el método de los mínimos cuadrados. Como valor de referencia se puede tomar el mínimo en cada función y comprobar el rendimiento y la memoria consumida por cada método. En la Tabla 3.2 aparecen estos datos.

Geometría	Mínimo	Tiempo (ms)	Memoria (MB)
Cúbica	8.17 mm	13.8	187
Paralelepípeda	291 subdivisiones	13.4	191

Tabla 3.2: Comparación del tiempo y memoria consumida por los dos métodos en el punto mínimo de cada función.

Todos los experimentos realizados en la presente Tesis se han llevado a cabo con los dos métodos, sin embargo, vistos los resultados tan similares, se ha optado por ofrecer en la mayoría de los ejemplos, las gráficas pertenecientes a los voxels cúbicos por ser esta geometría más natural y homogénea.

3.2.4 Cambios de configuración

Ya ha sido citado que el marco de desarrollo del método propuesto es una aplicación de simulación de tareas de mantenibilidad. Por lo tanto, durante una simulación, las condiciones del cálculo de colisiones pueden variar. La condición normal será el cálculo de un solo par de objetos, la herramienta contra el resto del entorno (este par de objetos se introduce al inicio de la simulación). Sin embargo, la simulación estará regida por acciones de usuario que podrá

añadir o eliminar objetos al cálculo de las colisiones. Estas acciones son mostradas en la Figura 3.22 dónde se representa el flujo de una simulación.

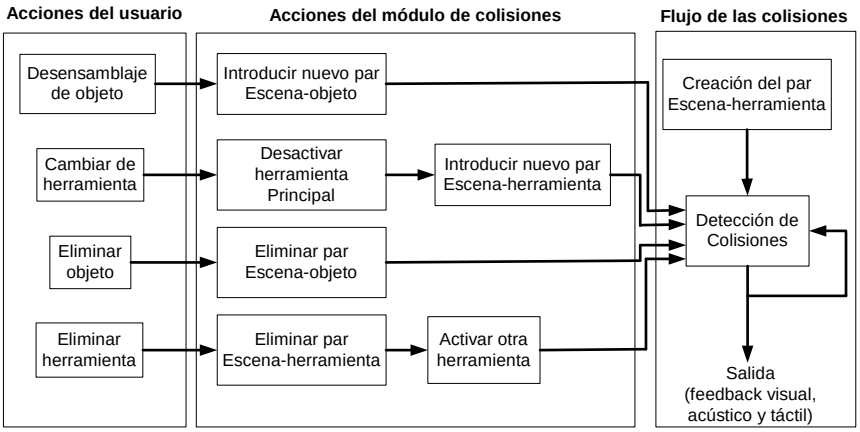


Figura 3.22: Acciones del usuario y flujo de ejecución del módulo de colisiones en una simulación de mantenibilidad.

En cualquier momento de la simulación el usuario puede desensamblar un objeto cualquiera (tornillos, tuercas, tubos, etc.) y por lo tanto se tiene que tener en cuenta también el cálculo de las colisiones de ese elemento contra la maqueta (además del par existente escena-herramienta). En la Figura 3.23 se muestra un ejemplo.

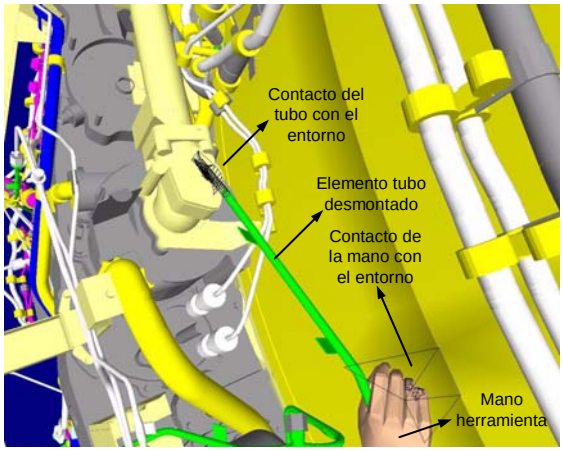
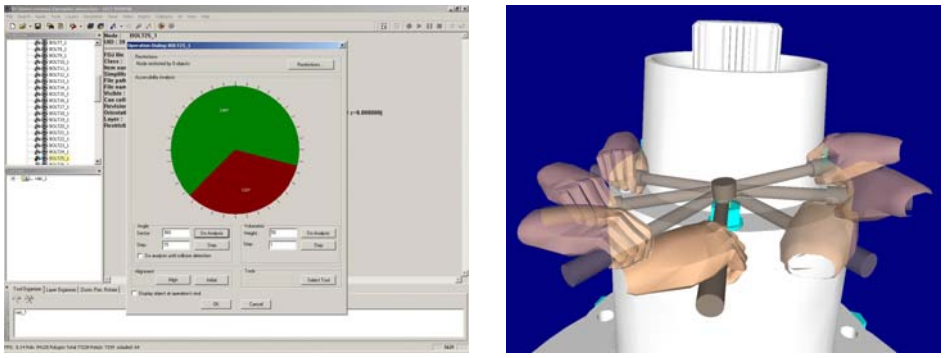


Figura 3.23: Colisión de un elemento desmontado con su entorno.

Otra posibilidad es el cambio de herramienta que también tiene que estar contemplado. En este caso la herramienta principal queda como secundaria por si el usuario quiere volver a usarla más adelante.

Estas acciones necesitan ser interactivas para no entorpecer la tarea del usuario. La ejecución de cualquier acción por parte del usuario se realiza sin retardo visual perceptible gracias al método propuesto que no voxeliza el objeto móvil. Únicamente se crea la *estructura TriCol* vista anteriormente y que no tiene un tiempo de preproceso significativo. Hay que tener en cuenta que los objetos a insertar en el cálculo de colisiones tendrán un número de polígonos muy por debajo del escenario global (cientos o miles en el caso de tubos).

Los algoritmos y el software desarrollado además de integrarse en un sistema de mantenimiento virtual, han servido también para desarrollar módulos de análisis de accesibilidad. En la Figura 3.24 se muestra un ejemplo de este tipo de operaciones. El sector libre de colisiones aparece en verde al realizar un análisis de 360º en una herramienta.



(a) (b)

Figura 3.24: Análisis de accesibilidad de una herramienta y su entorno. En (a) aparece la GUI indicando el sector libre de contacto (en verde) y en (b) se representan posiciones de la herramienta a analizar.

Todas estas acciones hacen posible la interacción del usuario con el sistema de realidad virtual en tiempo real, requisito para lograr simular interactivamente las tareas de mantenibilidad y accesibilidad requeridas.

3.3 REFERENCIAS

Akenine-Möller, T., “Fast 3D Triangle-Box Overlap Testing”, *Journal of Graphics Tools (JGT)*, Vol. 6, No. 1, pp. 29-33, 2001.

Barriuso, J.R., "Cálculo de la Distancia entre Objetos representados con precisión mediante Poliedros Cóncavos durante una Simulación Mecánica", *Tesis Doctoral*. Universidad de Navarra. Octubre 1998.

Elmasri, R. y Navathe S.B., "Sistemas de Bases de Datos. Conceptos Fundamentales, 2ª Edición", *Addison-Wesley Iberoamericana, S.A.* 1997.

Foley, J.D., Dam, A. Van, Feiner, S.K. and Hughes, J.F., *Computer Graphics principles and practice*. Second Edition, Addison-Wesley, 1990.

García-Alonso, A., Serrano, N. y Flaquer, J., "Solving the Collision Detection Problem", *IEEE Computer Graphics and Applications*, Vol. 13 (3), pp. 36-43. Mayo 1994.

Green, D. y Hatch, D., "Fast Polygon-Cube Intersection Testing", *In Graphics Gems V*, Academic Press, pp. 375-379, 1995.

Greene, N., "Detecting Intersection of a Rectangular Solid and a Convex Polyhedron", *In Graphics Gems IV*, Academic Press, pp. 74-82, 1994.

Gregory, A., Lin, M.C., Gottschalk, S. y Taylor, R., "Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback", *Computational Geometry – Theory & Applications* 15, pp. 269-285, 2000.

Held, M., Klosowski, J.T. y Mitchell, J.S.B., "Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs", *Proceedings of the Seventh Canadian Conference on Computer Geometry*, Vol. 3, pp. 205-210, Québec City, Québec, Canada. Agosto 1995.

Möller, T., "A Fast Triangle-Triangle Intersection Test", *Journal of Graphics Tools (JGT)*, Vol. 2, No. 2, pp. 25-30. 1997.

Voorhies, D., "Triangle-Cube Intersection", *In Graphics Gems III*, Academic Press, pp. 236-239, 1992.

CAPÍTULO 4

CÁLCULO DEL VOXELIZADO ÓPTIMO

4.1 INTRODUCCIÓN Y NOTACIÓN PREVIA

Al utilizar las técnicas de partición espacial aparece un primer problema muy claro. A la hora de discretizar el espacio 3D y generar la estructura de voxels (voxelización o mallado de celdas), se tiene que elegir el número de voxels en cada dimensión. Dependiendo de ese número, el algoritmo de detección de colisiones puede comportarse muy eficientemente o por el contrario puede consumir tiempos que impidan su uso en tiempo real.

Independientemente de la problemática del espacio en memoria y suponiendo que se tiene memoria suficiente para cualquier prueba, el óptimo será aquel cuyo coste en tiempo de cálculo sea mínimo para el computador. Es decir, que la suma del coste de todas las fases del método sea mínima. Además, este óptimo estará acotado por los siguientes sucesos:

- Si la voxelización es demasiado gruesa comparada con el tamaño de los triángulos del modelo, entonces en cada celda habrá un gran número de polígonos. El resultado puede ser demasiados chequeos de intersección entre primitivas.

- Si el mallado es muy fino en comparación con el tamaño de los triángulos del modelo entonces el problema es la explosión

combinatoria que aparece entre los pares voxel-triángulo que además provoca otra explosión triángulo-triángulo.

Los casos anteriores son los dos extremos y lo ideal sería encontrar un método que automáticamente calculara el grado de voxelizado óptimo para un modelo cualquiera. El problema es que este número depende de muchos factores y es difícil hallar un método analítico.

Para empezar, se enumeran los parámetros que inicialmente se pensaba que afectan al tiempo de cálculo de colisiones en un instante dado. Posteriormente, al ir presentando los resultados obtenidos en los experimentos se irá afinando la definición de los parámetros que afectan en la elección del nivel de discretización.

- *Teselación de los modelos*: no es lo mismo que un modelo esté teselado uniformemente, es decir, que el número de triángulos por unidad de área sea constante, a que no sea uniforme.

- *Zona de prueba*: el tiempo de cálculo depende de en qué zona del modelo se realice la prueba. No es lo mismo en una zona con alta densidad de triángulos que en otra con poca.

- *Volumen de contacto*: los tiempos tampoco son los mismos si la herramienta penetra profundamente dentro del modelo estático o si sólo está tocando con un extremo.

En el caso del mantenimiento virtual el volumen de contacto queda restringido por el uso de una interfaz háptica. La restitución de esfuerzos realizada por esta interfaz impedirá siempre un grado de penetración elevado. Como por otra parte el estado de no colisión es siempre más rápido de evaluar, la búsqueda del óptimo deberá centrarse para estados de colisión, pero poco profunda. En definitiva, el volumen de contacto no será un parámetro para determinar la voxelización óptima, sino para determinar la posición relativa de la herramienta que se utilizará para validarla.

Antes de proseguir, es necesario definir una serie de términos que van a ser utilizados a lo largo de todo el capítulo.

- Ya que con el método MVF no existe voxelización para el objeto móvil, las referencias a voxels serán exclusivamente para referirse a las celdas generadas para el objeto estático.

- En el capítulo anterior se hablaba de las dos implementaciones llevadas a cabo dependiendo de la geometría usada para la definición

del voxel. Por lo tanto, de aquí en adelante, el significado de *nivel de voxelización* dependerá de la geometría usada. En el caso del uso de voxels cúbicos se estará haciendo referencia al tamaño del lado o arista del cubo. Todos los parámetros y variables dependientes de esta geometría se denotarán con el superíndice *c*. Por otro lado, si se usan los voxels paralelepípedos, entonces se estará hablando del número de voxels en cada dimensión y el superíndice asociado será *p*.

- Grano de celda o tamaño de voxel (δ): continuando con lo anterior, este parámetro sólo se usará con el método de voxels cúbicos. Denota el tamaño del lado o arista del voxel.

- Número de subdivisiones o número de voxels en cada eje (N): en el método de voxels paralelepípedos, al tener distinto grano de celda cada dimensión se usa este otro parámetro homogéneo para las tres dimensiones.

- Nivel de voxelización óptimo experimental: corresponde con el valor óptimo encontrado mediante experimentos. Según el método usado, se denotará por grano óptimo (δ_{opt}) o número de subdivisión óptima (N_{opt}).

- Zona óptima experimental: es un intervalo de niveles de voxelización que se toman como buena aproximación al nivel de voxelización óptimo. También se obtiene mediante experimentación y al ser un intervalo su notación será [$\delta_{\min}$, $\delta_{\max}$] o [N_{min}, N_{max}].

- Voxels objetivo (v_o): número medio de voxels que están intersectando con el AABB de un triángulo del objeto móvil. Enlazando con el capítulo anterior, este número corresponde con el cardinal del conjunto $V_{o,t}$ que se definía en la ecuación (3.6).

- Facetas por voxel (f_e): número medio de triángulos del objeto estático contenidos parcial o totalmente dentro de un voxel.

- Volumen contenedor global: hace mención a la caja paralela a los ejes (AABB) que engloba a toda la escena, es decir, a la maqueta virtual aeronáutica. Sus dimensiones se denotan por X, Y y Z.

4.2 SOLUCIONES EXISTENTES AL PROBLEMA

Determinar el mallado óptimo no es una tarea trivial por el número de parámetros que pueden afectar al rendimiento de la aplicación (Gregory et al. 2000 y Zhang y Yuen 2002).

Existen trabajos en los que se plantea el problema del tamaño de voxel a escoger, pero en todos ellos se propone una solución basada en experimentos propios. Por ejemplo Held et al. (1995) encuentran que para sus propios modelos, el rango óptimo se encuentra entre los niveles 5x5x5 y 50x50x50. Sin embargo García-Alonso et al. (1994) mantienen un OBB para cada modelo en la escena. Estos OBBs son subdivididos a su vez en voxels. En sus experimentos encontraron que un nivel óptimo de discretización era subdividir cada OBB en 512 celdas o voxels. Otros autores como McNeely et al. (1999), determinan la discretización del espacio en función de la precisión que se quiera alcanzar, sin realizar consideraciones de optimización. En sus experimentos voxelizan con una precisión de 5 mm.

En Gregory et al. (2000) se dan dos aproximaciones al problema: una experimental y otra analítica. En la primera afirman que la voxelización óptima viene dada por un tamaño o grano de celda igual a la media de las longitudes de todas las aristas de la escena multiplicada por una constante. Esta constante es un parámetro dependiente del modelo y se elige basándose en una serie de experimentos.

El problema sigue siendo que la fórmula de Gregory continua dependiendo de una serie de experimentos que se tienen que realizar en un proceso previo a la simulación virtual.

Siendo este trabajo el que más se acerca a conjeturar un posible óptimo al tamaño de voxel de la escena, en los siguientes párrafos se tratará de explicar y describir el método analítico que presentan para comprobar que ese método no está completo y por lo tanto no sirve como solución analítica general.

Siendo P el modelo poligonal representando el entorno en $\mathbb{R}^3$ como una colección de N triángulos, su solución se basa en una serie de suposiciones:

- La posición actual del dispositivo háptico estará muy cercana a la anterior, de tal manera que el *path* barrido entre las dos posiciones intersectará como mucho en S celdas donde $S \approx 1$.

- P está teselado uniformemente, es decir, el número de triángulos por unidad de superficie es constante a lo largo de P.

- Ya que se considera que únicamente está en contacto la punta del háptico, en un momento dado sólo habrá un triángulo en contacto con esa punta.

Estos postulados son muy rígidos y no se cumplen en el caso de la presente Tesis. La primera suposición geométricamente es casi imposible ya que en el cálculo de las colisiones se tiene en cuenta todo el modelo 3D de la herramienta virtual y no únicamente un punto.

Los modelos usados son exportados directamente de CAD sin teselar uniformemente. Y aunque se realice un post-proceso de teselación, los requerimientos de memoria y tiempos de ejecución se incrementarían notablemente por lo que la segunda suposición tampoco es aceptada.

Por último, la tercera suposición está relacionada con la primera; al tener en cuenta todo un objeto 3D se puede dar, y de hecho sucede, el caso de tener varios contactos a la vez y con múltiples triángulos cada uno de ellos.

A pesar de todo, Gregory et al. calculan una posible solución óptima basándose en estas suposiciones y utilizando costes de ejecución de dos métodos de detección de colisiones: un OBBTree y un mallado uniforme de celdas, siendo C_r y C_g sus costos computacionales asociados respectivamente. Ellos dan una solución híbrida al problema de las colisiones (C_h). Para que este nuevo método sea mejor que los dos anteriores se tendría que cumplir las dos inigualdades siguientes:

$$C_h < C_r \text{ y } C_h < C_g \tag{4.1}$$

Con esta solución aunque logran dar un rango de valores óptimos, este rango corresponde al de un número M, siendo M el número medio de triángulos por voxel de tamaño óptimo. Todavía queda cómo sacar el tamaño óptimo del voxel a partir de M, aspecto que no tratan. Los autores afirman que sigue siendo complejo obtener un valor cercano al óptimo para cualquier modelo. Esto último es lo que se pretende solucionar con el presente capítulo de la Tesis.

Como argumento añadido se tiene el tamaño de los modelos usados. Mientras que en sus experimentos no usan modelos mayores de las decenas de miles de triángulos, en esta Tesis se pretende estudiar un problema con modelos masivos de millones de triángulos.

Una vez demostrado el interés del problema y la ausencia de soluciones, en los siguientes apartados se muestran las dos soluciones propias a las que se ha llegado, experimental una y analítica la otra. En la primera, la selección del nivel depende de cada modelo y se basa en experimentos propios. Para ello se ha desarrollado un software que lanza un análisis automático en el que se

prueba un rango de mallados o voxelizaciones diferentes para un modelo dado, con un objeto móvil en diferentes posiciones y con penetraciones diferentes también entre el objeto móvil y el modelo estático. El intervalo de granularidades o niveles de discretización del mallado a probar se definen paramétricamente al inicio del análisis.

En la segunda solución se ha hallado una fórmula general que suministra un nivel óptimo de voxelización. Finalmente, en el apartado 4.5 se valida esta solución gracias a los resultados conseguidos en la solución experimental.

4.3 SOLUCIÓN EXPERIMENTAL

Como ya se ha dicho anteriormente, mediante el software de análisis automático se puede lanzar una secuencia de pruebas con el objetivo de obtener la tabla de tiempos resultantes y a partir de ella determinar dónde se encuentra la zona óptima. Antes de mostrar los resultados de estas simulaciones, hay que definir primero lo que se entiende por *zona óptima* aunque ya se introducía la idea al inicio del capítulo. En esta zona se encontrará el rango de niveles de voxelización que dan los tiempos mínimos: voxelizaciones de grano más fino o más grueso que la zona óptima requieren tiempos de cálculo notoriamente mayores. Esta zona se estudia trabajando con funciones que aproximan los tiempos obtenidos en el análisis automático.

Para aproximar los resultados de tiempos se usan funciones polinómicas. Estas aproximaciones tendrán la forma:

$$f(x) = c_{n+1}x^n + c_n x^{n-1} + \cdots + c_2 x + c_1 \qquad (4.2)$$

Los tiempos vuelven a incrementarse con niveles altos de voxelización porque a partir de cierto número de subdivisiones, el tiempo consumido por el chequeo de los voxels es mayor que la reducción de tiempo debido a la disminución del número de pares de triángulos generada por dicha subdivisión.

Dependiendo del tamaño del modelo, esta cota superior se puede alcanzar para voxelizaciones con menor o mayor número de celdas (mayor o menor grano). Si los modelos son masivos o con mucho volumen desaprovechado, se requiere una gran cantidad de memoria para almacenar la estructura de voxels. Si no se dispone de suficiente memoria para representar ese dominio, la gráfica aparentemente no presenta un mínimo, sino que parece asintótica ya que no se podrá contemplar los casos en los que los valores del eje de tiempos vuelven a subir. Estos casos se pueden evitan utilizando las técnicas descritas en el capítulo anterior para el ahorro de memoria.

Está claro que hay que tener un criterio para acotar y definir una zona de voxelización óptima teniendo en cuenta además de los tiempos, la memoria consumida. Podemos definir un solo valor óptimo o una zona óptima alrededor de ese valor óptimo. El valor óptimo será meramente simbólico y servirá de referencia para determinar la zona óptima. Cualquier voxelización dentro de la zona óptima se tomará como una buena elección de discretización.

Aunque en el ejemplo de la Figura 4.1 se aproxima una gráfica de tiempos con un polinomio de grado 2, la siguiente consideración es válida para polinomios de grado mayor. Se define como valor óptimo para el nivel de voxelizado el mínimo de la función *f(x)* utilizada como aproximación. Se define la *zona óptima* como los valores más cercanos al óptimo. Como candidatos se han escogido aquellos cuyos tiempos no se eleven más de 1 ms.

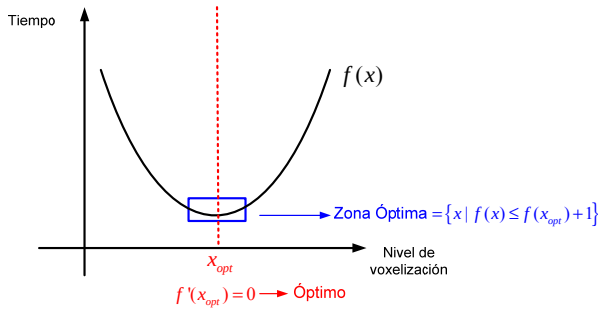


Figura 4.1: Gráfica de tiempos aproximada por una función polinómica de grado 2.

La Tabla 4.1 indica el número de polígonos de los modelos escogidos para los experimentos. Los modelos se pueden ver en la Figura 4.2 y en la Figura 4.3.

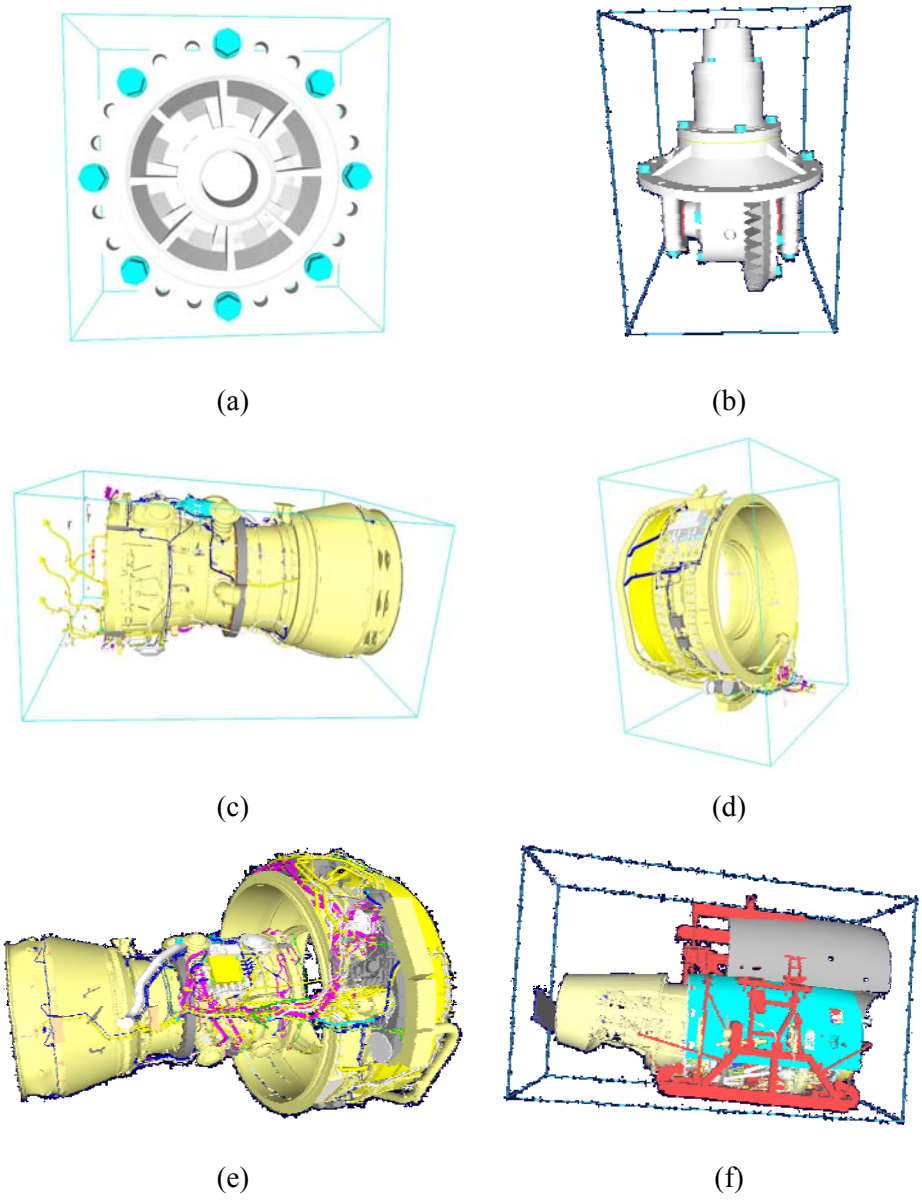
Modelo estático	Polígonos
CarcasaTornillos	51782
Diferencial	149119
Turbina	711014
Externals	841091
Motor	1615172
Tay	3571682

(a)

Modelo de herramienta	Polígonos
Torque	544
Scanner	606
Wrench	770
Square	954
Mano-h0	2687

(b)

Tabla 4.1: Descripción de los modelos estáticos (a) y de herramientas (b).

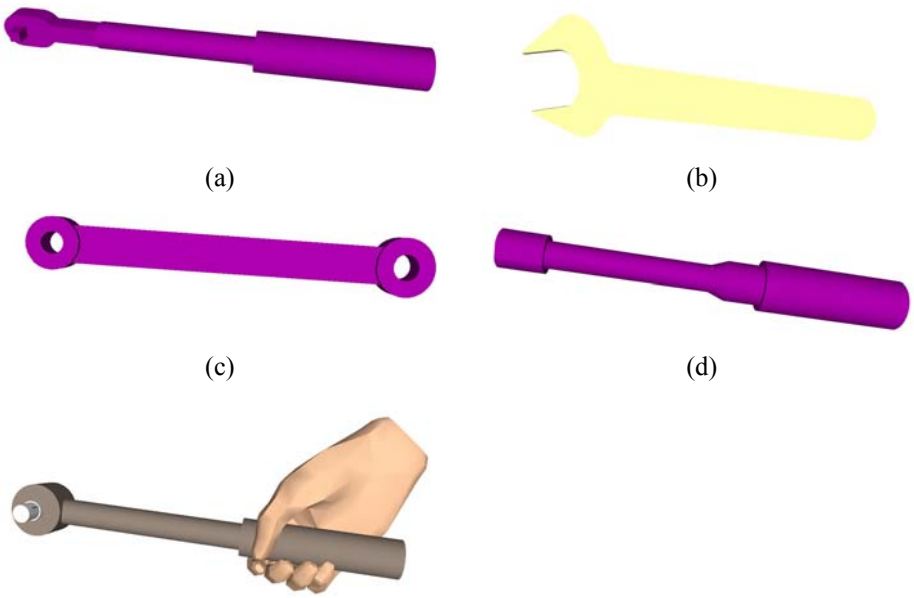


(a) (b)

(c) (d)

(e) (f)

Figura 4.2: Modelos estáticos “CarcasaTornillos” (a), “Diferencial” (b), “Turbina” (c), “Externals” (d), “Motor” (e) y “Tay” (f).



(a) (b)

(c) (d)

(e)

Figura 4.3: Modelos móviles “Torque” (a), “Scanner” (b), “Wrench” (c), “Square” (d) y “Mano-h0” (e).

Antes de continuar se enumeran una serie de criterios que serán utilizados en toda la experimentación.

- Para las pruebas experimentales, se ha lanzado el software de análisis automático en diferentes posiciones (en concreto diez posiciones diferentes) de trabajo de la herramienta u objeto móvil. La media de estas posiciones será la utilizada para generar la gráfica de tiempos.

- Estas posiciones analizadas no han sido escogidas al azar. Se han seleccionado varias posiciones que corresponden con operaciones de mantenimiento realizadas por los usuarios finales (tareas como alinear la herramienta con un tornillo, acceso a zonas complicadas con alta densidad de objetos, etc.). Siempre teniendo en cuenta que no tiene sentido testear posiciones que no se van a reproducir en una tarea de mantenibilidad. Por ejemplo, el caso de tener la herramienta totalmente dentro del modelo y por tanto con todos sus polígonos en colisión. Este caso nunca se dará en la realidad ya que lo impiden las leyes físicas y en la simulación tampoco se dará porque lo impedirá la restitución de esfuerzos por parte del dispositivo háptico. Como

mucho se tendrá uno o varios pequeños contactos con a lo sumo cientos de polígonos en colisión.

- Para mostrar los pasos seguidos en los experimentos, las gráficas que se muestran a continuación corresponden con el modelo estático "Motor" y el modelo de herramienta "Mano-h0". Los pasos para el resto de pruebas son los mismos.

- Estas gráficas se han obtenido para la geometría cúbica de los voxels. Los resultados para voxels no cúbicos serán mostrados también al final del apartado.

En la Figura 4.4 se puede apreciar un ejemplo del rendimiento del módulo de colisiones en tres posiciones diferentes. No se muestran las diez posiciones para no enturbiar la representación gráfica. Por lo que se puede apreciar en la Figura 4.4, los tiempos dependen evidentemente de la posición de los objetos (la zona de prueba que se menciona al inicio del capítulo), pero para calcular la zona óptima de voxelización lo que interesa es la forma parabólica de la curva y eso es algo que no varía significativamente en las distintas posiciones de los objetos.

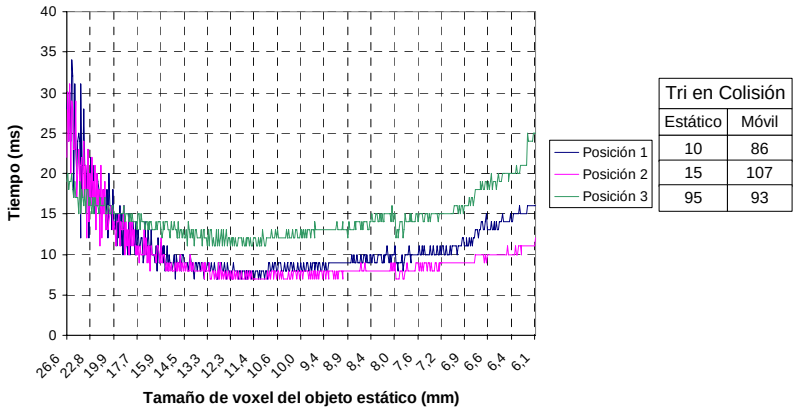


Figura 4.4: Rendimiento con el modelo "Motor" para tres posiciones diferentes de la "Mano-h0".

Ahora se puede coger la media de los 10 tiempos tomados y aproximar la curva con la función polinómica que más interese como se muestra en la Figura 4.5.

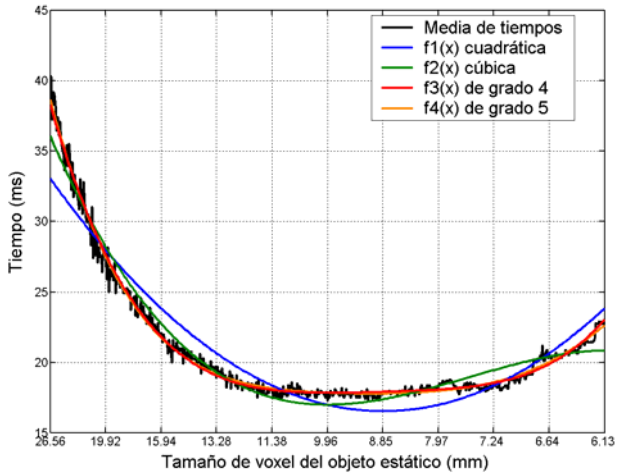


Figura 4.5: Media de tiempos y diferentes aproximaciones con el modelo "Motor" y con la herramienta "Mano-h0".

Para poder saber cuán buena es la aproximación del polinomio a la gráfica de tiempos, una medida posible es el valor de R^2 o coeficiente de determinación obtenido mediante mínimos cuadrados Este coeficiente coge valores entre 0 y 1 y cuantifica lo cercanos que están los valores estimados de la función polinómica de los datos reales. Cuanto más se acerque a 1 mejor será la aproximación.

Una buena aproximación a los datos reales puede ser un valor de R^2 comprendido entre $0.95 \le R^2 \le 1$. Como criterio de elección se escoge el polinomio i donde $R_i^2 - R^2_{i-1} \le 0.01$. Polinomios de mayor grado que ese i, tendrán un coeficiente de determinación muy similar al R_i^2, es decir, las funciones de aproximación serán muy similares.

Para el ejemplo anterior del modelo "Motor", los coeficientes R^2 son los siguientes:

$$\begin{aligned} R^2_{f_1(x)} &= 0.897328 \\ R^2_{f_2(x)} &= 0.960669 \\ R^2_{f_3(x)} &= 0.987699 \\ R^2_{f_4(x)} &= 0.988418 \end{aligned} \tag{4.3}$$

Por tanto y según lo que se ha dicho anteriormente, la función $f_4(x)$ es la que se usará para calcular el óptimo y la *zona óptima*.

Para calcular el mínimo se usará la primera derivada de $f_4(x)$. Para calcular la *zona óptima*, se considerarán aquellos niveles cuyos tiempos no superen el milisegundo del tiempo óptimo. Para el ejemplo de "Motor", en la Figura 4.6 se muestra el óptimo y la zona óptima usando la función $f_4(x)$ que es la que más se ajusta a la media de tiempos.

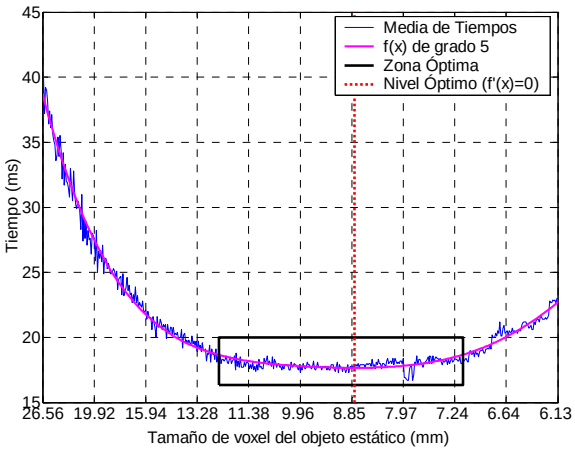


Figura 4.6: Óptimo y Zona Óptima experimental del modelo "Motor" con la herramienta "Mano-h0".

En la Tabla 4.2 y en la Tabla 4.3 se ofrecen todos los experimentos realizados combinando los modelos estáticos con la herramienta "Mano-h0". Se han realizado los experimentos usando tanto la geometría del cubo para formar los voxels como la paralelepípeda. Estos valores servirán para validar posteriormente la solución analítica que se propone en este capítulo.

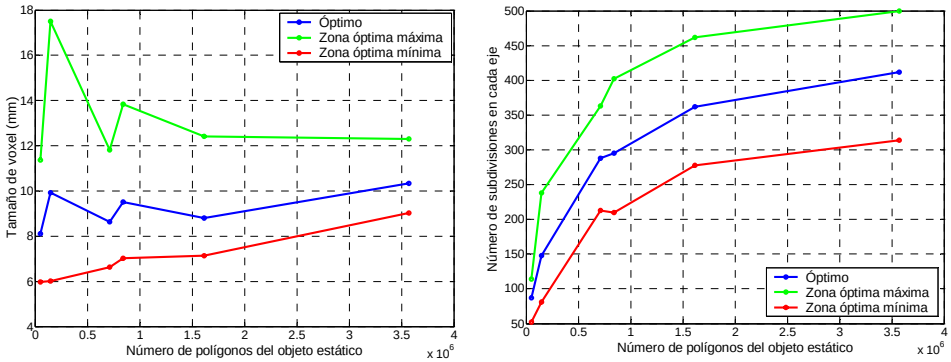
Modelo estático		**Carcasa**	**Diferencial**	**Turbina**	**Externals**	**Motor**	**Tay**
Óptimo δ_{opt}		8.11	9.93	8.62	9.52	8.8	10.34
Zona Óptima	**Mínimo** $\delta_{\min}$	5.98	6.01	6.63	7.03	7.14	9.04
	Máximo $\delta_{\max}$	11.37	17.5	11.82	13.84	12.41	12.3

Tabla 4.2: Óptimos y zonas óptimas para los diferentes modelos estáticos con la herramienta "Mano-h0" y usando geometría cúbica para los voxels.

Modelo estático		Carcasa	Diferencial	Turbina	Externals	Motor	Tay
Óptimo N_{opt}		87	148	288	295	362	412
Zona Óptima	**Mínimo** N_{min}	52	81	213	210	278	314
	Máximo N_{max}	114	238	363	402	462	500

Tabla 4.3: Óptimos y zonas óptimas para los diferentes modelos estáticos con la herramienta "Mano-h0" y usando geometría paralelepípeda para los voxels.

Estos valores se pueden apreciar mejor si se representan en sendas gráficas (Figura 4.7).



(a) (b)

Figura 4.7: Óptimos y zonas óptimas para las parejas de objetos estático-"Mano-h0" usando geometría cúbica (a) o paralelepípeda (b) para los voxels.

Las demás combinaciones de los modelos estáticos con el resto de herramientas también se han llevado a cabo pero no se muestran los resultados debido al gran conjunto que forman. Cuando se valide la fórmula analítica al final del capítulo entonces se verá el comportamiento que tiene el método con todo el conjunto de pruebas.

Además de los ejemplos prácticos expuestos anteriormente, también se han realizado experimentos con modelos sencillos y teóricos. La primitiva usada para representar tanto al objeto estático como al móvil ha sido una esfera de diámetro 500 mm. Ambos objetos se penetran 75 mm. Aunque estos ejemplos nunca se van a dar ya que en la realidad el objeto estático es mucho mayor que el móvil, pueden ofrecer información orientativa acerca del comportamiento del método.

En la primera de las tablas (Tabla 4.4) se dan los óptimos y zonas óptimas encontradas al enfrentar dos esferas del mismo tamaño en cada caso. Por el

contrario, en la Tabla 4.5 se dan los mismos resultados pero enfrentando una esfera móvil de triangularización fija (960 triángulos) a diferentes representaciones de esferas estáticas. Al ser modelos esféricos cuyos volúmenes contenedores son cúbicos, únicamente se ha tenido en cuenta el método de voxels cúbicos para estas pruebas. Para la elección del número de polígonos de las esferas, se elige coger uno o dos ejemplos en cada orden de magnitud.

		Polígonos de cada esfera (500 mm)							
		224	**960**	**3024**	**3968**	**16128**	**20136**	**65024**	**198468**
Óptimo δ_{opt}		26.57	20.2	16.83	16.83	9.71	9.53	9.35	9.18
Zona Óptima	**Mínimo** $\delta_{\min}$	10.98	11.74	11.48	12.02	7.77	7.77	8.71	8.86
	Máximo $\delta_{\max}$	100.98	42.08	28.05	22.95	12.02	11.48	9.9	9.35

Tabla 4.4: Óptimos y zonas óptimas para parejas de esferas del mismo tamaño (mm).

		Polígonos de la esfera estática (500 mm)							
		224	**960**	**3024**	**3968**	**16128**	**20136**	**65024**	**198468**
Óptimo δ_{opt}		21.04	20.2	17.41	16.83	15.78	14.03	9.9	9.53
Zona Óptima	**Mínimo** $\delta_{\min}$	10.3	11.74	10.98	10.74	9.9	9.35	7.43	7.78
	Máximo $\delta_{\max}$	126.23	42.08	28.05	28.05	24.04	21.04	18.03	10.98

Tabla 4.5: Óptimos y zonas óptimas (mm) para parejas de esferas (esfera móvil de 960 polígonos).

Los valores representados gráficamente aparecen en la Figura 4.8.

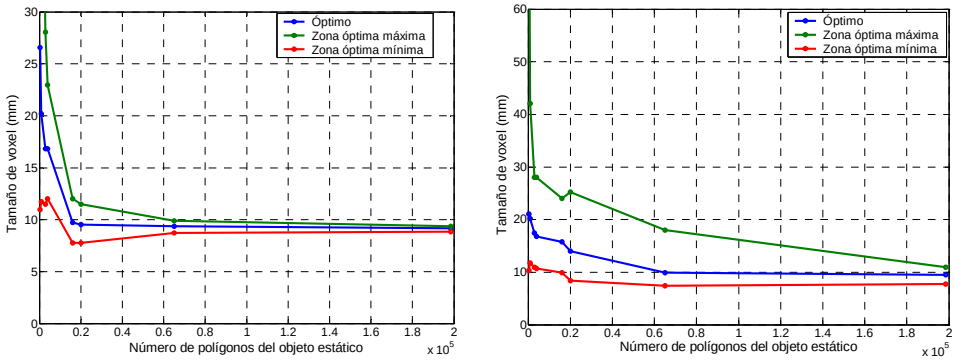


Figura 4.8: Óptimos y zonas óptimas representadas gráficamente para parejas de esferas con idéntico tamaño (a) y con una de las esferas de tamaño fijo (b).

Estas gráficas servirán posteriormente en la validación de la fórmula para observar por dónde se mueve el óptimo calculado analíticamente. En el siguiente apartado se describen los pasos seguidos para llegar a la fórmula final que se utiliza para calcular el voxelizado óptimo de una escena. Posteriormente, en otro apartado, se validará ese óptimo analítico con todos los resultados que acaban de ser mostrados a lo largo de este apartado.

4.4 SOLUCIÓN ANALÍTICA

En este apartado se describe la solución analítica o heurística que se ha hallado para solucionar el problema de la búsqueda de la voxelización óptima.

Ya que la tarea no es nada trivial por su dependencia de múltiples variables, primero se enumeran aquellos factores que a priori se piensa que pueden influir en esa solución analítica, razonando el por qué de esa influencia, para después ir eliminando factores hasta llegar a la solución escogida.

Considerando que en la escena sólo hay dos objetos, estático conteniendo toda la maqueta virtual y móvil que representa a la herramienta, los factores que pueden estar implicados en la fórmula analítica son:

- El número de polígonos del objeto estático y del móvil.
- El área media de los polígonos de ambos objetos.
- La longitud media de las aristas de los polígonos de los dos objetos.

- El volumen contenedor que encierra a cada uno de los objetos.

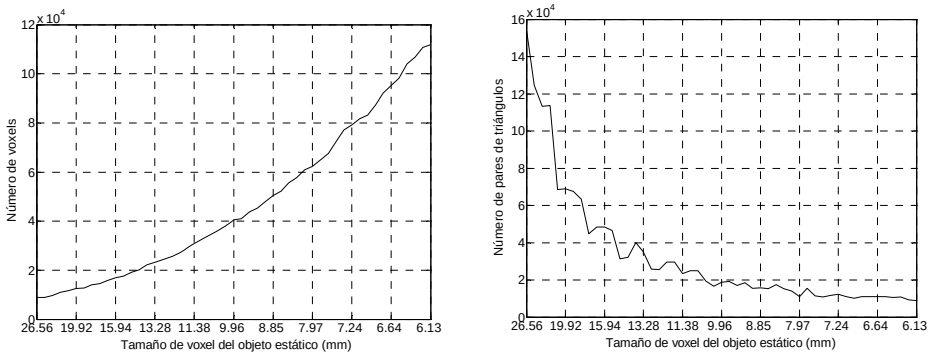
Antes de llegar a la solución propuesta, se tuvieron en cuenta varias opciones que se descartaron una vez realizados los experimentos del capítulo anterior y estudiado sus resultados.

Para empezar se podría pensar en utilizar el objeto estático como único objeto a considerar en la fórmula ya que la voxelización se va a realizar sobre ese modelo. Sin embargo no sería correcto ya que como se ha podido comprobar con la solución experimental, para un mismo modelo estático el voxelizado óptimo cambia dependiendo de la herramienta elegida en cada momento.

Tampoco puede depender única y exclusivamente del objeto móvil ya que de los resultados experimentales se puede concluir que si bien el modelo de la herramienta afecta al óptimo, la elección del modelo estático tiene mayor peso en el óptimo encontrado.

Entonces, y como se mencionaba al principio de este apartado, la solución final dependerá de la geometría de ambos objetos, tanto estático como móvil. Como a priori no se puede descartar ninguno de los factores expuestos, se ha modelado el comportamiento del método con varias fórmulas usando tanto el área como las aristas de los objetos. En el apartado 4.5 en el que se validan las fórmulas, se observarán cuales de ellas se comportan mejor.

Tal y como funciona el método propuesto y descrito en el capítulo 3, existen dos niveles de precisión: voxels y facetas. En las siguientes figuras se representa el número de primitivas implicadas en cada nivel. Mientras que en la Figura 4.9a se muestra el conjunto de voxels que se tiene en cuenta en cada nivel de voxelizado (incluidos los vacíos), en la Figura 4.9b aparece el número de parejas de triángulos que se consideran en la *narrow phase*. En este último caso se representa el número total de parejas de triángulos antes de aplicar los distintos filtros descritos en el apartado 3.2.2.3.



(a) (b)

Figura 4.9: Número de voxels implicados (a) y número de pares de triángulos testeados (b) en cada nivel de voxelizado elegido.

En la Figura 4.6 se podía apreciar que a partir de un grano de celda próximo a 8.85 mm los tiempos vuelven a ser peores. Sin embargo, el número de testeos triángulo-triángulo realizados a partir de ese nivel sigue disminuyendo. Esto indica que llega un punto (nivel óptimo de voxelizado) a partir del cual el mayor porcentaje de tiempo se lo lleva otro tipo de cómputo en el algoritmo. La respuesta está en la etapa de los voxels. Esta etapa es previa a los testeos de facetas y comprende el conjunto de voxels del objeto estático que se va a mirar. A partir del tamaño 8.85 mm el número de voxels a testear es tan elevado, que aunque la mayoría de ellos esté vacío, el tiempo de su comprobación empieza a ser el cuello de botella del algoritmo.

Este subconjunto de voxels depende del objeto móvil. Depende del número de polígonos de la herramienta así como también del área de esos polígonos. Contra mayor sea el área de los triángulos del objeto móvil, mayor será el conjunto de voxels que intercepte con ese triángulo.

Por lo tanto, parece que el nivel de voxel o *broad phase* depende de la geometría del objeto móvil pero el nivel de triángulo o *narrow phase* se ve afectado más por la geometría del objeto estático. En el siguiente apartado se describe el método hallado que relaciona todos los parámetros implicados en la solución finalmente adoptada.

4.4.1 La función de coste del algoritmo

Se llamará *función de coste* del algoritmo a la expresión matemática que describe el comportamiento del mismo, es decir, que permite predecir el tiempo que consumirá el algoritmo para solucionar un problema de colisión en un escenario determinado. En este apartado se irá explicando cómo se llega a la

expresión final de la función. Si esta función se construye dependiente de la variable que interese, en este caso el tamaño del voxel, basta con encontrar el valor de esa variable que haga mínima la función y se habrá encontrado el óptimo de la función de coste.

Si se quiere definir el comportamiento del algoritmo de colisiones, para ello hay que fijarse en la descripción que se da en el capítulo anterior. En él se exponen los dos niveles de precisión que posee el algoritmo: nivel de voxel y nivel de triángulo. Cada triángulo del objeto móvil pasa por estos dos niveles. Por tanto, si n_m es el número de triángulos del objeto movil, v_o es el número de voxels objetivo por cada triángulo móvil y f_e es el número de facetas por voxel ocupado del objeto estático, la función de coste z que describe el comportamiento del algoritmo puede expresarse por la ecuación (4.4).

$$z = n_m \cdot v_o \cdot f_e \tag{4.4}$$

La función de coste depende de los dos objetos; lógico por otra parte ya que el algoritmo también depende de los dos objetos. Sin embargo esta fórmula considera únicamente el caso peor, es decir, cuando todo el objeto móvil (y por tanto todas sus facetas) está en colisión con la superficie estática de contacto. En una simulación normal, este suceso no ocurre nunca ya que las condiciones normales serán de no colisión o de colisión con unos pocos triángulos implicados. Por eso, la función z se separa en dos factores, cada uno correspondiente a un nivel de precisión como queda reflejado en (4.5).

$$z = n_m \cdot p_{nc} \cdot v_o + n_m \cdot (1 - p_{nc}) \cdot v_o \cdot f_e \tag{4.5}$$

Donde p_{nc} es la probabilidad de que un triángulo del objeto móvil no esté en colisión con algún voxel con geometría. Según experimentos realizados con diferentes escenarios y herramientas, p_{nc} varía entre 0.95 y 0.99 cuando la herramienta se encuentra en posiciones normales de trabajo.

Sólo queda un detalle para ajustar la función al comportamiento del algoritmo. En la expresión (4.5), los parámetros v_o y f_e tienen el mismo peso computacional al contrario de lo que ocurre en la realidad donde el chequeo entre triángulos es significativamente más costoso que el tiempo consumido comprobando si un voxel está lleno o vacío. El ratio entre los dos tiempos de computación, t_f/t_v, se ha llamado r_t y según experimentación, el tiempo de chequeo entre triángulos es aproximadamente cien veces mayor que el de voxels. La ecuación (4.6) modela finalmente el comportamiento del algoritmo presentado en el capítulo anterior.

$$z = n_m \cdot p_{nc} \cdot v_o + n_m \cdot (1 - p_{nc}) \cdot v_o \cdot f_e \cdot r_t \tag{4.6}$$

Una vez expresado el comportamiento del algoritmo mediante la función de coste, a continuación hay que determinar una expresión de los parámetros v_o y f_e en función del tamaño del voxel (variable) y de la geometría de la escena (parámetros geométricos específicos).

4.4.2 Parámetros de la función de coste

En este apartado se obtiene una expresión de la función de coste que permite obtener un tamaño de voxel (δ) o nivel de subdivisión (N) que minimiza la función de coste.

Para una escena y voxelización específica, v_o y f_e pueden ser medidos directamente. Sin embargo, para sustituirlos en la expresión matemática de la función de coste, estos valores deben ser aproximados (predecidos, conjeturados) como función de una variable (δ o N) y de la geometría de la escena.

El número de voxels objetivo, v_o, es el número medio de voxels intersectados por un triángulo del objeto móvil. Realmente, el algoritmo lo que testea es la intersección entre el AABB del triángulo y la malla de voxels estáticos (descrito en el apartado 3.2.2.2). Antes de dar una expresión para v_o, hay que estimar las dimensiones de las cajas contenedoras de los triángulos.

En el caso de un triángulo equilátero, una forma de construir una caja que siempre contenga al triángulo sea cual sea la orientación del mismo es eligiendo aquella caja cúbica cuyo lado sea igual a la arista del triángulo. Un ejemplo en 2D se puede apreciar en la Figura 4.10.

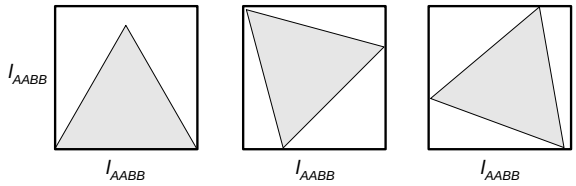


Figura 4.10: Caja contenedora mínima de un triángulo equilátero en cualquiera de sus posibles orientaciones.

En la solución adoptada, se parte de la arista media del triángulo móvil como valor para las aristas del AABB (l_{AABB}).

Una vez se tienen las dimensiones de una caja paralela a los ejes, se puede predecir, en el peor caso y en función del grano de celda, el número de voxels que puede llegar a intersectar esa caja (cubo o paralelepípedo).

$$v_o^c = \left(\frac{l_{AABB}}{\delta} + 2 \right)^3, \quad v_o^p = \left(\frac{3N \cdot l_{AABB}}{X+Y+Z} + 2 \right)^3 \tag{4.7}$$

Por otro lado, el número de triángulos por voxel (f_e) puede ser aproximado utilizando diferentes características geométricas del objeto estático: el área de los triángulos o sus aristas. En los trabajos experimentales de Gregory et al. (2000) se usaban aristas.

Para calcular f_e en función del área ($f_{e,area}$) se necesita estimar el área media de triángulo que puede contener un voxel (A_{vox}). Se estima ese valor según el área del triángulo mostrado en la Figura 4.11.

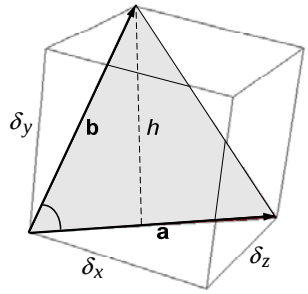


Figura 4.11: Voxel conteniendo al triángulo con mayor área posible.

Para calcular el área de ese triángulo se usará el producto vectorial entre los vectores **a** y **b** tal y como se muestra en (4.8).

$$A_{vox} = \frac{1}{2} \cdot |\mathbf{a} \times \mathbf{b}| \tag{4.8}$$

A esta área media contenida en un voxel, se le llamará de aquí en adelante A_{vox}. Calculando el producto vectorial entre los dos vectores definidos por los dos lados del triángulo quedaría:

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} \bar{i} & \bar{j} & \bar{k} \\ 0 & \delta_y & \delta_z \\ \delta_x & 0 & \delta_z \end{pmatrix} = \delta_y \delta_z \bar{i} + \delta_x \delta_z \bar{j} - \delta_x \delta_y \bar{k} \Rightarrow \tag{4.9}$$

$$\Rightarrow |\mathbf{a} \times \mathbf{b}| = \sqrt{(\delta_y \delta_z)^2 + (\delta_x \delta_z)^2 + (\delta_x \delta_y)^2}$$

Para los voxels cúbicos el grano de celda es idéntico en los tres ejes. En el caso paralelepípedo, la variable es el número de subdivisiones que se saca gracias a las dimensiones del volumen contenedor global (*X*, *Y*, *Z*). Sustituyendo estas afirmaciones en (4.9) y aplicando el resultado en (4.8), el área contenida en un voxel quedaría expresado por las fórmulas de (4.10).

$$A_{vox}^{c} = \frac{\delta^2\sqrt{3}}{2}, \quad A_{vox}^{p} = \frac{1}{2N^2}\sqrt{(XY)^2 + (XZ)^2 + (YZ)^2} \tag{4.10}$$

Finalmente, si A_e es el área media de los triángulos estáticos, la predicción, basada en el área, para el número de triángulos por voxel será el área media predecida contenida en un voxel partido por A_e. Esto mismo se expresa con las fórmulas (4.11).

$$f_{e,area}^{c} = \frac{A_{vox}^{c}}{A_e} = \frac{\delta^2\sqrt{3}}{2A_e}, \quad f_{e,area}^{p} = \frac{\sqrt{(XY)^2 + (XZ)^2 + (YZ)^2}}{2A_eN^2} \tag{4.11}$$

En lugar del área, se puede tomar como criterio de comparación las aristas de los triángulos. En este caso, la predicción se puede obtener comparando esas aristas con la diagonal del voxel ($f_{e,diag}$) o con su lado medio ($f_{e,lado}$) con lo que saldrían otras dos expresiones para cada geometría del voxel.

$$f_{e,diag}^{c} = \frac{diag_{vox}^{c}}{E_e} = \frac{\delta\sqrt{3}}{E_e}, \quad f_{e,diag}^{p} = \frac{\sqrt{X^2 + Y^2 + Z^2}}{E_eN}$$

$$f_{e,lado}^{c} = \frac{lado_{vox}^{c}}{E_e} = \frac{\delta}{E_e}, \quad f_{e,lado}^{p} = \frac{X + Y + Z}{3E_eN} \tag{4.12}$$

Recordando la función de coste vista en (4.6), v_o y f_e son predecidas con las fórmulas vistas en este apartado, n_m es un dato de entrada que se obtiene del modelo de herramienta usado. Además también se afirmaba que p_{nc} y r_t eran constantes que se mantenían en los experimentos típicos de mantenibilidad. Por todo esto, la función *z* sólo depende de una única variable: el grano o subdivisión; por lo que se puede buscar el valor óptimo que coincidirá con el mínimo de la función. A los valores que hacen mínima la función se les llamará δ_a y N_a respectivamente (óptimos analíticos).

Estos valores óptimos son obtenidos en la etapa de preproceso y el tiempo requerido para evaluar la función de coste no es significativo. Los valores se pueden buscar ya sea mediante un muestreo de un intervalo del

dominio o bien resolviendo, según sea el caso, una de las ecuaciones mostradas en (4.13).

$$\frac{dz_c}{d\delta} = 0, \quad \frac{dz_p}{dN} = 0 \tag{4.13}$$

La solución de estas ecuaciones tanto para el caso cúbico (z_c) como para el paralelepípedo (z_p) se presenta en el Anexo A.

4.5 VALIDACIÓN DE LA SOLUCIÓN ANALÍTICA

Una vez obtenidos los resultados de la función de coste para todas las parejas de modelos, y teniendo también los resultados experimentales del apartado 4.3, se puede comparar la solución analítica con la experimental para validar la primera.

Una forma de contrastar los resultados es definiendo un ***error relativo de tamaño de voxel***, ε_δ. La fórmula (4.14) se escribe para el caso de voxels cúbicos pero en el caso de paralelepípedos sería idéntica cambiando los tamaños de voxel δ por el número de subdivisiones *N*.

$$\varepsilon_\delta = \frac{\left|\delta_{opt} - \delta_a\right| \cdot \left(\left\lfloor \frac{\delta_{opt}}{\delta_a} \right\rfloor + \left\lfloor \frac{\delta_a}{\delta_{opt}} \right\rfloor \right)}{\left(\delta_{opt} - \delta_{min}\right) \cdot \left\lfloor \frac{\delta_{opt}}{\delta_a} \right\rfloor + \left(\delta_{max} - \delta_{opt}\right) \cdot \left\lfloor \frac{\delta_a}{\delta_{opt}} \right\rfloor} \tag{4.14}$$

Si este error es cero, entonces δ_a coincide exactamente con el tamaño de voxel óptimo encontrado experimentalmente (al que se llamaba δ_{opt}). Si $0 < \varepsilon_\delta \leq 1$, δ_a no es el óptimo pero se encontrará dentro de la zona óptima.

Asimismo, también se define un ***error absoluto*** ε_a. Este error indicará cuántos milisegundos se aleja la predicción del óptimo real conseguido con la experimentación. Este error sólo se calculará en los ejemplos en los que la aproximación calculada no entra dentro de la zona óptima.

Primero se analizan las pruebas realizadas con las primitivas esféricas para pasar luego a los casos prácticos. En la Tabla 4.6 aparece el error relativo correspondiente a cada prueba esfera contra esfera.

	Polígonos de cada esfera							
Método	224	960	3024	3968	16128	20136	65024	198468
Área	0.86	0.63	0.5	0.25	0.38	0.92	11.5	22.5
Diagonal	1.14	1.00	1.5	2.0	2.3	2.22	1.5	22.5
Lado medio	1.14	1.05	1.67	2.25	2.8	2.78	1.00	21.0

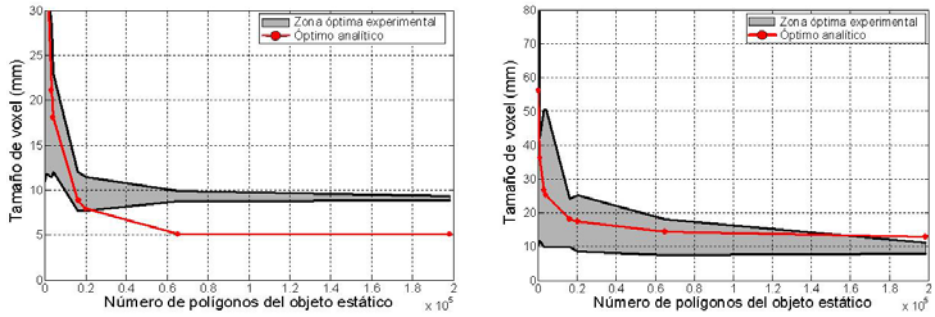
Tabla 4.6: Valores de ε_δ para los pares de objetos esfera-esfera.

Las celdas en gris se refieren a los casos en los que no se ha logrado aproximar correctamente. El método que parece aproximar mejor es el de las áreas. Sin embargo, en los dos últimos casos, los que enfrentan dos esferas de 60000 y 200000 polígonos, también falla este método siendo su ε_a 10 ms y 74 ms en cada caso. Estas escenas son casos improbables ya que no tiene sentido tener una herramienta de 200000 polígonos, tantos como podría tener algún escenario estático. Por eso, también se realizaron pruebas manteniendo una esfera con un tamaño fijo de 960 triángulos (un nivel de detalle razonable para el objeto móvil). En la Tabla 4.7 aparecen los resultados de la aproximación.

	Polígonos de la esfera estática							
Método	224	960	3024	3968	16128	20136	65024	198468
Área	0.75	0.85	0.91	0.83	0.36	0.58	0.7	2.0
Diagonal	0.75	1.15	1.73	1.67	2.0	2.17	1.74	6.0
Lado medio	0.8	1.23	1.73	1.67	2.0	2.17	1.78	6.0

Tabla 4.7: Valores de ε_δ para los pares de objetos esfera-esfera con una de ellas de tamaño fijo a 960 polígonos.

Con estos ejemplos el método se comporta mejor pero falla en el último caso. El error absoluto en esta ocasión es de 3.1 ms. Este valor quiere decir que aunque se ha fallado en la aproximación al óptimo, el valor calculado dista únicamente en unos pocos milisegundos de la solución óptima. En la Figura 4.12 se puede ver en gris la zona óptima experimental y por dónde pasan los óptimos calculados analíticamente en los dos tipos de experimentos. En ambas figuras se tiene en cuenta únicamente el método de las áreas que es el que mejor se comporta.



(a) (b)

Figura 4.12: Comparación del método con la zona experimental en el caso de esferas del mismo tamaño (a) y con una de las esferas fijada con un tamaño de 960 polígonos (b).

Una vez vistos los ejemplos teóricos, en las siguientes tablas (Tabla 4.8 y Tabla 4.9) se dan los resultados del método aplicándolo a los ejemplos prácticos. El error relativo muestra el comportamiento de la función de coste para cada par de objetos estático-móvil. Cada tabla refleja uno de los dos tipos diferentes implementados de geometría de voxel.

		Modelo de herramienta				
Método	Modelo estático	Torque	Scanner	Wrench	Square	Mano-h0
Área	CarcasaTornillos	0.60	0.00	0.40	0.25	0.41
	Diferencial	0.36	0.58	0.27	0.46	0.49
	Turbina	0.81	0.92	0.97	0.73	0.73
	Externals	0.70	0.60	0.67	0.58	0.43
	Motor	0.77	0.31	0.36	0.67	0.64
	Tay	0.32	0.07	0.85	0.10	0.07
Diagonal	CarcasaTornillos	1.47	1.53	0.04	1.08	1.93
	Diferencial	1.16	1.16	0.87	1.00	1.31
	Turbina	1.39	1.38	1.75	1.45	2.04
	Externals	1.22	1.18	1.42	1.31	1.66
	Motor	1.56	1.28	1.14	1.65	1.88
	Tay	1.80	1.28	0.16	1.12	2.78
Lado medio	CarcasaTornillos	1.55	1.83	0.71	1.17	2.14
	Diferencial	1.23	1.33	1.15	1.10	1.48
	Turbina	1.39	1.50	2.16	1.45	2.15
	Externals	1.22	1.35	1.80	1.31	1.87
	Motor	1.56	1.55	1.54	1.79	2.12
	Tay	1.97	1.66	1.10	1.27	3.29

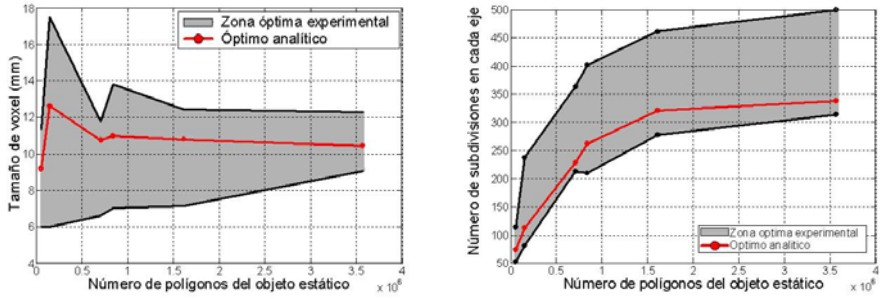
Tabla 4.8: Valores de ε_δ para los pares de objetos estático-móvil usando geometría cúbica en los voxels.

		Modelo de herramienta				
Método	Modelo estático	Torque	Scanner	Wrench	Square	Mano-h0
Área	CarcasaTornillos	0.63	0.21	0.25	0.46	0.37
	Diferencial	0.81	0.62	0.36	0.74	0.52
	Turbina	0.77	0.90	0.29	0.68	0.79
	Externals	0.73	0.53	0.44	0.76	0.39
	Motor	0.78	0.49	0.13	0.76	0.49
	Tay	0.69	0.21	0.80	0.17	0.76
Diagonal	CarcasaTornillos	1.24	1.10	0.44	1.11	1.40
	Diferencial	1.43	1.12	0.83	1.44	1.27
	Turbina	1.08	1.39	0.81	1.10	1.84
	Externals	1.12	1.23	0.98	1.39	1.71
	Motor	1.47	1.24	1.03	1.59	2.19
	Tay	1.68	1.21	0.46	1.51	2.18
Lado medio	CarcasaTornillos	1.31	1.31	0.76	1.18	1.57
	Diferencial	1.43	1.27	1.08	1.50	1.43
	Turbina	1.08	1.49	1.11	1.10	1.84
	Externals	1.12	1.43	1.27	1.39	1.71
	Motor	1.47	1.46	1.50	1.59	2.51
	Tay	1.68	1.56	1.00	1.65	2.52

Tabla 4.9: Valores de ε_N para los pares de objetos estático-móvil usando geometría paralelepípeda en los voxels.

Gracias a estos resultados se puede comprobar que la solución analítica basada en las áreas logra aproximar correctamente al óptimo en todos los ejemplos probados. Los métodos basados en las aristas (método diagonal y método del lado medio) no tienen un buen comportamiento por lo que a partir de ahora se rechazarán.

Tanto si se usa la geometría cúbica como la paralelepípeda, los resultados son similares en cuanto a que los dos métodos hayan tamaños de voxel que caen dentro de la zona óptima experimental. Como en los ejemplos esféricos, en las siguientes figuras se muestra la zona gris correspondiente a la zona óptima y una línea roja que representa el recorrido que hace el óptimo analítico dependiendo del tamaño del modelo. Para este ejemplo, la Figura 4.13 sólo tiene en cuenta los resultados de las escenas con un único modelo móvil: "Mano-h0".



(a) (b)

Figura 4.13: Comparación del método con la zona experimental en el caso de voxels cúbicos (a) y paralelepípedos (b).

Gracias a estos resultados se prueba la validez de la función de coste para la aproximación a un tamaño óptimo de voxel.

A continuación y una vez comprobado la validez de la función de coste, se exponen una serie de casos extremos o *worst-cases* para observar cómo se comportaría la solución analítica propuesta en estos sucesos. Únicamente se tendrá en cuenta la solución basada en las áreas de los triángulos dado su probado buen funcionamiento. Asimismo, las pruebas realizadas usan la geometría cúbica de los voxels.

El primero de ellos es el modelo "MotorCarcasa". Este modelo es idéntico al modelo "Motor" usado anteriormente con la diferencia, como se puede apreciar en la Figura 4.14, que este nuevo ejemplo posee un par de carcasas recubriendo todo el motor y un par de anclajes que hacen que el volumen de la caja contenedora del objeto se incremente en un 310% y por tanto se desaproveche más espacio por culpa de la estructura de voxels. Se ha querido usar este modelo menos denso para estudiar el impacto que tienen modelos no tan compactos en la predicción del óptimo.

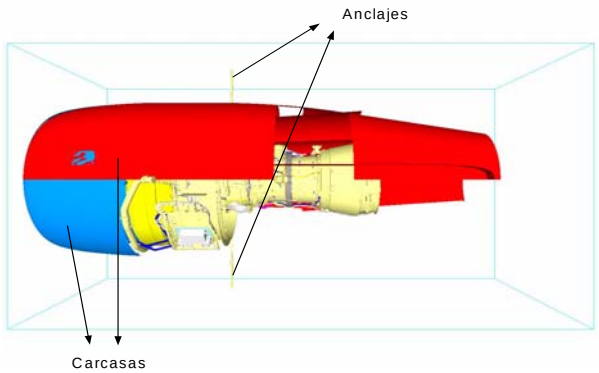


Figura 4.14: Representación 3D del modelo "MotorCarcasa".

En este ejemplo es tan grande el volumen considerado que el número de voxels que se necesitan para ajustar al tamaño óptimo del voxel por triángulo (f_e) es mayor, por lo que hay un doble desaprovechamiento de memoria: uno por los voxels vacíos y otro por el ajuste al tamaño del triángulo. Este desaprovechamiento se traduce en que el tipo de curva de tiempos que sale en este caso tiene forma asintótica y no parabólica. Para solucionarlo es indispensable usar las técnicas de ahorro de memoria vistas en el capítulo anterior (técnicas *hashing*).

En la Tabla 4.10 se resumen las zonas óptimas encontradas con el modelo "MotorCarcasa" y diferentes modelos de herramientas, así como el óptimo encontrado analíticamente. El óptimo experimental también es mostrado pero es un valor simbólico. Realmente lo que interesa es si el óptimo analítico se encuentra o no dentro de la zona óptima.

Modelo de herramienta	Óptimo δ_{opt}	Zona óptima $[\delta_{\min}, \delta_{\max}]$	Óptimo analítico δ_a	Error relativo ε_δ
Torque	11.06	[10.29, 18.71]	15.80	0.73
Scanner	12.84	[10.29, 16.80]	11.12	0.62
Wrench	11.24	[8.95, 13.72]	9.00	0.97
Square	13.52	[10.29, 17.89]	14.17	0.19
Mano-h0	17.04	[11.43, 19.14]	12.90	0.65

Tabla 4.10: Óptimos y zonas óptimas encontradas experimentalmente y analíticamente con el modelo "MotorCarcasa" (mm).

Se puede apreciar que aún en este caso extremo, la función de coste logra aproximarse de forma correcta al óptimo en todos los casos.

Un segundo caso extremo que se presenta es el análisis de diferentes niveles de detalle de un mismo modelo 3D (se seguirá usando el modelo

"Motor" para poder contrastar los ejemplos). A diferencia del anterior caso extremo en el que se variaba el volumen contenedor, en este ejemplo se juega con las áreas de los polígonos.

Evidentemente, si se simplifica un modelo disminuyendo con ello el número de polígonos, la fórmula dará un grano de celda mayor ya que el A_e y E_e de las fórmulas (4.11) y (4.12) aumentan. En la Figura 4.15 se muestran las curvas de tiempos para distintos niveles de detalle de un mismo modelo (en este caso "Motor").

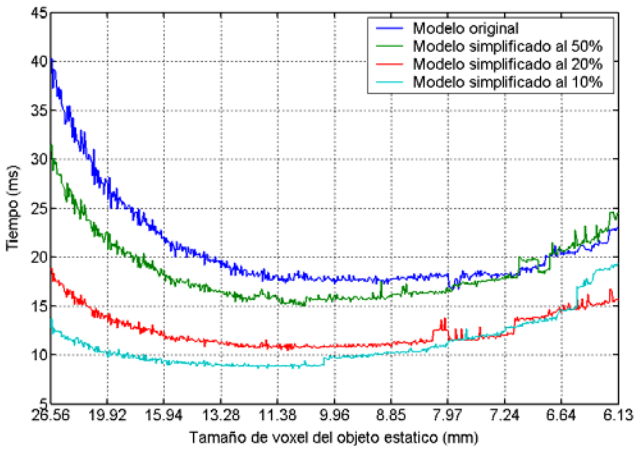


Figura 4.15: Tiempos medidos para un rango de niveles de voxelización y diferentes niveles de detalle del modelo "Motor".

Si se estudia la gráfica de tiempos, se puede observar un resultado muy importante de cara a la fórmula analítica. Los tiempos son menores según se considere el modelo más o menos simplificado. Esto es razonable ya que el modelo simplificado al 50% tiene la mitad de polígonos y por tanto debería dar tiempos más bajos que el modelo original al tener menos pares de triángulos a testear. A simple vista parece que las zonas óptimas coinciden. Sin embargo esto no ocurre así y el grano de celda va siendo mayor contra más simplificado esté el modelo. Esto se puede apreciar mejor en la Figura 4.16 donde se muestran las cuatro gráficas del óptimo experimental encontrado.

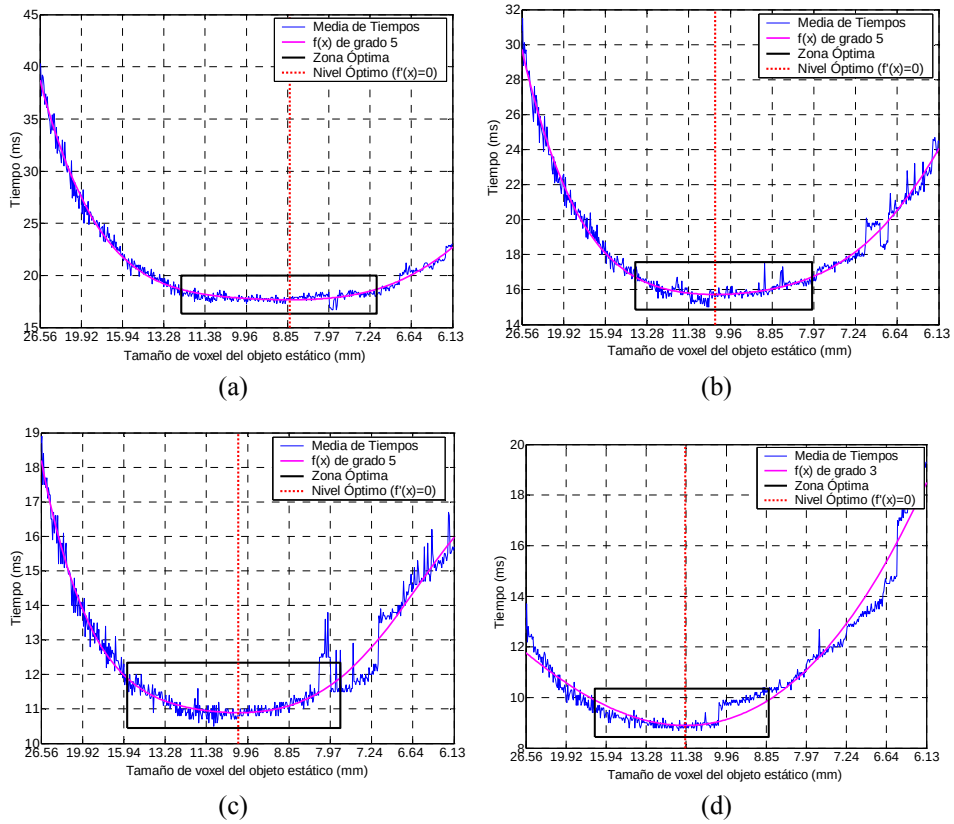


Figura 4.16: Óptimo y Zona Óptima experimental para el modelo "Motor" original (a), y con simplificaciones al 50% (b), 20% (c) y 10% (d) respectivamente.

En la Tabla 4.11 se resumen los óptimos y zonas óptimas de cada ejemplo. Asimismo, se muestra el error cometido entre la estimación analítica y el resultado experimental en los casos en los que no se logra aproximar correctamente.

Nivel de detalle (% del original)	Óptimo δ_{opt}	Zona óptima $[\delta_{\min}, \delta_{\max}]$	Error relativo ε_{δ}	Error absoluto ε_a (ms)
100%	8.79	[7.14, 12.41]	0.64	-
50%	10.46	[8.00, 13.93]	0.86	-
20%	10.24	[7.78, 15.69]	1.21	1.8
10%	11.42	[8.79, 16.88]	1.45	2

Tabla 4.11: Óptimos y zonas óptimas (mm) encontradas experimentalmente con el modelo "Motor" con diferentes niveles de detalle.

Se comprueba que en este caso el heurístico vuelve a funcionar correctamente salvo con los modelos que más se han simplificado. El error obtenido no está muy alejado del óptimo experimental (1.8 y 2 ms respectivamente) pero no se ha podido estimar bien un valor de voxelización. La razón es que son casos tan extremos que la fórmula no tolera bien el tamaño del área media de los polígonos que poseen estos modelos. Hay que tener en cuenta que el área media en el modelo original es de 89.2 mm^2 mientras que el modelo con una décima parte de los triángulos tiene un área media de 847.58 mm^2 (y 443.12 mm^2 en el caso de la simplificación al 20%). Estas áreas son tan grandes que tienen un gran peso al introducirlas en la función de coste. Una posible mejora sería el investigar este suceso e intentar modificar la función de coste para que contemple estos casos tan extremos. Sin embargo, para las simulaciones de mantenimiento y las precisiones con las que se trabaja en el mundo CAD de la aeronáutica, estos modelos tan simplificados tienen un nivel de detalle tan grueso que no son viables para este tipo de aplicaciones. Además, al ser modelos tan sencillos, se procesan con tal rapidez que la optimización pierde importancia.

4.6 ÚLTIMAS CONSIDERACIONES

Un problema que se podría presentar es que, en una misma simulación de mantenimiento, se cambie una herramienta por otra, lo cual modifica la función de coste *z*. Para solucionarlo se pueden tomar dos tipos de decisiones: la primera voxelizar el objeto estático cada vez que se cargue una nueva herramienta. Este proceso puede ser costoso (para ver los tiempos de cálculo de un voxelizado mirar el apartado 3.2.1.3) y contraproducente en cuanto a la interactividad de la simulación. La otra alternativa más atractiva es diseñar todas las herramientas con el mismo nivel de detalle. Esto no es nada fuera de lo común y ya que todas las herramientas tienen un tamaño parecido, sus aristas serán también parecidas (parámetro dependiente en la función de coste según la ecuación (4.7)).

Ligado a lo anterior, nótese que los resultados varían unos de otros dependiendo del nivel de detalle del objeto móvil. Esto se ha mostrado así para verificar la validez de la fórmula obtenida en este capítulo. Sin embargo y como se ha hecho constar anteriormente, en la práctica todas las herramientas utilizadas en una misma simulación virtual se habrían modelado con un nivel de detalle similar por lo que no habrá peligro en necesitar diferentes voxelizados del objeto estático.

En este capítulo se ha presentado una posible solución general y analítica al problema de la búsqueda del tamaño óptimo de voxel. Tanto en los casos teóricos, como en los prácticos y en los *worst-cases*, la fórmula heurística logra

calcular un valor que entra dentro de lo que se ha definido experimentalmente como zona óptima (1 ms de error respecto al óptimo medido experimentalmente). Además, el cálculo del valor óptimo no requiere gran coste computacional al poder ser evaluado como una expresión dependiente de parámetros geométricos de la escena (la expresión final queda reflejada en el Anexo A).

4.7 REFERENCIAS

García-Alonso, A., Serrano, N. y Flaquer, J., "Solving the Collision Detection Problem", *IEEE Computer Graphics and Applications*, Vol. 13 (3), pp. 36-43. Mayo 1994.

Gregory, A., Lin, M.C., Gottschalk, S. y Taylor, R., "Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback", *Computational Geometry – Theory & Applications* 15, pp. 269-285, 2000.

Held, M., Klosowski, J.T. y Mitchell, J.S.B., "Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs", *Proceedings of the Seventh Canadian Conference on Computer Geometry*, Vol. 3, pp. 205-210, Québec City, Québec, Canada. Agosto 1995.

McNeely, W.A., Puterbaugh, K.D. y Troy, J.J., "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", *Proceedings of the ACM SIGGRAPH'99 - Computer Graphics*, pp. 401-408. Los Angeles, California, USA. Agosto 1999.

Zhang, D. y Yuen, M.M.F., "A Coherence-based Collision Detection Method for Dressed Human Simulation", *Computer Graphics Forum*, Vol. 21, No. 1, pp. 33-42. 2002.

CAPÍTULO 5

CÁLCULO DE LA RESPUESTA DE COLISIÓN

5.1 INTRODUCCIÓN

En aplicaciones con restitución de esfuerzos, el módulo de Colisiones tiene que realizar un trabajo computacional adicional a la "simple" detección de intersecciones en la escena. Por ejemplo, en simulaciones dinámicas es deseable que los objetos se comporten físicamente como si estuvieran en el mundo real, o también en simulaciones de mantenibilidad se requiere una respuesta, ofrecida como sensación de contacto, por parte del sistema. Este cálculo adicional se denomina *Respuesta de Colisión*.

Al igual que antes, el módulo de Colisiones tiene que ser capaz de detectar las colisiones entre objetos con un grado de interactividad razonable. El coste computacional que se añade y que está asociado al cálculo de una respuesta de colisión no debe influir negativamente en el *frame rate* de colisiones.

En el caso de la mantenibilidad, la respuesta viene dada por una sensación de contacto necesaria para que el operario tenga la más cercana impresión de estar trabajando como sucedería en la realidad. El hardware encargado de ofrecer esa restitución de esfuerzos se denomina dispositivo háptico.

Ya que la restitución de esfuerzos mediante hápticos es una tarea propia tanto del software de las colisiones como de teoría de control, la descripción del

método, en el apartado 5.4, está dividida en dos partes, cada una describiendo la tarea que realiza cada módulo implicado. Previamente a la descripción del método, en los apartados 5.2 y 5.3 se ofrecen dos secciones ilustrativas: la primera describe en líneas generales un dispositivo háptico. La segunda es un pequeño estudio del arte enfocado al problema del cálculo de la penetración entre dos objetos.

El objetivo de esta investigación ha sido lograr unos algoritmos que consigan, con un mínimo de coste computacional, una información mediante la cual el sistema de control del háptico pueda producir una restitución de fuerza que siendo fiel al escenario también produzca una buena percepción (*feedback* táctil).

5.2 DISPOSITIVOS HÁPTICOS

5.2.1 Introducción previa

Los dispositivos hápticos entran dentro del grupo de las interfaces de usuario. Dentro de este grupo están aquellos dispositivos de interacción persona-computador basados en una comunicación bidireccional (canal entrada/salida) con el sistema. Las entradas vienen definidas por acciones del usuario mientras que el ordenador es el encargado de producir respuestas adecuadas (a veces a través de estas mismas interfaces) a dichas entradas.

El canal de entrada contiene la interfaz que utiliza el usuario para comunicar información e interactuar con el entorno virtual. Algunos ejemplos son los teclados, ratones, ratones 3D, guantes o incluso el reconocimiento de voz.

El canal de salida incluye la interfaz utilizada por el usuario para recibir información del entorno virtual. Algunas de las interfaces más importantes son la salida gráfica al monitor, la salida de audio y la reflexión de fuerza; pero por ejemplo, también puede ser un canal de salida un generador de olores.

Dentro de la reflexión de esfuerzos se encuentran los sistemas de reflexión táctil y de fuerza y más concretamente los dispositivos hápticos. Estas interfaces ofrecen un feedback táctil ya sea en entornos virtuales o en reales como es el caso de la teleoperación.

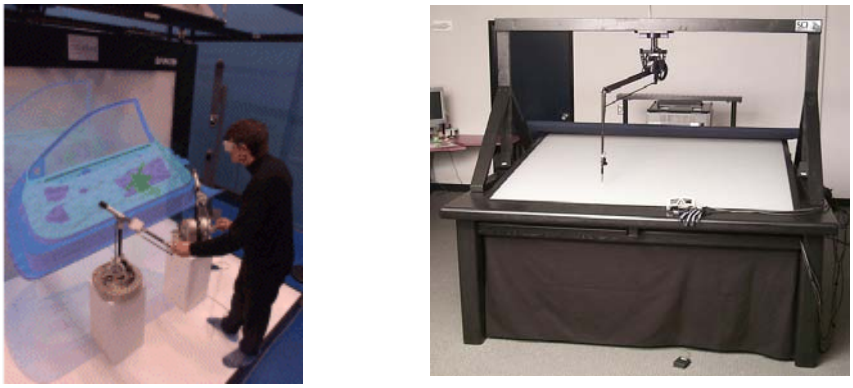
5.2.2 Descripción general

Los dispositivos hápticos proporcionan tanto una entrada al sistema virtual como una salida al usuario. Las entradas del sistema háptico, además de retornar

cambios visuales en el entorno virtual, también suministran salidas formadas por acciones de los motores. Estas respuestas producen una fuerza y un desplazamiento de tal forma que se pueden llegar a sentir contactos, pesos o rugosidades.

Dependiendo del dispositivo y del tipo de aplicación, los hápticos se pueden clasificar de diferentes maneras. Por ejemplo, una posible clasificación depende de dónde esté ubicado el dispositivo: éste puede ser de sobremesa o estar anclado en algún sitio fijo, o por el contrario puede adaptarse al cuerpo del usuario como un exoesqueleto.

Los mejores resultados se han conseguido con los primeros, de los que existen diferentes configuraciones según factores como la forma de trabajo, visibilidad, etc. En la Figura 5.1a se muestra a un usuario manejando dos hápticos anclados al suelo y en la Figura 5.1b aparece un sistema virtual desarrollado por Brederson et al. (2000) en el que el háptico está situado por encima de la cabeza del usuario y es combinado con varios dispositivos hardware más.



(a) (b)

Figura 5.1: Sistema de realidad virtual con restitución de fuerzas de Haption (a) y del Instituto de la Imagen y Computación Científica de la Universidad de Utah (b).

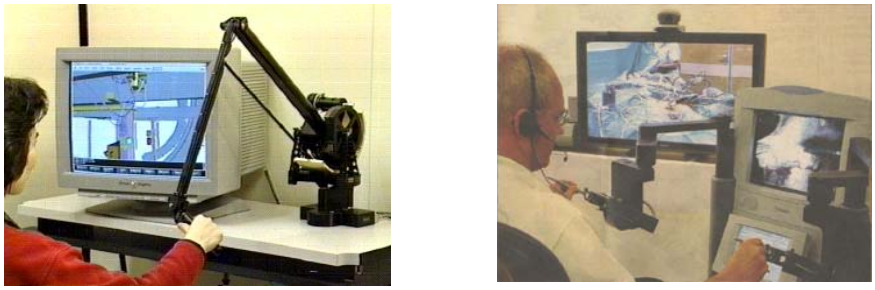
La ventaja de los exoesqueletos (Figura 5.2 y Figura 5.3) es que la cinemática y el espacio de trabajo coinciden con el de un humano. Sin embargo, el diseño resulta muy complejo y además el usuario puede acabar fatigándose por el peso del dispositivo. De todas maneras, a veces la manipulación de diseños CAD es más intuitiva con un dispositivo que asemeja una mano humana que no con una herramienta, como muestra la aplicación virtual del Departamento de Ciencias de la Computación de la Universidad de Utah en la que utilizan un brazo de la compañía Sarcos (Hollerbach et al. 1997).

Mundo virtual Pieza de ensamblaje Mundo real

Figura 5.2: Brazo háptico de la empresa Sarcos para la evaluación de modelos CAD de ensamblajes mecánicos.

Figura 5.3: Exoesqueleto ligero *CyberGrasp* de Immersion.

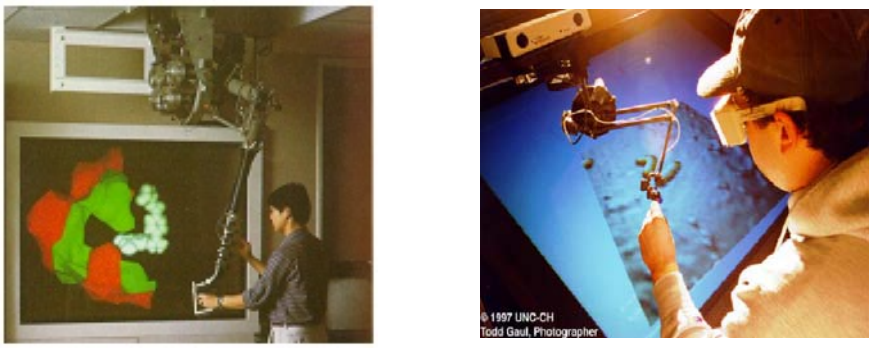
Otra diferencia se encuentra en el tamaño del háptico así como la forma de manipular ese dispositivo háptico. Para aplicaciones donde la fuerza es considerable, el agarre del sistema háptico se realiza con la totalidad de la mano. Un ejemplo es el desmontaje de objetos en tareas de mantenibilidad con motores de aviación (McNeely et al. 1999) como aparece en la Figura 5.4a. La otra alternativa son las aplicaciones donde se necesita una mayor precisión, como en las aplicaciones quirúrgicas, en las que el agarre se realiza con los dedos (Figura 5.4b).



(a) (b)

Figura 5.4: Simulación de mantenibilidad de la compañía Boeing (a) y operación quirúrgica transatlántica realizada con el sistema de teleoperación *ZEUS* de Computer Motion (b).

El departamento de Ciencias de la Computación de la Universidad de Carolina del Norte posee una interfaz de los dos tipos para la visualización e interacción en escalas del nanómetro (Taylor II 1999). Para la simulación de interacciones entre proteinas y drogas, han construido el dispositivo de la Figura 5.5a, y para el examen y manipulación de virus, bacterias y secuencias de ADN, han desarrollado el sistema de la Figura 5.5b.



(a) (b)

Figura 5.5: La aplicación *Docker* simulando las fuerzas entre una droga y su receptor proteínico (a) y la aplicación *nanoManipulator* actuando como un microscopio de gran escala.

Otra clasificación para esta clase de dispositivos se basa en los grados de libertad que posee (DOF, *Degree Of Freedom*): 2D, 3D, 6D, etc. Un háptico muy extendido es el llamado *PHANToM* (Personal HAptic iNTerface Mechanism) de SensAble Technologies Inc. del cual existen varias versiones. Los primeros dispositivos eran 6DOF-in/3DOF-out (Figura 5.6a). Es decir, el háptico tiene sensorizados 6 grados de libertad para definir la posición y la orientación del usuario pero sólo 3 grados son actuados con fuerzas, los correspondientes a las translaciones. Los 6 grados de libertad actuados se pueden obtener con versiones más modernas (Figura 5.6b).

Figura 5.6: *PHANToM* Premium 1.0 (a) y *PHANToM* 1.5/6DOF (b).

Existen sistemas que combinan varias clases de los dispositivos vistos en este apartado. La empresa Immersión combina dos guantes *CyberGrasp* con sendos hápticos anclados al suelo y sujetos al usuario por las muñecas. Esto permite que además de tener sensaciones táctiles en cada falange de las manos, el sistema puede ofrecer también sensaciones como el agarre de objetos. En la Figura 5.7 se puede apreciar a un usuario usando este sistema.

Figura 5.7: Sistema *Haptic Workstation* de Immersion.

5.3 ANTECEDENTES

El problema del contacto y la respuesta adecuada está influido por tres subproblemas: el problema de la detección de colisiones, el cálculo de la distancia y/o penetración y el cómputo de las fuerzas.

El problema de las colisiones ya ha sido estudiado con detenimiento en el capítulo 1. Dependiendo del método usado en este paso, el cálculo de la respuesta de colisión será más o menos fácil. Por ejemplo, los métodos basados en proximidad son más directos para el cálculo de la penetración porque ya se está calculando una distancia entre los objetos (Mirtich 1998). Sin embargo, métodos más comunes que utilizan partición espacial necesitan arreglárselas con menos información, por ejemplo sólo conociendo los triángulos en contacto (Savall et al. 2002).

Para el cálculo de la penetración existen muchos trabajos previos. Algunos se basan en las regiones de *Voronoi* (Lin y Canny 1991 y Mirtich 1998), mientras que otros usan técnicas basadas en la diferencia de Minkowski (Gilbert et al. 1988, Cameron 1997, Van Den Bergen 1999, Van Den Bergen 2001 y Kim et al. 2002b). Todos ellos tratan de dar un valor a la penetración sufrida por dos objetos. Esta penetración se define como la mínima distancia que se aplica a uno de los objetos para que desaparezca la intersección. El orden de complejidad de estos algoritmos depende del número de vértices en la escena por lo que resultan poco escalables para entornos masivos.

Finalmente, para el cálculo de las fuerzas de contacto, llamado en la bibliografía *haptic rendering*, los métodos dependen del tipo de interacción con el dispositivo (Basdogan y Srinivasan 2002). Estos pueden ser métodos basados en punto, en rayo o en objetos.

Los métodos basados en punto sólo tienen en cuenta un único punto en la escena gráfica. Es decir, el dispositivo háptico se representa en el escenario mediante un punto. El cómputo de la penetración en este caso es sencillo y se calcula como la distancia entre ese punto y un punto de la superficie intersectada (Zilles y Salisbury 1995 y Salisbury et al. 1995).

Si la interfaz háptica se modela mediante una línea finita, la interacción se dice que está basada en rayo o segmento de línea. Las colisiones se calcularían entre ese segmento y los polígonos con los que se cruce (Gregory et al. 2000a y Ho et al. 2000). Las técnicas basadas en rayo se pueden considerar como una aproximación a una herramienta alargada pero no es suficiente para muchas aplicaciones en las que la herramienta se modela como un objeto poliédrico complejo.

Como ejemplo de esto último, en las simulaciones de mantenibilidad se necesita conocer con precisión el acceso de la herramienta a ciertos emplazamientos por lo que en estos casos se requiere una interacción 3D objeto-objeto. En esta línea, existen pocos trabajos. Algunos sólo llegan hasta el nivel de precisión del voxel ya que carecen de modelo faceteado (McNeely et al. 1999 y Renz et al. 2001). Otros autores llegan al nivel de las primitivas usando regiones Voronoi (Gregory et al. 2000b) o diferencias de Minkowski (Kim et. al. 2002a). Sin embargo, en estos últimos ejemplos los escenarios contienen decenas de miles de polígonos y no hay constancia de experimentos con modelos masivos.

5.4 DESCRIPCIÓN DEL MÉTODO

El cálculo final de fuerzas a restituir al usuario se lleva a cabo conjuntamente entre los módulos de Colisión y Control, por lo tanto este apartado divide y explica por separado los cálculos efectuados por cada módulo. A lo largo del apartado, se van presentando gradualmente problemas de cuyo análisis se sugieren las sucesivas mejoras que se desarrollan a lo largo del mismo. Las distintas mejoras introducidas, conducen al método presentado.

Basándose en los antecedentes presentados en el presente capítulo, el método propuesto está basado en la interacción háptica de objetos 3D. Este método es más preciso que los basados en punto y rayo. La desventaja es su mayor carga computacional pero como se verá posteriormente y en el siguiente

capítulo, se demuestra experimentalmente que su carga en el cómputo global de colisiones no es significativa.

5.4.1 Objetivo

Debido al diseño mecánico de la interfaz háptica que se ha utilizado en esta Tesis, el sistema sólo debe calcular y devolver una fuerza tridimensional al usuario. Al carecer el háptico de giros actuados, no hay que calcular pares de fuerzas. Muchos autores utilizan técnicas basadas en un punto o en un rayo ya que al no poseer pares, sólo es necesario calcular una dirección y un módulo para la fuerza. Sin embargo, las aplicaciones de mantenibilidad exigen que la herramienta sea modelada como un objeto 3D ya que es la única forma de realizar tareas en las mismas condiciones que en la realidad.

El modelo de contacto es el de un muelle simple (Burdea 1996) y la fórmula de la fuerza a restituir está gobernada por la ley de Hooke tal y como aparece en la ecuación (5.1).

$$\mathbf{F} = k x \mathbf{n} \tag{5.1}$$

donde k es la constante de rigidez del sistema, x es la penetración y $\mathbf{n}$ es la dirección de la fuerza o normal de contacto. Estos son los valores finales que utiliza el módulo de Control para calcular la fuerza de contacto. Sin embargo, el módulo de Colisiones también calcula unos valores que son enviados al módulo de Control para el cálculo de esa fuerza. Para evitar confundir los parámetros calculados por cada módulo, en la Tabla 5.1 se refleja el símbolo usado por cada uno de ellos para representar la normal de contacto y la penetración.

	Módulo de Colisiones	Módulo de Control
Normal de contacto	$\mathbf{n_c}$	$\mathbf{n}$
Penetración	d	x

Tabla 5.1: Símbolos usados por cada módulo para los valores de contacto.

Recapitulando, el módulo de Colisiones tiene dos importantes problemas: la detección de colisiones y la respuesta de colisión. La primera tarea detecta todas las intersecciones entre los objetos de la escena y ofrece una respuesta visual y/o acústica al usuario (por ejemplo dibujando los polígonos en colisión). Más formalmente, dado un triángulo t y un conjunto de triángulos T, la función booleana *col(t, T)* evaluará *true* si y sólo si $\exists t' \in T \mid t \; y \; t'$ intersectan; y si S y M son los conjuntos de triángulos de los objetos estático y móvil respectivamente, la detección de colisiones calcula dos conjuntos S_c y M_c definidos como:

$$S_c = \{t \in S, col(t, M) = true\}$$
$$M_c = \{t \in M, col(t, S) = true\} \quad (5.2)$$

La segunda tarea toma la información generada por la detección de colisiones y calcula la respuesta específica que requiere la aplicación, en este caso una dirección y una penetración. La Figura 5.8 representa el flujo de información entre los distintos módulos.

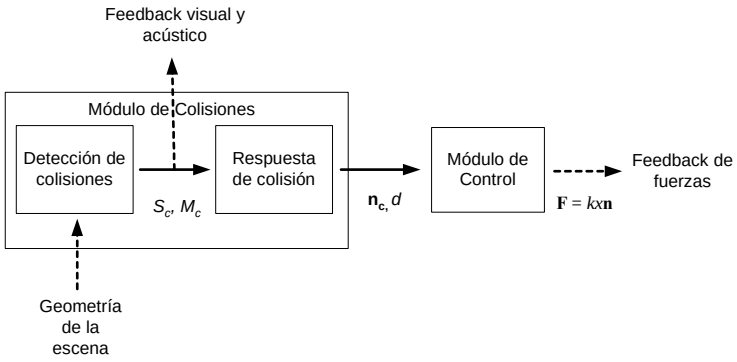


Figura 5.8: Flujo de información entre los módulos de Colisión y Control.

Tal y como se puede observar, el módulo de Colisiones estima la normal y penetración del contacto y la envía al módulo de Control. Sin embargo, este módulo tiene una frecuencia de muestreo mayor que la frecuencia de cálculo de las colisiones, por lo tanto el módulo de Control deberá usar diferentes técnicas de integración con esos valores para evitar fuerzas bruscas en el contacto. La arquitectura y frecuencias específicas de cada módulo son presentadas en el capítulo siguiente. En este capítulo sólo es necesario conocer que el módulo de Control, también llamado lazo de control o lazo rápido, se ejecuta a una frecuencia mayor que la detección de colisiones.

En los dos siguientes apartados se describe primero cómo calcula el módulo de Colisiones la dirección de la fuerza y la penetración entre los objetos, y segundo como usa el módulo de Control esos valores.

5.4.2 Respuesta de Colisión

Este algoritmo es el encargado de obtener geométricamente los valores necesarios para determinar la fuerza. Para ello utiliza los triángulos en contacto de uno y otro objeto. Al operar con triángulos se alcanza el máximo nivel de precisión ofrecido por el modelo CAD exportado.

Tal y como se verá en el siguiente capítulo, los módulos de Colisiones y de Control se ejecutan en dos PCs diferentes conectados por red. Los retrasos en las comunicaciones pueden originar problemas en el control del háptico por lo que es necesario que el módulo de Colisiones envíe por la red la mínima información posible, en este caso una dirección normal y una penetración. Aunque se envían datos para determinar una única fuerza, el módulo de Colisiones puede detectar si existe una o más zonas de contacto entre el objeto estático y el móvil y todas influyen en la respuesta que se calcula.

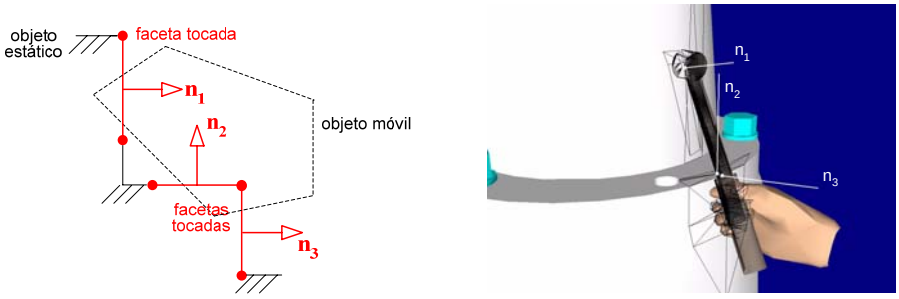
5.4.2.1 Cálculo de la dirección normal

La dirección normal o normal de contacto, $\mathbf{n_c}$, es el vector normal asociado al plano de contacto. Este vector representa la dirección de la fuerza que se restituirá al usuario a través del dispositivo háptico.

Para el cálculo de esta normal, se tiene en cuenta información de la superficie estática y más en concreto de las normales de los triángulos estáticos que están colisionando en ese momento. Ninguna información del objeto móvil o herramienta se usa para esta fase de la computación.

El método distingue entre dos posibles casos: la herramienta está tocando una única zona o por el contrario tiene colisión con varias zonas de contacto que pueden pertenecer o no a un mismo objeto. Es decir, los triángulos de S_c pueden pertenecer a una zona de contacto o varias. Así que dado un conjunto S_c, es necesario dar respuesta a este problema.

Un primer método para determinar si dos triángulos pertenecen a una misma zona de contacto consiste en estudiar si sus normales tienen direcciones parecidas, es decir, si su producto escalar es cercano a 1. Sin embargo, esta condición no es suficiente ya que varias facetas pueden tener normales parecidas y pertenecer a zonas distintas como sucede con las normales $\mathbf{n_1}$ y $\mathbf{n_3}$ de la Figura 5.9.



(a) (b)

Figura 5.9: Ejemplo 2D y 3D de una colisión con varias zonas de contacto.

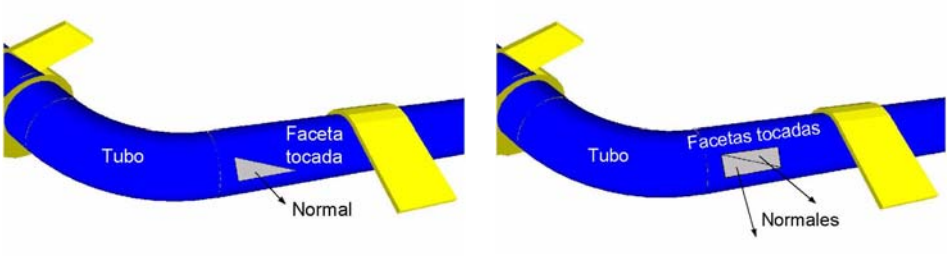
El cálculo de la penetración depende de dónde se sitúe el plano de contacto. En el ejemplo de la Figura 5.9 se calcularía un plano erróneo ya que $\mathbf{n_1}$ y $\mathbf{n_3}$ se considerarían normales parecidas pero se puede comprobar cualitativamente que realmente los contactos están alejados entre sí. Por lo tanto es necesario otro método para identificar zonas de contacto diferentes. El algoritmo utilizado primero evalúa las direcciones de las normales para una primera clasificación. En segundo lugar testea todos los triángulos verificando las adyacencias geométricas entre ellos. Básicamente, el algoritmo analiza la topología de los triángulos y en cuanto detecta dos triángulos compartiendo un mismo vértice, los introduce en la misma zona de contacto. Las distintas zonas de contacto que se obtienen cumplen lo siguiente:

- Las normales de los triángulos de cada zona son muy parecidas entre sí.

- Todos los triángulos de una zona de contacto comparten algún vértice con algún triángulo de esa misma zona.

5.4.2.1.1 Caso de una zona de contacto

En este caso la herramienta u objeto móvil está tocando una o varias facetas del escenario estático pero en una única zona de contacto. Si sólo hay una faceta de la superficie estática implicada en la colisión, el cálculo de la normal de contacto es muy fácil y coincide con la normal de la faceta tocada (Figura 5.10a).



(a) (b)

Figura 5.10: Una zona de contacto con una (a) o varias (b) facetas implicadas.

Sin embargo, los triángulos de las maquetas manejadas en este tipo de aplicaciones son muy pequeños y normalmente habrá más de un triángulo implicado en la colisión (Figura 5.10b). La solución más rápida es realizar la suma vectorial de las f normales implicadas y acto seguido normalizar la solución, como se muestra en la ecuación (5.3).

$$\mathbf{n}_c = \sum_{i=1}^{f} \mathbf{n}_i \quad , \quad \mathbf{n}_c = \frac{\mathbf{n}_c}{|\mathbf{n}_c|} \tag{5.3}$$

Este método puede presentar dos problemas. El primero ocurre cuando la herramienta penetra una distancia considerable dentro del objeto estático. En esta situación puede que traspase un objeto y toque otros objetos interiores. Las normales de los triángulos de esos objetos pueden introducir un error en el resultado de la normal de contacto. Sin embargo, la realimentación de fuerzas evita que se produzcan grandes penetraciones en una simulación por lo que este caso puede ser evitado eligiendo la adecuada rigidez para el sistema.

Otro problema aparece si la suma de las normales es nula. Este hecho se puede dar si la herramienta corta transversalmente un objeto cilíndrico. El típico ejemplo de este tipo de problema se puede observar en la Figura 5.11 que representa una colisión contra unos tubos. Estos elementos pueden ser muy largos y finos y la herramienta puede llegar a atravesarlos con facilidad.

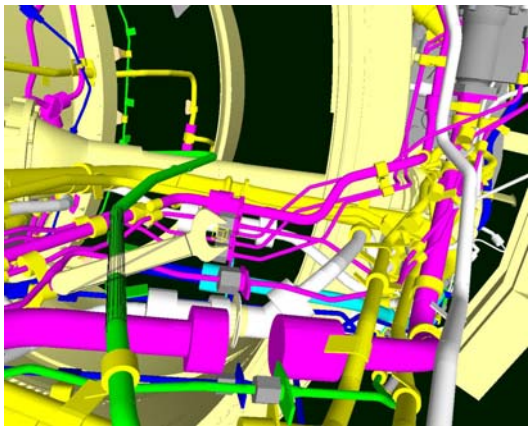


Figura 5.11: Colisión de la herramienta atravesando un tubo.

Para solucionar este problema, el módulo de Colisiones implementa una técnica de *Back-face Culling* aplicada a las colisiones (Vaněček 1994). Si una faceta estática que está en colisión tiene una normal en la dirección de la trayectoria de la herramienta, entonces esa faceta no se tiene en cuenta para el cálculo de la normal de contacto $\mathbf{n}_c$. Más formalmente, la faceta *i* es rechazada si $\mathbf{v} \cdot \mathbf{n}_i > 0$ donde $\mathbf{v}$ es el vector velocidad del objeto móvil y $\mathbf{n}_i$ es la normal de la faceta *i*. De esta manera también se incrementa la eficiencia del cálculo ya que se rechazan cálculos innecesarios para el resultado. A pesar de todo, si se detecta que la suma vectorial de normales sigue siendo nula, se toma la última normal de contacto coherente.

5.4.2.1.2 Caso de varias zonas de contacto

Una situación muy probable en una simulación de mantenibilidad es la colisión de la herramienta con varios objetos al mismo tiempo o con varias zonas del mismo objeto. En la Figura 5.12 aparece un objeto móvil intersectando con dos paredes de la superficie estática. Parece lógico pensar en dividir el contacto y tratar varios contactos por separado. En el apartado anterior se explicaba el método usado para discriminar las diferentes zonas de contacto.

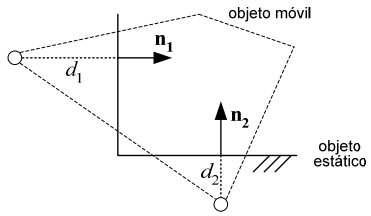


Figura 5.12: Colisión del objeto móvil con dos zonas de contacto.

Cuando se han detectado las diferentes zonas, se utiliza el método expuesto en el apartado anterior para calcular las diferentes sumas vectoriales de normales para cada contacto, $\mathbf{n_1}$ y $\mathbf{n_2}$. Las penetraciones de cada zona de contacto, d_1 y d_2, también se calculan pero el método se describirá en el siguiente apartado.

El apartado 5.4.3 explica como el modelo de contacto usado por el módulo de Control sólo tiene en cuenta una fuerza. Esta restricción viene impuesta por el mecanismo que actúa en tres grados, es decir, proporciona fuerzas y no pares. De este modo, el módulo de Colisiones tiene una carga computacional añadida que consiste en convertir varios contactos en una única acción física. La normal $\mathbf{n_c}$ corresponde a esa única normal que el módulo de Colisiones envía al módulo de Control.

La normal $\mathbf{n_c}$, depende de la penetración en cada zona de contacto. En la Figura 5.12 se observa que $\mathbf{n_1}$ debería tener más peso que $\mathbf{n_2}$ en la suma total. Esto es expresado matemáticamente por la ecuación (5.4).

$$\mathbf{n_c} = \sum_{i=1}^{m} \mathbf{n_i} d_i \quad , \quad \mathbf{n_c} = \frac{\mathbf{n_c}}{|\mathbf{n_c}|} \tag{5.4}$$

donde m es el número de zonas de contacto. Ya que se necesita calcular una única penetración para enviarla al módulo de Control, ésta se podría escoger como el módulo del vector $\mathbf{n_c}$ antes de ser normalizado. Sin embargo, las pruebas experimentales demuestran que este resultado a veces toma valores elevados produciendo contactos muy bruscos. La solución ideada envía como

penetración final la mínima entre todas las penetraciones, es decir, se escoge la mínima d_i.

Con esta estrategia se logra ocultar al algoritmo de control la existencia de varias zonas de contacto y enviarle únicamente una dirección y una penetración para la fuerza a restituir.

5.4.2.2 Cálculo de la penetración

En esta ocasión y al contrario que en el cálculo de la normal, para estimar la distancia que la herramienta ha penetrado en el objeto estático, se necesita información geométrica de ambos objetos. Antes de medir la penetración, hace falta definir el plano de contacto.

5.4.2.2.1 Definición del plano de contacto

Primero, usando información del objeto estático, se calcula un plano de contacto Π ya que la penetración se mide en base a ese plano. Si hubiera más de una zona de contacto, se calcularían tantos planos Π_i como normales $\mathbf{n_i}$ existan.

Ya que el plano debe ser perpendicular a la normal $\mathbf{n_i}$ asociada a esa zona de contacto y calculada en el apartado anterior, el algoritmo sólo necesita un punto $\mathbf{q}$ para definir completamente la ecuación del plano (5.5).

$$\Pi_i : \mathbf{n_i} \cdot \mathbf{x} + t = 0 \quad , \quad t = -\mathbf{n_i} \cdot \mathbf{q_i} \tag{5.5}$$

Donde $\mathbf{x}$ es cualquier punto perteneciente al plano Π_i. Para calcular el punto $\mathbf{q_i}$, el método propuesto busca el vértice de las facetas estáticas tocadas más alejado en la dirección de $\mathbf{n_i}$. Como el método es igual para cada zona de contacto a partir de ahora se supondrá que existe una única zona de contacto con varias facetas tocadas. La Figura 5.13 muestra cualitativamente cómo se calcula el plano de contacto.

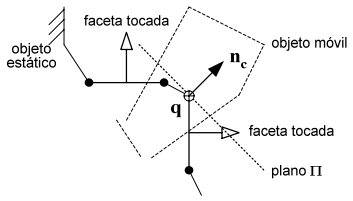


Figura 5.13: Plano de contacto Π situado en el vértice más alejado en la dirección de la normal $\mathbf{n_c}$.

El punto $\mathbf{q}$ podría haberse definido como el punto medio de las facetas estáticas en colisión pero la Figura 5.14 muestra el tipo de error que podría

suceder con ese método. En ese caso no hay ningún punto del objeto móvil que caiga en el lado negativo del plano de contacto, por lo que según el algoritmo del cálculo de la penetración (que se describe más adelante) no existe penetración entre los dos objetos.

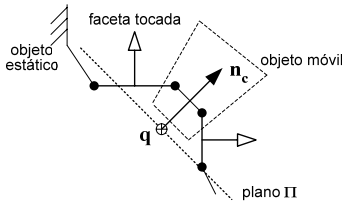


Figura 5.14: Error en el cálculo del plano de contacto eligiendo **q** como punto medio de las facetas en colisión.

El plano se tiene que calcular necesariamente en cada *frame* del módulo de Colisiones. Si se quedara fijo con la información del primer contacto, cualquier desplazamiento tangencial del objeto móvil hace que el plano fijado no corresponda con el plano real con el que se tendría que calcular la penetración.

Teniendo en cuenta que el método utiliza el vértice más alejado para el cálculo del plano $\mathbf{\Pi}$, también se pueden encontrar errores o discontinuidades en el cálculo de la penetración. Una vez que los dos objetos están en contacto, el plano de contacto $\mathbf{\Pi}$ irá dando saltos discretos conforme cambian las facetas estáticas en colisión. Si el plano de contacto cambia, también lo hará el valor de la penetración que está calculada en base a este plano. La longitud de esos saltos discretos será del orden de magnitud del tamaño de las facetas estáticas.

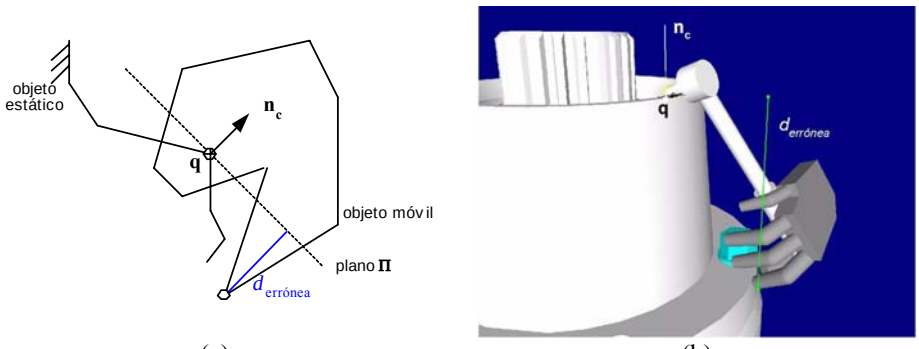
5.4.2.2.2 Medida de la penetración

Una vez que se ha calculado el plano de contacto, sólo resta estimar la penetración sufrida por el objeto móvil en el objeto estático. Esto se hace midiendo distancias de vértices del objeto móvil al plano de contacto. Una aproximación que parece válida es la de tomar la penetración como la máxima distancia entre el plano de contacto y los vértices que caigan en el lado negativo del plano. Formalmente se describiría mediante la ecuación (5.6) en la que V_m es el conjunto de vértices del objeto móvil y *dist()* es la distancia de un punto a un plano.

$$d = \max\{dist(\mathbf{v}, \mathbf{\Pi})\} \quad , \quad \mathbf{v} \in V_m \quad \wedge \quad \mathbf{n_c} \cdot \mathbf{v} + t < 0 \tag{5.6}$$

Sin embargo, esta aproximación es errónea trabajando con objetos no convexos porque la penetración se podría calcular en un punto que no tuviera

nada que ver con ese contacto. Esto podría llegar a dar no sólo penetraciones erróneas sino valores muy lejanos a la realidad con la consiguiente peligrosidad de cara a la restitución de fuerzas al usuario. En la Figura 5.15a se muestra un ejemplo 2D del error y en la Figura 5.15b aparece el mismo error en una simulación 3D. En la simulación se representaba la normal de contacto con una línea blanca y la penetración medida mediante una línea verde.



(a) (b)

Figura 5.15: Ejemplos 2D y 3D del cálculo erróneo de la penetración usando el método del vértice más alejado al plano de contacto.

Buscar entre aquellos puntos que estén en el interior del objeto estático tampoco serviría porque se podría dar el mismo tipo de error. Por lo tanto, el método propuesto opta por restringir la búsqueda a los vértices de los triángulos del objeto móvil que están en contacto. En otras palabras, por cada zona de contacto, el algoritmo busca la máxima distancia entre los vértices de los triángulos móviles que pertenezcan a esa zona de contacto. Para saber a qué zona de contacto pertenece un triángulo móvil en concreto, se mira la caja contenedora de ese triángulo y de la nube de puntos de la zona de contacto.

Con este método no se tiene en cuenta las facetas del objeto móvil que están dentro del objeto estático pero no están en colisión. Por tanto, la penetración medida puede no coincidir con la real. En la Figura 5.16a y Figura 5.16b se ve un ejemplo de cálculo correcto de la penetración, sin embargo en la Figura 5.16c y Figura 5.16d se comprueba que la penetración calculada no corresponde con la real (el punto **s** es el más alejado). En ambos casos el punto **r** es el punto elegido por el método para calcular la distancia al plano de contacto.

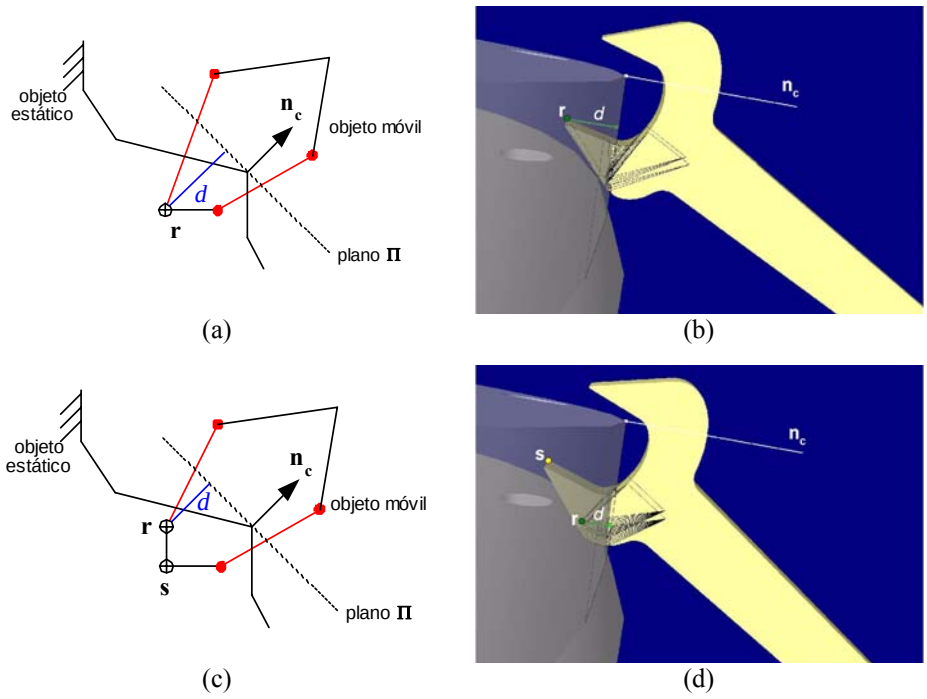


Figura 5.16: Ejemplos 2D y 3D de cálculos exactos (a) y (b) y erróneos (c) y (d) de la penetración.

Anteriormente se ha comentado que al determinar el plano de contacto aparecían discontinuidades. En este caso sucede lo mismo, ya que las facetas móviles en colisión pueden ir cambiando según el movimiento de la herramienta. En este caso, al manejar la geometría del objeto móvil, el orden del error o salto discreto coincide con el tamaño de las facetas de la herramienta.

Con el método presentado, la penetración calculada siempre será menor o igual, pero nunca mayor a la penetración real. Este aspecto es beneficioso de cara a la seguridad ya que se garantiza que la fuerza que se restituye al usuario no puede hacerse muy grande de forma inesperada.

Finalmente, en la Figura 5.17 se puede observar un instante de una simulación donde la herramienta está colisionando con objetos de la maqueta virtual. El módulo de Colisiones puede proporcionar información visual tanto de la detección como de la respuesta de colisión: la normal de contacto $\mathbf{n}_c$, la penetración d, y las facetas estáticas y móviles colisionando entre sí (dibujadas en negro).

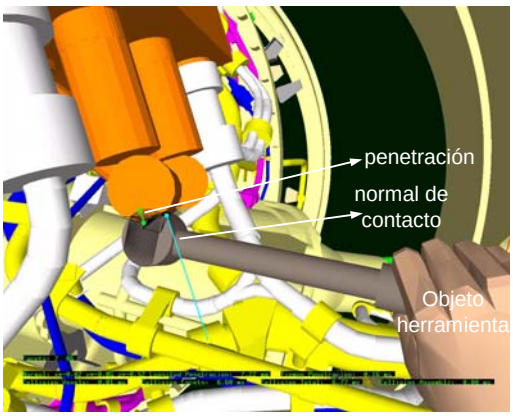


Figura 5.17: Representación gráfica de la información calculada por el módulo de Colisiones.

El cómputo llevado a cabo por el módulo de Colisiones para el cálculo de la respuesta de colisión no sobrepasa el milisegundo con lo que apenas influye en la computación total del módulo. Los experimentos con simulaciones de tareas de mantenimiento reales son presentados en el capítulo siguiente.

5.4.3 Lazo de control

Para poder valorar el trabajo aportado, es necesario también exponer cómo se integra en el lazo de control desarrollado por nuestro equipo (Savall et al. 2002).

El sistema virtual de mantenibilidad no sólo debe conseguir un buen *frame rate* en la visualización y en la gestión de las intersecciones, sino que también tiene que ofrecer un buen feedback táctil al usuario. Esta tarea es el objetivo del módulo o lazo de control.

Según los estudios de Shimoga (1992), la mayoría de autores eligen una frecuencia de muestreo entre 500 Hz y 1 kHz. Estas frecuencias logran mantener el sistema estable y cubren todos los efectos que se pueden simular en el tacto humano. Como se explica en la descripción de la arquitectura del sistema (capítulo 6), el lazo de control opera cada 2 ms. Sin embargo, el módulo de Colisiones normalmente tiene una frecuencia menor por lo que el control necesita extrapolar valores intermedios mientras no se reciba información del módulo de Colisiones. De lo contrario, la sensación táctil obtenida sería muy pobre debido a que el periodo de control vendría impuesto por el algoritmo de detección de colisiones.

El hecho de que el módulo de Colisiones calcule y envíe una única dirección normal resultante y una penetración equivalente no quiere decir que el algoritmo de control se limite simplemente a restituir una fuerza proporcional a

la penetración estimada en la dirección adecuada. El algoritmo de control debe afrontar dos problemas fundamentalmente: la actuación en ausencia de información de contacto (apartado 5.4.3.1) y la transición al nuevo contacto recibido (apartado 5.4.3.2).

Además, también se implementa la inclusión de fuerza de rozamiento. El modelo de fricción usado mejora la interacción con el háptico y logra simulaciones más realistas. De lo contrario, el usuario resbalaría por la superficie de contacto como si ésta fuera de hielo o deslizante. Se implementa un modelo de fricción basado en el descrito por Salisbury et al. 1995.

5.4.3.1 Actuación en ausencia de información de contacto

El control háptico requiere un valor normal y una penetración cada periodo de muestreo. Después del último mensaje recibido del módulo de Colisiones, existen varios muestreos en los que el usuario genera posiciones pero no se recibe ninguna información de colisiones, y a pesar de esa ausencia de información de colisión actualizada es necesario producir una nueva realimentación. Esto es causado por la diferencia de frecuencias entre los dos módulos. Por consiguiente, el módulo de Control debe estimar nuevos valores de penetración usando la última información de colisiones que posee.

Adachi (1995) usa un buen método para predecir el valor de la penetración. El sistema desarrollado utiliza este método y desarrolla técnicas adicionales para evitar o suavizar los cambios bruscos con cada nueva información de contacto recibida tal y como se verá en el siguiente apartado.

Cuando el módulo de Control recibe un valor de colisión, el lazo de control coge ese valor y la última posición muestreada del háptico y devuelve una fuerza en base a esos valores. En el siguiente muestreo del lazo de control se coge una nueva posición del usuario pero el valor de las colisiones pertenece a un estado anterior (el módulo de colisiones no ha enviado información actualizada). Adachi extrapola nuevos valores para la penetración manteniendo constante el valor de la normal.

Asumiendo que la normal de contacto **n** no cambia, la variación de la penetración, Δx , puede ser estimada de la siguiente manera: en cada periodo de muestreo se añade la proyección, en el eje de la dirección normal, de la diferencia entre la posición actual del háptico y la anterior. Esto se puede expresar con las ecuaciones (5.7) y (5.8).

$$\Delta x = -\Delta \mathbf{p} \cdot \mathbf{n} \tag{5.7}$$

$$x_{i+1} = x_i + \Delta x \tag{5.8}$$

En la Figura 5.18 se representa cualitativamente el heurístico donde *d* es el valor de la penetración recibida por el módulo de Colisiones. La primera extrapolación usa este valor para calcular x_1 que es la penetración usada por el módulo de Control para la restitución de esfuerzos en el muestreo 1. Las siguientes extrapolaciones x_i se calculan de la misma manera pero basadas en las penetraciones anteriores x_{i-1}. El vector trayectoria Δp_i es la traslación del háptico desde la posición *i-1* hasta la *i*. Este vector es proyectado en la normal de contacto y se usa esa proyección para calcular la nueva x_i.

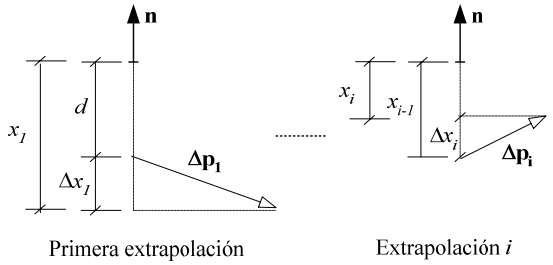


Figura 5.18: Estimación de la penetración en cada periodo de muestreo del módulo de Control.

Hay que apuntar que Δp ha sido alargada para una mejor representación pero realmente el desplazamiento del háptico entre dos muestreos es mucho más pequeño.

5.4.3.2 Transición al nuevo contacto recibido

Tal y como se describe en el apartado anterior, el módulo de Control va estimando los valores de la penetración según la última información de colisiones que posee y la posición del usuario. Sin embargo, la transición a los nuevos valores de colisión recibidos no se puede realizar de forma directa. Hay que tener en cuenta varios aspectos. El primero es el retraso (desfase) que existe debido al proceso de cálculo de la colisión y a las comunicaciones. El segundo se refiere a las técnicas que son usadas para suavizar la transición a los nuevos valores de contacto.

En el retraso influye la arquitectura basada en dos PCs usada en el sistema. El módulo de Control realiza lecturas de posición y envía esa

información al proceso gráfico que se ejecuta en un segundo PC[3]. Si el módulo de Colisiones detecta alguna interferencia en esa posición entonces envía la información de contacto al control. Sin embargo, por el hecho de tener una frecuencia menor en el cómputo de las colisiones y por el retardo implícito en la comunicación, cuando el módulo de Control recibe el nuevo contacto esa información no corresponde con la posición actual del usuario.

En la Figura 5.19 se representa el problema. El módulo de Colisiones detecta una colisión y envía la normal de contacto **n** y la penetración x_i. Estos valores son calculados para la posición *i* pero cuando el módulo de Control los recibe, la posición actual del usuario es la *i+3*.

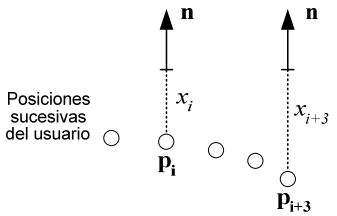


Figura 5.19: Retraso en la información de contacto.

Para calcular la nueva penetración x_j (en este ejemplo en concreto x_{i+3}) en la posición actual del usuario y minimizar el error del retardo, el módulo de Control utiliza la misma técnica que en el apartado anterior pero esta vez el vector de trayectoria se define entre las posiciones p_i y p_{i+3}. Generalizando, si se llama x_c a la penetración calculada en una cierta posición p_c, y p_a a la posición actual del usuario cuando el módulo de Control recibe la información, las ecuaciones (5.9) y (5.10) describen matemáticamente el algoritmo descrito.

$$\Delta x = -\left(\mathbf{p}_a - \mathbf{p}_c\right) \cdot \mathbf{n} \tag{5.9}$$

$$x_j = x_a + \Delta x \tag{5.10}$$

Ya que el módulo de Control necesita conocer la posición p_c en la que son calculadas las colisiones, es necesario que el módulo de Colisiones le envíe esta información junto con la normal y la penetración.

---

[3] Este segundo PC biprocesador ejecuta las tareas de visualización y GUI en un procesador mientras que el otro se ocupa del cálculo de las colisiones. Una descripción más detallada de esta arquitectura se ofrece en el capítulo 6

El segundo aspecto al que se hacía referencia al hablar de la transición al nuevo contacto trata de cómo suavizar el cambio de estado con la nueva información. Evidentemente, al ir a frecuencias diferentes, cuando el módulo de Control recibe la nueva normal y penetración, éstas pueden ser muy diferentes de los valores que se estaban usando. La nueva penetración recibida puede ser muy distinta de la última penetración estimada con las ecuaciones (5.7) y (5.8). Con mayor razón puede ocurrir esto con la dirección normal, que ha permanecido constante desde el anterior mensaje de penetración.

Mark et al. (1996) desarrolló un método para suavizar ese cambio brusco que consiste en calcular *n* penetraciones o transiciones intermedias, es decir, la nueva información de penetración no se aplica en el momento de ser recibida, sino que se distribuye su aplicación en *n* ciclos de control. Así se interpola *n* valores para ir restituyendo una fuerza gradual en los siguientes *n* periodos de muestreo. Con esta estrategia, queda fijada la actuación del dispositivo en los *n* periodos de muestreo siguientes a la recepción de la información de contacto.

El parámetro *n* dependerá, por tanto, del número de periodos de muestreo que pueden existir entre la recepción de dos mensajes de contacto. Si se considera x_a como la penetración recibida en el mensaje anterior de contacto y x_n la nueva penetración recibida, las ecuaciones que rigen la estrategia de Mark y que calculan cada penetración x_i del intervalo son las siguientes:

$$\Delta x = \frac{1}{n+1}\left(x_a - x_n \right) \tag{5.11}$$

$$x_i = x_a + i\Delta x \tag{5.12}$$

De todas formas, hay que tener en cuenta que la verdadera expresión de la penetración no sólo viene fijada por estas ecuaciones. En cada transición, la penetración posee una componente adicional debida a la estimación que se hace en cada periodo de muestreo (ecuaciones (5.7) y (5.8)).

El algoritmo del módulo de Control usa también esta técnica de interpolación para realizar transiciones en las direcciones de las normales calculando *n* normales intermedias como se representa en la Figura 5.20.

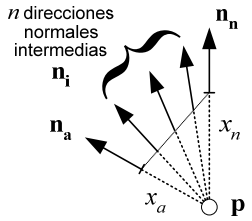


Figura 5.20: Interpolación de normales.

$\mathbf{n_a}$ y x_a son los valores de la información anterior de contacto y la nueva información recibida viene expresada por $\mathbf{n_n}$ y x_n. Más formalmente, la dirección normal $\mathbf{n_i}$ en cada instante intermedio vendría dada por la siguiente formulación.

$$\mathbf{\Delta n} = \frac{1}{n+1}\left(x_n \mathbf{n_n} - x_a \mathbf{n_a}\right) \tag{5.13}$$

$$\mathbf{n_i'} = \mathbf{n_a} + i\mathbf{\Delta n} \tag{5.14}$$

$$\mathbf{n_i} = \frac{\mathbf{n_i'}}{\left|\mathbf{n_i'}\right|} \tag{5.15}$$

El vector $\mathbf{\Delta n}$ puede considerarse como el vector "incremento constante" de la dirección normal a lo largo de las *n* situaciones intermedias. Esta forma de calcular $\mathbf{\Delta n}$ es similar al método propuesto por Constantinescu et al. (2002), con una diferencia: en la ecuación (5.13) las normales nueva y anterior son ponderadas con sus correspondientes penetraciones.

Hay que hacer notar que los vectores unitarios $\mathbf{n_i}$ de las direcciones normales intermedias no forman entre sí un ángulo constante. Este hecho es más acusado conforme las direcciones normales anterior y nueva son más dispares. Este método es mucho más rápido, desde el punto de vista de tiempo de cálculo, que intentar encontrar los vectores intermedios que formen un ángulo constante.

Un ejemplo de la actuación del módulo de Control en ausencia de información y con cambios bruscos de penetración se muestra en la Figura 5.21. Las penetraciones se muestran a lo largo del tiempo. La penetración calculada por el módulo de Colisiones es representada mediante estrellas y la estimada por el módulo de Control por círculos. El entorno estático está compuesto por 1.6 millones de polígonos. En este caso, el módulo de Colisiones ha detectado interferencias y enviado información al control cada 14 ms, sin embargo el lazo rápido tiene un periodo fijo de 2 ms.

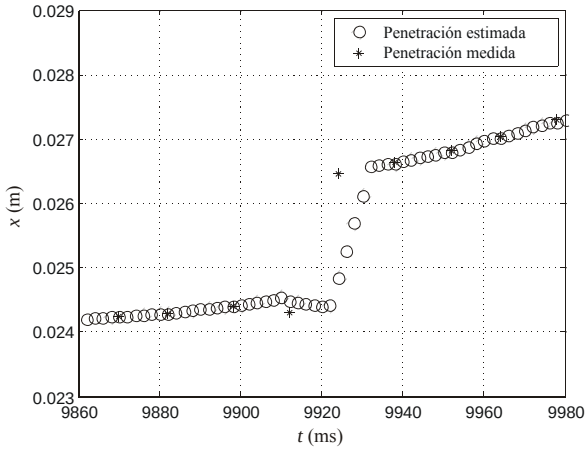


Figura 5.21: Simulación de un cambio brusco de penetración.

En ausencia de información, el módulo de Control estima bien la penetración usando el movimiento del usuario tal y como se describe en el apartado 5.4.3.1. Sin embargo, en el instante t=9924 ms el módulo de Colisiones calcula una penetración que difiere con la anterior (cambio brusco). En este caso, el módulo de Control realiza una transición gradual en los siguientes 5 periodos de muestreo. Después de esta transición sólo vuelve a actuar la estimación del control en ausencia de penetración.

A pesar de los métodos anteriores, se verificó experimentalmente que en ciertos momentos de la simulación todavía se producían fuerzas bruscas. Esto puede ocurrir cuando el módulo de Colisiones envía una normal muy diferente a la anterior. Por ejemplo, en lugares donde el usuario pueda tocar paredes con normales muy dispares entre sí, estos cambios bruscos de dirección pueden provocar rápidos movimientos de rebote de una a otra pared. Estos rebotes serán tanto más bruscos cuanto mayor sea la rigidez virtual de los objetos que se haya implementado.En este caso el algoritmo anterior que interpola *n* normales no es suficiente para generar una transición suave.

Para solucionarlo hay que modificar los dos módulos, el de colisión y el de control. Cuando el módulo de Colisiones detecte una normal demasiado diferente a la anterior, se tiene que notificar al módulo de Control con ese suceso para que éste implemente una estrategia diferente a la interpolación. El método consiste en bloquear todas las direcciones del dispositivo excepto la dirección de salida que coincide con la última normal recibida, precisamente aquella que se juzga que es muy distinta de las que se han calculado con anterioridad.

La Figura 5.22 representa este método en el que $\mathbf{p_b}$ es la posición del usuario cuando el módulo de Colisiones notifica un cambio muy brusco de dirección $\mathbf{n_b}$.

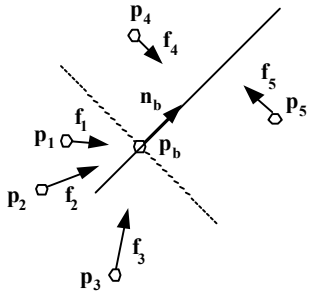


Figura 5.22: Diferentes movimientos del usuario y las respuestas del algoritmo de control para el caso de cambios muy bruscos de dirección.

Los diferentes intentos de movimiento por parte del usuario son representados por los $\mathbf{p_i}$. Si el usuario se mueve a una posición situada en el semiespacio opuesto a $\mathbf{n_b}$ ($\mathbf{p_1}$, $\mathbf{p_2}$ ó $\mathbf{p_3}$), es decir tiene un movimiento de entrada al objeto, la fuerza que se restituye será radial y esférica hacia el punto $\mathbf{p_b}$. La ecuación (5.16) describe lo anterior.

$$\mathbf{F} = k\left(\mathbf{p}_\mathbf{b} - \mathbf{p}\right) \tag{5.16}$$

Si el movimiento es en dirección a la normal $\mathbf{n_b}$ ($\mathbf{p_4}$ ó $\mathbf{p_5}$) o de salida del objeto, la fuerza será también radial pero cilíndrica proyectando esa posición en el eje imaginario de la normal $\mathbf{n_b}$ tal y como lo describe la ecuación (5.17).

$$\mathbf{F} = k\left(\left[\left(\mathbf{p}_\mathbf{b} - \mathbf{p}\right) \times \mathbf{n}_\mathbf{b}\right] \times \left(-\mathbf{n}_\mathbf{b}\right)\right) \tag{5.17}$$

Este método fuerza o "empuja" al usuario a moverse en dirección a la normal problemática. El proceso se termina cuando el módulo de Colisiones notifica que ya no existe colisión, es decir, que el usuario ha abandonado la zona de contacto.

Como se expone con más detalle en el capítulo siguiente, la combinación de todas las estrategias descritas en este apartado y en el anterior, logran generar un tacto suave aun en presencia de varias zonas de contacto o de grandes cambios de dirección en la superficie de contacto.

5.5 ÚLTIMAS CONSIDERACIONES

El problema del cálculo de la fuerza de contacto es una tarea compleja y crucial para una correcta sensación táctil. Aunque se emplee un "sencillo" modelo elástico de contacto, es necesario realizar numerosos cálculos y aproximaciones por parte de los módulos de colisiones y control.

Ciertas limitaciones que quedan patentes a lo largo de una simulación háptica son debidas a la ausencia de pares en el sistema. Otras son consecuencia de distintos factores que hay que asumir, como el retraso de las comunicaciones, o los intervalos de tiempo sin información debido a las distintas frecuencias de los módulos. Finalmente, otros errores son consecuencia exclusiva de las simplificaciones efectuadas por el método implementado. En este terreno quizá lo más difícil sea deducir información fiable cuando se está colisionando en varias zonas de contacto a la vez.

Sin embargo, estos algoritmos implementados en el sistema consiguen una sensación táctil correcta en la mayoría de los casos.

5.6 REFERENCIAS

Adachi, Y., Kumano, T. y Ogino, K. "Intermediate Representation for Stiff Virtual Objects", *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pp. 203-210. 1995.

Basdogan, C., y Srinivasan, M.A., "Haptic Rendering in Virtual Environments", *The Handbook of Virtual Environment: Design, Implementation, and Applications (Human Factors and Ergonomics). Chapter 6.* Pp. 117-134. STANNEY, K.M., Ed. Lawrence Earlbaum Inc. London. 2002.

Brederson, J.D., Ikits, M., Johnson, C. y Hansen, C., "The Visual Haptic Workbench", *The Fifth PHANToM Users Group Workshop (PUG'00)*, pp. 46-49. 2000.

Burdea, G.C., *Force and Touch Feedback for Virtual Reality*, John Wiley and Sons, Inc., New York. 1996.

Cameron, S.A., "Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra", *Proceedings of the IEEE*

International Conference on Robotics and Automation, pp. 3112-3117. Albuquerque, New Mexico. Abril (20-25) 1997.

Constantinescu, D., Salcudean, S.E., Croft, E.A., "Local Interaction Models for Haptic Rendering of Rigid Environments", *Proceedings of the 2nd IFAC Conference on Mechatronics Systems*, pp. 553-558. Berkeley, California, USA. Diciembre 2002.

Gilbert, E.G., Johnson, D.W. y Keerthi, S.S., "A Fast Procedure for Computing the Distance between Complex Objects in Three Dimensional Space", *IEEE Journal of Robotics and Automation*, Vol. 4, No. 2, pp. 193-203. Abril 1988.

Gregory, A., Lin, M.C., Gottschalk, S., y Taylor, R., "Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback", *Computational Geometry – Theory & Applications 15*, pp. 269-285. 2000.

Gregory, A., Mascarenhas, A., Ehmann, S., Lin, M.C., y Manocha, D., "Six Degree-of-Freedom Haptic Display of Polygonal Models", *Proceedings of the Conference on Visualization 2000*, pp. 139-146. 2000.

Ho, C., Basdogan, C., y Srinivasan, M.A., "Ray-Based Haptic Rendering: Force and Torque Interactions between a Line Probe and 3D Objects in Virtual Environments", *The International Journal of Robotics Research 19* (7), pp. 668-683. Julio 2000.

Hollerbach, J.M., Cohen, E., Thompson, W.B., Freier, R., Johnson, D.E., Nahvi, A., Nelson, D.D., Thompson II, T.V., y Jacobsen, S.C., "Haptic interfacing for virtual prototyping of mechanical CAD designs", *Proceedings of the ASME Design for Manufacturing Symposium*. Sacramento, CA, USA. Septiembre 1997.

Kim, Y.J., Otaduy, M.A., Lin, M.C. y Manocha, D., "Six-Degree-of-Freedom Haptic Display Using Localized Contact Computations", *Proceedings*

of the Tenth Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, pp. 209-216. Orlando, Florida. Marzo (24-25) 2002.

Kim, Y.J., Lin, M.C. y Manocha, D., "DEEP: Dual-space Expansion for Estimating Penetration depth between convex polytopes", *Proceedings of the IEEE International Conference on Robotics and Automation*, Washington D.C. Mayo (11-15) 2002.

Lin, M.C., y Canny, J.F., "A Fast Algorithm for Incremental Distance Computation", *IEEE Conference on Robotics and Automation*, pp 1008 – 1014. Sacramento, California. Abril (9-11) 1991.

Mark, W.R., Randolph, S.C., Finch, M., Van Verth, J.M., y Taylor II, R.M., "Adding Force Feedback to Graphics Systems: Issues and Solutions", *Proceeding of the ACM SIGGRAPH'96 - Computer Graphics*, New Orleans, Louisiana, pp. 447-452. Agosto 1996.

McNeely, W.A., Puterbaugh, K.D., y Troy, J.J., "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", *Proceedings of the ACM SIGGRAPH'99 - Computer Graphics,* Los Angeles, California, USA, pp. 401-408. Agosto 1999.

Mirtich, B., "V-Clip: Fast and Robust Polyhedral Collision Detection", *ACM Transactions on Graphics*, Vol. 17, No. 3, pp. 177-208. Julio 1998.

Renz, M., Preusche, C., Pötke, M., Kriegel, H., y Hirzinger, G., "Stable Haptic Interaction with Virtual Environments Using an Adapted Voxmap-PointShell Algorithm", *Proceedings of International Conference Eurohaptics 2001*. Birmingham, UK. Junio 2001.

Salisbury, K., Brock, D., Massie, T., Swarup, N., y Zilles, C., "Haptic Rendering: Programming Touch Interaction with Virtual Objects", *Proceedings 1995 Symposium on Interactive 3D Graphics*, Monterey, California, pp. 123-130. 1995.

Savall, J., Borro, D., Gil, J.J. y Matey, L., "Description of a Haptic System for Virtual Maintainability in Aeronautics", *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2887-2892, EPFL, Lausanne, Switzerland. Septiembre (30) – Octubre (4) 2002.

Shimoga, K., "Finger Force and Touch Feedback Issues in Dextrous Telemanipulation". *Proceedings of NASA-CIRSSE International Conference on Intelligent Robotic Systems for Space Exploration*, NASA, Greenbelt, MD, Septiembre 1992.

Taylor II, R.M., "Scientific Applications of Force Feeback: Molecular Simulation and Microscope Control". *SIGGRAPH'99 Course 38 Additional Couse Notes for "Haptics: From Basic Principles to Advanced Applications"*. 1999.

Van Den Bergen, G., "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects", *Journal of Graphics Tools (JGT)*, Vol. 4, No. 2, pp. 7-25. Julio 1999.

Van Den Bergen, G., "Proximity Queries and Penetration Depth Computation on 3D Game Objects", *Proceedings of the Computer Game Developer's Conference*. 2001.

VANĚČEK, G. "Back-face Culling applied to Collision Detection of Polyhedra". *Journal of Visualization and Computer Animation,* Vol. 5, No. 1, pp. 55-63. 1994.

Zilles, C.B., y Salisbury, J.K., "A Constraint-based God-object Method For Haptic Display". *Proceedings IEE/RSJ International Conference on Intelligents Robots and Systems, Human Robot Interaction, and Cooperative Robots, Vol 3*, pp. 146-151. 1995.

CAPÍTULO 6

DESCRIPCIÓN DEL PROTOTIPO Y RESULTADOS EXPERIMENTALES

6.1 INTRODUCCIÓN

El presente capítulo describe el sistema completo del prototipo desarrollado y usado para la presente Tesis.

El capítulo comienza con la descripción del prototipo (apartado 6.2). En él se verá la arquitectura del sistema con los elementos físicos que la componen y se describe cómo están distribuidos esos elementos. El módulo de Colisiones desarrollado como fruto de esta Tesis se comunica con otros módulos, por ello también se describe brevemente los diferentes módulos software que conforman el sistema y cómo interactúan entre ellos.

Una vez descrito todo el prototipo utilizado, en el apartado 6.3 se muestran los resultados experimentales generados por el sistema en condiciones normales de trabajo.

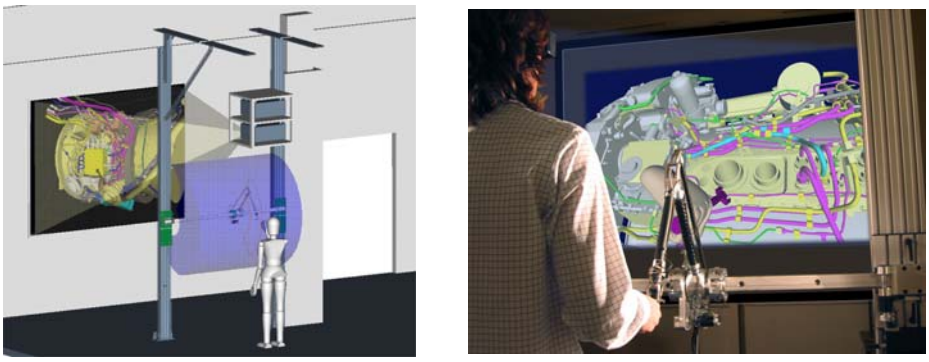
6.2 ARQUITECTURA DEL SISTEMA

Este prototipo ha sido desarrollado íntegramente en el departamento de Mecánica Aplicada del CEIT (Centro de Estudios e Investigaciones Técnicas de

Guipúzcoa). En los capítulos introductorios ya fue introducida la problemática existente en la empresa aeronáutica y más concretamente en ITP (Industria de TurboPropulsores). Para solucionarlo, se ha creado una herramienta de realidad virtual usada para comprobar el diseño y la mantenibilidad de los motores aeronáuticos.

La herramienta desarrollada llamada REVIMA (Desarrollo de una Herramienta de REalidad VIrtual para la Simulación de procesos de montaje y MAntenimiento) no sólo está orientada al mundo aeronáutico sino que puede ser aplicada en tareas de mantenimiento de cualquier sistema mecánico. Esta herramienta está diseñada para trabajar justo entre las etapas de diseño y construcción de la maqueta física, con el objetivo de ayudar a encontrar problemas de mantenibilidad lo antes posible, y por lo tanto facilitar la reducción de costes al reducir la cantidad de maquetas físicas necesarias.

Además, REVIMA puede servir como herramienta de entrenamiento antes de pasar a operar con los motores reales, o como banco de pruebas para el diseño de operaciones de mantenimiento al proporcionar pistas sobre los problemas que pueden aparecer al realizar una de estas operaciones. En la Figura 6.1 se puede observar el sistema totalmente montado y en funcionamiento.



(a) (b)

Figura 6.1: Modelo CAD del sistema (a) y el sistema real en funcionamiento (b).

6.2.1 Arquitectura del Sistema

REVIMA es un sistema multidisciplinar que incluye las siguientes líneas o áreas de investigación (Savall et al. 2002): diseño mecánico, teoría de control, gráficos por computador, geometría computacional e interacción persona-computador.

El desarrollo comprende dos áreas principalmente: diseño hardware y desarrollo software. El primero concierne al diseño, fabricación y montaje del dispositivo háptico LHIfAM (*Large Haptic Interface for Aircraft Maintainability*) y es comentado brevemente en el apartado 6.2.1.1. El desarrollo software se divide en 4 módulos: el control del háptico, la visualización, la detección de colisiones, y la GUI (*Graphical User Interface*). Esta última se encarga de centralizar todos los eventos además de ofrecer la interfaz gráfica al usuario. Los otros tres módulos se describen en los apartados 6.2.1.3, 6.2.1.4, 6.2.1.2 respectivamente. Los módulos software están basados en C++ y en la librería gráfica OpenGL. El entorno de trabajo ha sido Microsoft Visual C++ y el sistema operativo Microsoft Windows 2000.

En la Figura 6.2 se puede observar un esquema del sistema global. El sistema se ejecuta en 2 PCs llamados PC de Control y PC de Simulación. El primero se encarga de ejecutar el bucle o lazo de control que maneja y controla el LHIfAM. Básicamente, se dedica a la lectura de posiciones del dispositivo, su envío al PC de simulación y la lectura de información de colisiones con la que podrá calcular la fuerza a ejercer por los motores del háptico.

El PC de simulación es un procesador dual con el objetivo de paralelizar las tareas de simulación. Mientras que en un procesador se ejecutan las tareas de visualización y GUI, el otro procesador está dedicado por entero al cálculo de las colisiones. Esto permite tener dos procesos asíncronos en los que el cómputo de uno no influye para nada en la frecuencia de cálculo del otro. Este computador posee además 2 GB de memoria RAM necesarios para almacenar tanto la geometría de la escena como la estructura de voxels utilizada por el módulo de Colisiones. Ambos PC están conectados a través de una LAN Ethernet usando protocolos de comunicaciones de Internet.

Esta arquitectura que separa el control del háptico en un PC y la simulación en otro PC distinto ya ha sido usada anteriormente (Mark et al. 1996, McNeely et al. 1999 y Gregory et al. 2000). Sin embargo, en esos casos la simulación no se ejecutaba también en paralelo sino que únicamente se disponía de un procesador para todas las funciones.

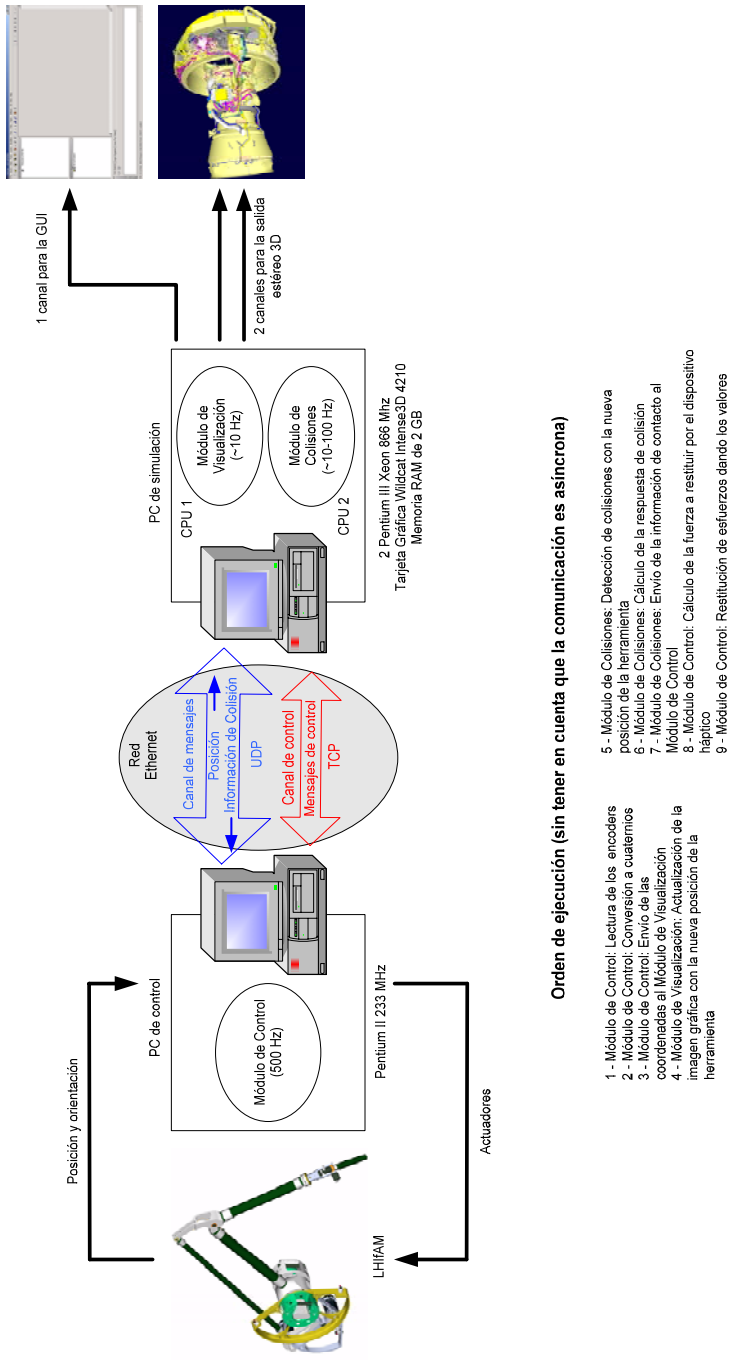


Figura 6.2: Esquema de la arquitectura del sistema REVIMA.

Las razones por las que se ha elegido esta arquitectura en concreto son las siguientes:

- El lazo de control necesita ejecutarse con prioridad de tiempo real para conseguir una frecuencia de muestreo de al menos 500 Hz.

- Esta arquitectura permite reusar la interfaz háptica en otras aplicaciones.

- De la misma forma, también es fácil conectar diferentes dispositivos al PC de simulación. Hasta ahora se han conectado los hápticos PHANToM Premium 1.0 y LHIfAM, un ratón 3D de Ascensión Technology y un 6-DOF Space Mouse de 3Dconnexion.

La salida al usuario del PC de simulación se realiza a través de 3 canales de vídeo. Uno muestra la interfaz gráfica gracias a la cual se maneja toda la aplicación (incluido el PC de control), y dos canales más para la salida visual que permite una visualización en estéreo de la escena virtual.

A continuación se describen brevemente los diferentes módulos del sistema y la comunicación que existe entre ellos.

6.2.1.1 Módulo hardware: Descripción del LHIfAM

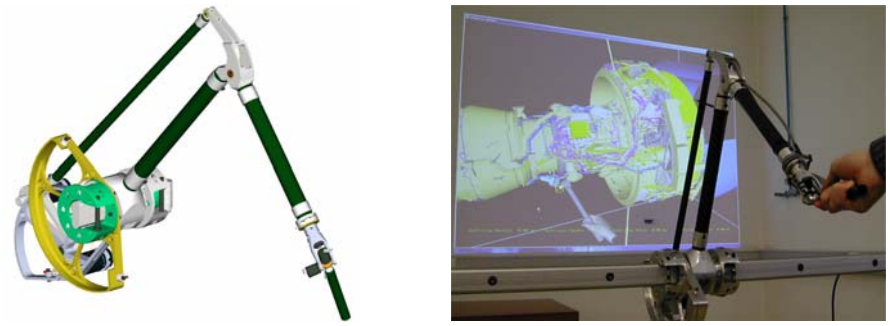
El reto principal del háptico ha sido el conseguir un espacio de trabajo con unas dimensionas tan grandes como una turbina aeronáutica. Esto se ha logrado gracias a que el dispositivo está montado sobre una guía que coincide con el eje longitudinal de lo que sería el motor.

El primer prototipo que se montó usaba el háptico PHANToM Premium 1.0 de 6 DOF con 3 de ellos actuados y fue montado sobre una guía lineal tal y como aparece en la Figura 6.3.

Figura 6.3: PHANToM sobre una guía lineal.

Una vez conseguido que el sistema funcionase correctamente con este pequeño prototipo, se montó el prototipo final utilizando el háptico LHIfAM (Figura 6.4a). Al igual que el PHANToM, este háptico posee 6 DOF pero sólo 3 de ellos están actuados. El desplazamiento a lo largo del eje del cilindro del motor se consigue usando la guía lineal mientras que para conseguir los otros dos grados de libertad se monta sobre esa guía un mecanismo paralelogramo.

El espacio de trabajo corresponde con un sector cilíndrico aproximado al área de trabajo ocupada por una maqueta aeronáutica. En la Figura 6.4b se puede ver el dispositivo sobre su guía y al usuario realizando una operación de desensamblaje sobre el modelo virtual.



(a) (b)

Figura 6.4: Modelo CAD del LHIfAM (a) y usuario manejando el LHIfAM con una maqueta virtual (b).

Algunos datos del dispositivo hardware se enumeran a continuación:

- La ventaja del tamaño del dispositivo en cuanto al espacio de trabajo tiene como inconveniente la inercia que aparece cuando se intenta mover el háptico a lo largo de la guía lineal. Para compensar esa inercia se hace uso de un sensor de fuerza gracias al cual se puede medir la fuerza que intenta hacer el usuario y de este modo ayudarle en esa traslación.

- Las dimensiones del espacio cilíndrico son: radio interno, 242 mm; radio externo, 742 mm; longitud de 1500 mm y ángulo de 120º. Este espacio de trabajo se muestra en la Figura 6.5.

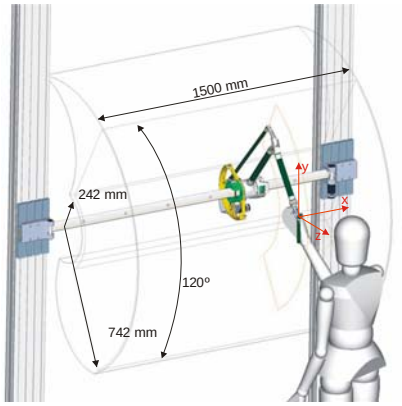


Figura 6.5: Dimensiones del espacio de trabajo del LHIfAM.

- Para poder reproducir diferentes operaciones de mantenimiento y chequear distintas situaciones desde un punto de vista ergonómico, la interfaz háptica está montada en una estructura en la que se puede graduar la altura del dispositivo y además el sector cilíndrico puede ser reubicado en caso necesario. Básicamente se quiere simular las posiciones que se muestran en la Figura 6.6.

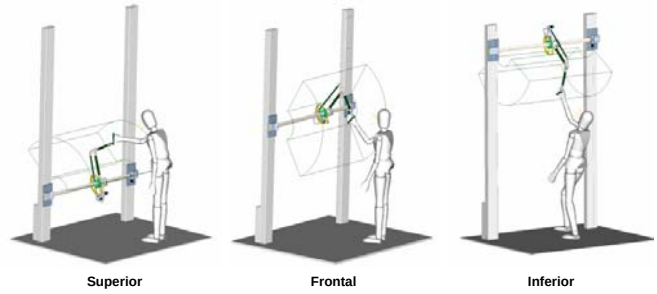


Figura 6.6: Diferentes reubicaciones del espacio de trabajo.

6.2.1.2 Módulo de Control

Por su importancia en el cálculo de la respuesta de colisión, este módulo ya se comentó en detalle en el capítulo 5 al describir los algoritmos usados para calcular la fuerza restitutiva.

El objetivo del control es la restitución de esfuerzos a través del dispositivo háptico usando la información obtenida por el módulo de Colisiones. El modelo de contacto es proporcional a la penetración entre los dos objetos (modelo muelle) utilizando una constante de rigidez (k) que se elige

experimentalmente con el máximo valor que mantenga la estabilidad del sistema.

Generalizando, las tareas de este módulo son:

- Lectura de los encoders del dispositivo háptico y envío de esa posición y orientación al Módulo de Visualización. Esta operación se realiza cada periodo de muestreo del lazo de control (500 Hz).

- Recepción de información sobre las colisiones ocurridas en la escena virtual. Esta información se recibe a una frecuencia menor que la de control (exactamente a la frecuencia de cálculo del Módulo de Colisiones) por lo que el Módulo de Control utiliza técnicas de interpolación y extrapolación descritas en el apartado 5.4.3.

- Una vez calculados los valores del módulo y dirección de la fuerza a restituir, el Módulo de Control aplica los valores adecuados en los motores para obtener esa fuerza.

Además de los mensajes de posicionamiento y colisiones, este módulo también intercambia una serie de mensajes de control. Estos pueden ser órdenes remotas de arranque o desconexión, mensajes de problemas con el dispositivo, o también mensajes de propiedades de los objetos. Estos últimos ocurren cuando el usuario expresa el deseo de coger o desensamblar algún elemento de la maqueta. El software del PC de simulación es el que tiene acceso a la geometría de la escena por lo tanto necesita comunicar las propiedades de ese objeto al Módulo de Control para que éste actúe en consonancia con el objeto.

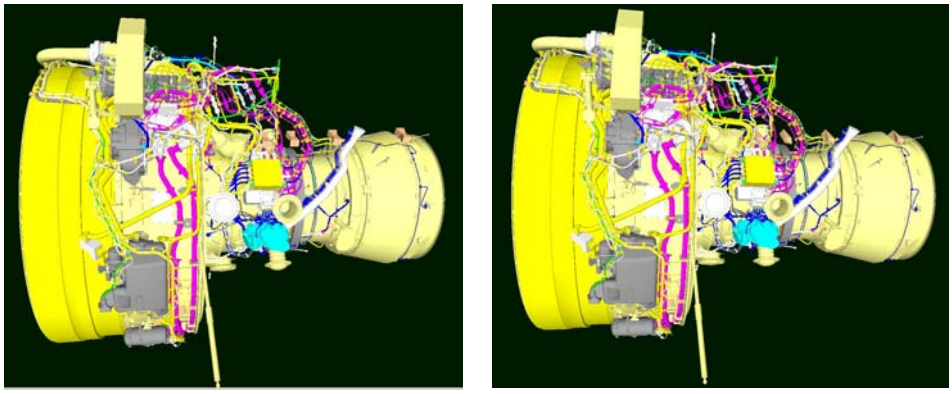
Un ejemplo típico es el desensamblaje de un tornillo. El Módulo de Control necesita conocer varios datos. El peso es uno de ellos ya que el háptico es capaz de simular pesos de objetos. Mientras se está realizando el desmontaje, el háptico debería restringir el movimiento hacia dentro o hacia fuera en dirección al eje del elemento (efecto “tunel”) por lo que se necesita conocer el eje. La longitud del elemento también es necesaria para saber cuándo se ha terminado la operación.

6.2.1.3 Módulo de Visualización

El módulo visual es el encargado de todo lo relativo a la escena gráfica que se visualiza. Como requerimiento de interactividad se impone un *frame rate* mínimo de 10 Hz pero este valor puede ser superado de diferentes formas.

- Si la maqueta es demasiado densa y pesada de interactuar con ella, una solución es aplicar previamente un proceso de simplificación. Se

han conseguido buenos resultados disminuyendo un 40% el número de polígonos del modelo (Figura 6.7). De todas formas no es una buena solución ya que se pierde nivel de precisión respecto a las maquetas originales exportadas de CAD.



(a) (b)

Figura 6.7: Modelo de 1615172 polígonos original (a) y simplificado en un 40% (b).

- Las tareas de mantenimiento normalmente se realizan en áreas muy específicas de la maqueta. En tiempo de ejecución es posible definir volúmenes de trabajo como aparece en la Figura 6.8. El resto de la maqueta no es visible y ello incrementa el *frame rate* visual.

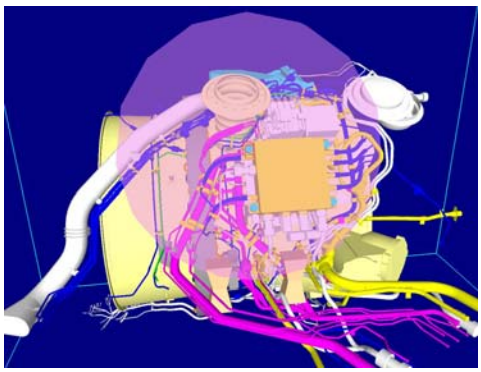


Figura 6.8: Definición de un volumen de trabajo alrededor de un elemento.

- El módulo visual implementa diversas técnicas de visualización orientadas a la optimización del *frame rate* (Amundarain et al. 2002). Entre ellas se han desarrollado estructuras optimizadas para organizar la geometría, se utilizan estructuras de organización propias de las

tarjetas de hoy en día (*display lists* y *vertex arrays*) y se implementan métodos de visualización que intentan renderizar la escena con el menor número de polígonos como son el *fustrum culling* y *occlusion culling*.

Este módulo recibe periódicamente (cada 2 ms) un mensaje de posicionamiento desde el módulo de Control. Ya que la frecuencia es mucho mayor que el *frame rate*, se tiene en cuenta sólo el último paquete recibido en ese momento. Una vez recibido el paquete se visualiza la herramienta que está moviendo el usuario en la nueva posición indicada.

6.2.1.4 Módulo de Colisiones

El módulo de Colisiones es el encargado de detectar las intersecciones ocurridas en la escena virtual y calcular y enviar la información necesaria al módulo de Control para el cálculo de la fuerza. El estudio, análisis y desarrollo de este módulo ha sido el objetivo principal de la presente Tesis.

La frecuencia del módulo de colisiones debería ser lo más parecida a la del módulo de Control para evitar inestabilidades en el sistema háptico. Debido a las características del problema, los 500 Hz no están al alcance del cálculo de las colisiones. Por esta razón, el módulo de Control tiene que implementar técnicas de interpolación y extrapolación (explicadas en el apartado 5.4.3) que estimen valores en los periodos de muestreo en los que el módulo de Colisión no envía información. Contra más se acerquen las dos frecuencias (la de control y la de colisiones) menos valores se necesitará predecir y más estable será el sistema.

La frecuencia del módulo de Colisiones depende en gran medida de la densidad de polígonos de la zona en la que se esté trabajando. Como objetivo se había impuesto una cota superior de 100 ms al igual que el módulo de Visualización. Sin embargo, la detección y respuesta en estado de colisión supera con creces este valor llegando incluso a realizar los cálculos en menos de 10 ms. Se podría determinar que el intervalo de frecuencias del módulo de Colisiones en condiciones normales de trabajo opera entre los 10 y 100 Hz.

Este módulo ocupa uno de los procesadores del PC de Simulación y comparte la memoria con el módulo de Visualización para el acceso a la geometría. En cuanto a comunicaciones, este módulo envía mensajes al PC de Control con la información calculada y necesaria para restituir la fuerza.

6.2.1.5 Comunicación entre módulos

Los diferentes módulos se comunican usando dos canales, uno de ellos usa el protocolo de comunicaciones de Internet UDP/IP y el otro usa TCP/IP.

Cada protocolo tiene sus ventajas e inconvenientes y por ello se usan ambos dependiendo del tipo de mensaje que se quiera enviar. El protocolo UDP no es tolerante a fallos ya que carece de un control de errores y tampoco asegura que los paquetes lleguen en orden. Sin embargo, por estos mismos motivos la cabecera de este protocolo es más pequeña por lo que en la práctica es más rápido que otros protocolos fiables. En aplicaciones de tiempo real en las que la pérdida de algún paquete no implique ningún riesgo se puede utilizar el protocolo UDP para la comunicación entre dos procesos. Es el caso de los mensajes de posición que se envían desde el módulo de Control hasta el módulo de Visualización. Hay dos razones más por las que la utilización de este protocolo no afecta al funcionamiento del sistema:

- Los errores o pérdidas de paquetes son habituales en el entorno de Internet pero el sistema REVIMA está construido sobre una red local con una conexión punto a punto entre los dos PCs, por lo que los errores en las comunicaciones son mínimos.

- Además no tiene importancia que se pierda algún paquete de posicionamiento ya que el módulo de control se ejecuta a una frecuencia mucho mayor que el *frame rate* de la visualización (50 veces más rápido aproximadamente).

De todas maneras, existen mensajes que sólo se envían una vez y que además hay que asegurar que ese mensaje llega a su destino. Para ello se utiliza el protocolo TCP que asegura confiabilidad en las comunicaciones. Esta fiabilidad se traduce en tres características que aseguran una correcta comunicación:

- Control de errores para evitar paquetes corruptos.

- Numeración de los paquetes para que la entrega se realice en el orden correcto.

- Reenvío del paquete si se detecta que no ha llegado a su destino.

Toda esta fiabilidad introduce retardos en la comunicación por un tamaño mayor de paquete y por reenvíos en caso de error. Estos retardos serían un problema para los mensajes que se envían continuamente como pueden ser los

de posicionamiento o los de colisiones, pero no lo es para mensajes específicos de control que requieren fiabilidad antes que rapidez.

El sistema está totalmente automatizado para controlarlo desde la GUI del PC de Simulación. Esto incluye que cualquier comando de control del háptico, desde el arranque e inicialización del háptico hasta la desconexión del mismo, se realiza de forma remota desde el PC de Simulación. Otro tipo de mensajes que necesitan fiabilidad son notificaciones al Módulo de Control de propiedades del objeto a desensamblar o al revés, avisos del Módulo de Control de problemas con el dispositivo como la incorrecta inicialización, excesivo calor de los motores, etc.

En la Figura 6.9 se representa los diferentes tipos de mensajes que se pueden cruzar entre los módulos del sistema. Cada mensaje tiene un primer campo que identifica el tipo y con ello la cantidad de información que incorpora ese paquete.

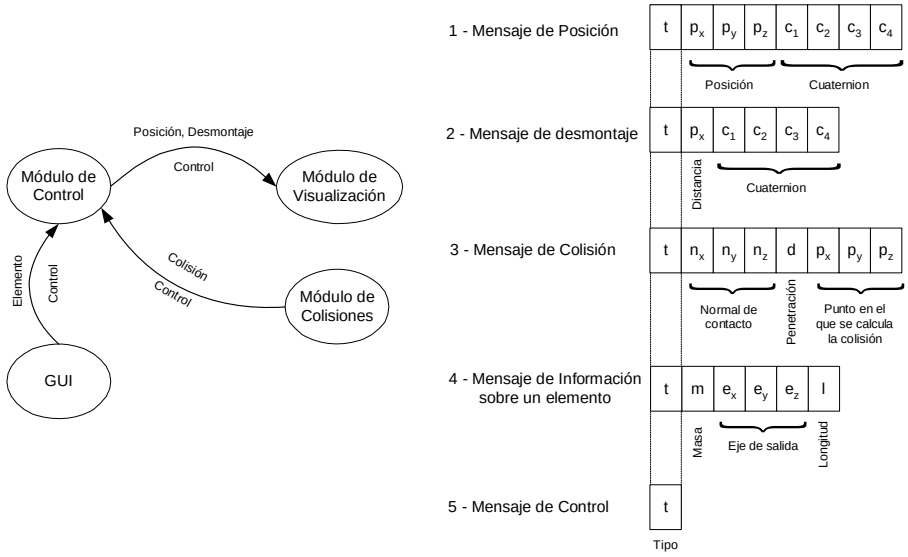


Figura 6.9: Mensajes intercambiados entre los módulos de REVIMA.

Los mensajes de control son varios dependiendo de qué se quiere comunicar pero todos ellos sólo tienen un único campo: el tipo de mensaje.

6.3 RESULTADOS EXPERIMENTALES

Este apartado analiza y describe los resultados experimentales obtenidos sobre el prototipo descrito en el apartado anterior. Estos resultados son fruto de la

implementación final del método mostrada a lo largo de los capítulos de esta Tesis.

Los diferentes experimentos se han realizado sobre un conjunto de ejemplos que cubren en la medida de lo posible varios aspectos. Por un lado se trabaja con tamaños variables de los modelos considerando todos los órdenes de magnitud que pueden llegar a encontrarse en las simulaciones de mantenimiento. Por otro lado los ejemplos que se consideran forman parte de varias tareas tipo de mantenimiento: desmontaje de tornillo, tuerca y tubo. Estos ejemplos han sido ofrecidos por la empresa ITP.

Los valores numéricos obtenidos pueden cambiar para computadoras más avanzadas, pero las conclusiones extraídas seguirían siendo las mismas.

6.3.1 Experimentos de tiempos y memoria consumida

Para observar el comportamiento de los algoritmos propuestos con una visión más global, a continuación se describen varios experimentos en los que se analizan los dos condicionantes que se han estudiado siempre a lo largo de esta Tesis: el tiempo y la memoria consumida.

Los experimentos que se han realizado tienen en cuenta los siguientes parámetros considerando que son los que más variación introducen en el método.

- Número de triángulos de la maqueta estática: un mayor número de polígonos implica un mayor número de operaciones y por tanto mayor coste computacional.

- Número de triángulos del objeto móvil: el método propuesto depende mucho del número de polígonos de la herramienta ya que el bucle principal recorre toda su geometría.

Durante los capítulos anteriores, siempre se realizaban las comparaciones y experimentos dependiendo del mismo parámetro: el tamaño del voxel. En este capítulo este parámetro desaparece ya que se supone que cada experimento se realiza con el tamaño óptimo de voxel calculado analíticamente con el método descrito en el capítulo 4. Tampoco se hace distinción entre la geometría del voxel ya que se ha demostrado que para un mismo consumo de memoria, apenas hay diferencia entre la forma paralelepípeda y la cúbica.

En todos los experimentos se ha usado la misma máquina específica para el prototipo. Las características ya se ofrecían en la Figura 6.2 pero a continuación se vuelven a recordar en la Tabla 6.1.

	PC del prototipo
Procesador	2 Pentium III Xeon a 866 MHz
Memoria RAM	2 GB
Tarjeta Gráfica	Wildcat Intense3D 4210

Tabla 6.1: Características del PC usado en las pruebas.

6.3.1.1 Estudio de tiempos

En los ejemplos siguientes se consideran modelos y herramientas típicos de trabajo. Los experimentos mostrados a continuación consisten en medir el tiempo de cálculo de colisiones entre el modelo estático y una herramienta posicionada en emplazamientos de tareas de mantenimiento típicas. El primer experimento reflejado en la Figura 6.10 pretende mostrar el rendimiento del método dependiendo del tamaño del objeto estático. Estas pruebas son realizadas en condiciones normales, es decir, con un modelo normal de la herramienta (cientos de polígonos).

En general, para herramientas pequeñas se tiene un buen rendimiento sin verse afectado por el tamaño de la maqueta estática, o en otras palabras, el método es escalable en el número de polígonos del objeto estático. Según las pruebas obtenidas y reflejadas en la Figura 6.10, se puede llegar a realizar operaciones de mantenimiento con modelos de hasta varios millones de triángulos sin que el módulo de Colisiones tarde más de 9-10 ms.

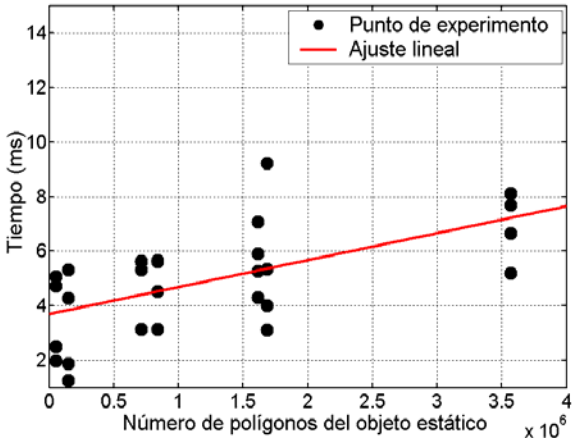


Figura 6.10: Rendimiento del cálculo de las colisiones en función del número de triángulos del objeto estático. Los modelos estático-móvil son modelos utilizados en las simulaciones de mantenibilidad.

Sin embargo, sí que se nota cambio al aumentar el orden de magnitud del objeto móvil, normal por otro lado porque el algoritmo recorre toda la estructura geométrica de la herramienta. Por ello, y para observar el comportamiento en las situaciones más desfavorables, la Figura 6.11 se construye en función del tamaño del objeto móvil introduciendo dos herramientas con órdenes de magnitud mayores a lo acostumbrado a manejar.

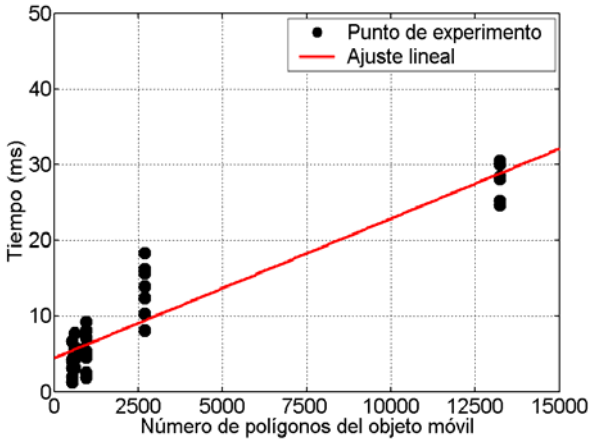


Figura 6.11: Rendimiento del cálculo de las colisiones en función del número de triángulos del objeto móvil. Los modelos estático-móvil son modelos utilizados en las simulaciones de mantenibilidad.

El conjunto de experimentos se puede ajustar con una ecuación lineal mediante el método de los mínimos cuadrados obteniéndose la recta roja dibujada en las gráficas. Este ajuste puede dar una estimación del comportamiento del método dependiendo del tamaño de los modelos.

En la Figura 6.11 la pendiente del ajuste lineal es mayor de lo que se deduce una mayor dependencia del objeto móvil. Sin embargo esto no es un inconveniente, al contrario, es preferible tener mayor dependencia de la herramienta que de la maqueta. Si se tuviera un método muy dependiente del objeto estático sería más peligroso ya que se tiene menos poder sobre el grado de precisión con que se modela estos objetos masivos. Sin embargo, una herramienta es un objeto sencillo y fácil de modelar a la precisión que uno quiera (incluso bajar esa precisión si se requiere mayor rendimiento).

De todas maneras, el modelado de herramientas con unos cientos de polígonos es suficiente para obtener la precisión requerida en las simulaciones.

6.3.1.2 Estudio de la memoria consumida

El otro factor muy dependiente del método es la memoria consumida. En la Figura 6.12 aparece la memoria consumida por los dos métodos empleados tal y como se explicaba en el apartado 3.2.1.2. En ella se ilustra tanto el método original como la memoria consumida en el caso de utilizar las técnicas de *hashing*. Se han realizado los experimentos con un nivel del 70% para el *hashing* ya que se comprobaba que éste era un límite apropiado sin que el rendimiento del algoritmo se viera afectado. Esto quiere decir que se logra ahorrar hasta un 70% de la memoria utilizada por los voxels.

De sobra conocido, los métodos de voxels tienen la desventaja de su gran consumo de memoria como puede verse en la Figura 6.12. Es por ello que se necesitan implementar técnicas para minimizar ese consumo. El método presentado utiliza el *hashing* para mejorar ese consumo por su bajo coste computacional en el algoritmo.

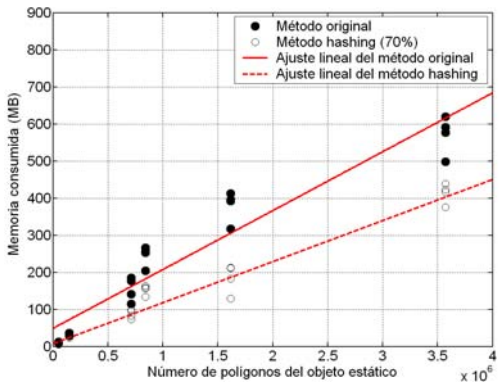


Figura 6.12: Memoria consumida en diferentes experimentos con el método original y usando técnicas de *hashing.*

Otros autores (McNeely et al. 1999) utilizan híbridos entre voxels y octrees para disminuir aún más el consumo de memoria pero eso implica una pequeña búsqueda en una estructura jerárquica lo que puede repercutir en el rendimiento. Estos mismos autores realizan una comparativa de memoria[4] entre su método VPS (*Voxmap PointShell*) ofrecido por Boeing y el algoritmo

[4] Página Web de Boeing con una comparación de memoria entre VPS y V-Collide: http://www.boeing.com/phantom/vps/extra/memory.html.

utilizado en el sistema libre V-Collide (Hudson et al. 1997). Comparando este estudio con la Figura 6.12, en lo que respecta a memoria consumida el método propuesto en esta Tesis se colocaría por debajo del V-Collide pero consumiría más memoria que el método ofrecido por la compañía Boeing. Por ejemplo, los datos para un modelo de 3 millones de triángulos serían 140 MB necesarios con el VPS y 640 MB con el V-Collide, frente a los aproximadamente 340 MB del método propuesto con *hashing* del 70%.

De todas maneras, hay que dejar claro que desde el principio se ha dado mayor importancia al rendimiento del algoritmo que a la memoria consumida por él, una de las razones porque se disponía de memoria suficiente.

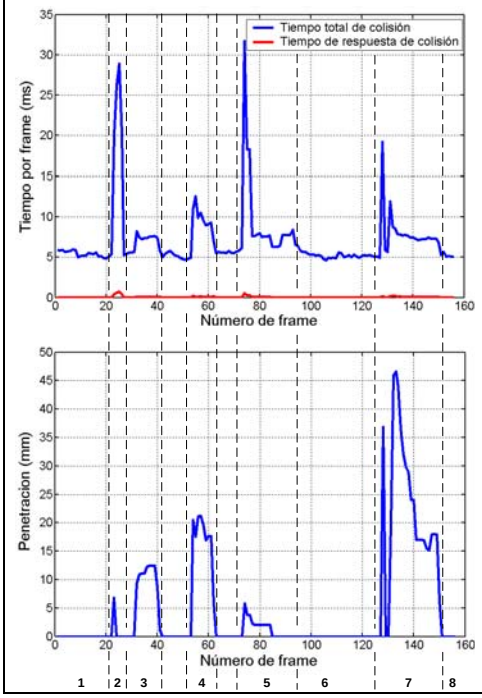
6.3.2 Experimentos con tareas de mantenimiento

Una vez visto cómo se comporta el algoritmo de colisiones (tiempo y memoria consumida) en función del tamaño de la escena, otro tipo de experimento interesante es observar lo que pasa a lo largo del tiempo. En concreto, y sin olvidar que todo el estudio tiene como objetivo final la utilización del prototipo en casos prácticos, se va a mostrar los resultados experimentales obtenidos durante la realización de tres tareas típicas de mantenimiento. Estas tareas son el desmontaje de tornillos, tuercas y tubos y se describen en los siguientes apartados.

6.3.2.1 Desmontaje de tornillo

En cualquier tarea de mantenimiento, primero hay un acercamiento del usuario a la zona de trabajo para posteriormente empezar a interactuar con el elemento en cuestión. Esto mismo se muestra en la Figura 6.13 dónde se detalla una tarea de desensamblaje completa.

El primer paso como se ha dicho es la aproximación al elemento objetivo. Una vez seleccionado el elemento se procede a su desmontaje hasta su total liberación, momento en el que se analizan también las colisiones de ese elemento con su entorno. Todo ello aparece en la Figura 6.13 en la que el tiempo ha sido dividido en dos fases: la detección de las colisiones y el cálculo de la respuesta de colisión. Asimismo, también se muestran las penetraciones que se han ido calculando a lo largo del tiempo.



Leyenda

1.- Aproximación de la herramienta al modelo.
2.- Contacto con el entorno.
3.- Ídem del anterior.
4.- Primer contacto con el elemento.
5.- Contacto y selección del elemento.
6.- Extracción del elemento (operación de desensamblaje).
7.- Contacto del elemento con el entorno.
8.- Liberación del elemento.

Figura 6.13: Simulación del desmontaje de un tornillo con el tiempo empleado en las colisiones y la penetración calculada entre los objetos estático-móvil.

Esta tarea ha sido realizada utilizando el modelo estático "Motor" (1.6 millones de polígonos) y la herramienta "Mano-h0" (2500 polígonos) que además de la mano incluye la herramienta adecuada para la operación.

Una rápida conclusión es el bajo coste computacional que tiene el cálculo de la respuesta de colisión (inferior a 1 ms). La mayor parte del tiempo es utilizado en la fase de la detección de colisiones.

El coste computacional cuando no existe colisión es de 5 ms aproximadamente. Esto puede parecer excesivo pero es una penalización que se obtiene con el algoritmo propuesto. Este método está optimizado para el estado de colisión el cual se calcula en bastante menos de los 100 ms que se habían impuesto como primer objetivo.

En la simulación se toca varias veces el entorno con la herramienta pero hay que fijarse en el momento en el que se coge el elemento (punto 5). Al principio hay un pico de unos 30 ms de cálculo pero después, cuando el usuario estabiliza la herramienta y la alinea para coger el elemento, las colisiones sólo tardan 2 ó 3 ms más del estado en el que no existe intersección alguna. Y la

distancia penetrada también es mínima (2.5 mm aproximadamente). Esta es la situación ideal y a la que llegaría un operario con la experiencia suficiente para no colisionar con el entorno salvo en los casos inevitables. En esta simulación se ha tocado el entorno varias veces de forma arbitraria para comprobar el funcionamiento del método con usuarios no tan experimentados, de ahí los picos que aparecen en las gráficas.

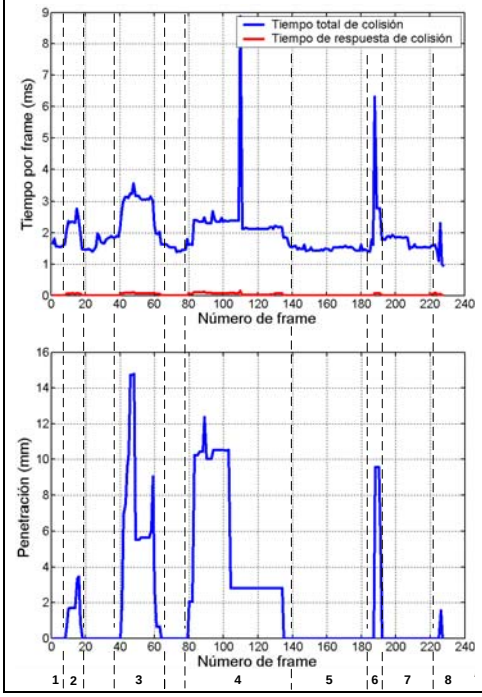
En el punto 6 no existe colisión ya que mientras se desmonta un elemento se desactivan las colisiones entre ese elemento y los demás objetos. Lo que si está activo, y aparecería en el gráfico si se hubiera dado el caso, es la detección de contacto entre la herramienta y el entorno mientras dura la operación. Esto último es de gran ayuda al operario para detectar si será posible desmontar un elemento con una herramienta dada.

Una vez acabada la operación, es posible y necesario detectar las colisiones de ese elemento con su entorno. El punto 7 refleja este tipo de intersecciones. La liberación del elemento (punto 8) consiste en soltarlo con lo que desaparecería de la escena y se volvería al estado carente de intersecciones.

6.3.2.2 Desmontaje de tuerca

El siguiente ejemplo consiste en el desmontaje de un elemento que está ligado siempre a un tubo. Normalmente se tienen estos elementos para enganchar los tubos a la carcasa del motor. Si se quiere desmontar algún tubo, antes habrá que desmontar las tuercas que lo mantienen sujeto. Este desmontaje únicamente consiste en deslizar la tuerca por el tubo una cierta longitud.

Este ejemplo tiene como objetivo comprobar el buen funcionamiento del háptico una vez se tiene cogida una tuerca. Este buen comportamiento pasa por detectar los finales de carrera del tubo, es decir, detectar cuando la tuerca llega a uno de los extremos del tubo. Esto significa que hace falta detectar la colisión del elemento con el resto de objetos que puedan existir a lo largo de ese tubo. La Figura 6.14 describe la operación que simula este desmontaje.



Leyenda

1.- Aproximación de la herramienta al modelo.
2.- Primer contacto con el elemento.
3.- Contacto y selección del elemento.
4.- Colisión en un extremo del tubo.
5.- Movimiento a lo largo del tubo.
6.- Colisión en el otro extremo del tubo.
7.- Liberación del elemento.
8.- Colisión de la herramienta al salir de la zona de trabajo.

Figura 6.14: Simulación del desmontaje de una tuerca con el tiempo empleado en las colisiones y la penetración calculada entre los objetos estático-móvil.

En esta simulación se ha usado un modelo estático de 600000 polígonos y la herramienta "Spanner" con 606 polígonos.

Lo único destacable es el contacto con los dos extremos del tubo, reflejados en los puntos 4 y 6 respectivamente. En el primero de ellos se mantiene el háptico en el contacto un cierto tiempo para comprobar su estabilidad ante colisiones permanentes, mientras que en el segundo simplemente se toca y se rehuye la zona de colisión. La liberación del elemento consiste en soltarlo y dejarlo en el propio tubo (pero ya fuera de su ubicación original).

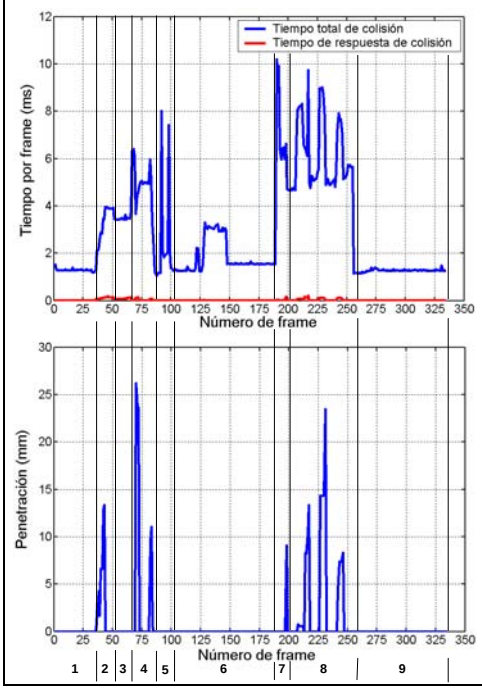
6.3.2.3 Desmontaje de tubo

El tubo es uno de los elementos más problemáticos en una operación de desensamblaje. Las colisiones que pueden generar una vez soltados de su enganche son múltiples debido a la longitud y diferentes curvaturas que poseen este tipo de objetos. Esta serie de colisiones puede provocar en algunos casos cambios bruscos de dirección de la fuerza con lo que se sentiría una especie de "rebote" mientras se está sacando el tubo de la zona de trabajo.

La sensación táctil puede no ser tan buena como en los otros dos ejemplos pero es debido a la limitación del sistema en los grados de libertad actuados. La ausencia de pares hace que el usuario sólo pueda sentir una única fuerza direccional.

En esta simulación en realidad se desensamblan dos objetos. Primero se desmonta una tuerca que actúa como restricción de ensamblaje para el tubo y por lo tanto tiene que ser desmontada previamente, y segundo el propio tubo.

En la Figura 6.15 se puede apreciar los tiempos y penetraciones obtenidos en esta tarea de mantenimiento.



Leyenda

1.- Aproximación de la herramienta al modelo.
2.- Selección de la tuerca.
3.- Movimiento de la tuerca a lo largo del tubo para su desensamblaje.
4.- Colisión de la herramienta con el entorno en la operación de desensamblaje de la tuerca.
5.- Liberación de la tuerca y colisión de la herramienta con el entorno.
6.- Aproximación de la herramienta al tubo.
7.- Selección del tubo.
8.- Múltiples contactos del tubo con su entorno.
9.- Liberación del tubo.

Figura 6.15: Simulación del desmontaje de un tubo con el tiempo empleado en las colisiones y la penetración calculada entre los objetos estático-móvil.

El objeto estático es el mismo que el utilizado en el ejemplo anterior, pero el modelo de herramienta en este caso es una mano que consta de 680 polígonos.

El punto 2 refleja el contacto y selección de la tuerca y en el punto 3 se desliza esa tuerca a lo largo del tubo para su desmontaje. Una vez libre de restricciones, el tubo se puede desmontar (punto 7). La serie de contactos sucesivos reflejados en el punto 8 son debidos a lo mencionado anteriormente.

Una vez suelto el tubo son detectadas múltiples colisiones hasta que se logra alejar de la zona.

La sensación táctil es todo lo buena que se puede obtener en un sistema sin momentos gracias a la combinación de algoritmos por parte de los módulos de Colisión y Control expuestos en el capítulo 5.

6.4 ÚLTIMAS CONSIDERACIONES

En este capítulo se ha descrito por un lado la arquitectura del sistema global y por otro lado se han presentado diferentes experimentos realizados con esa arquitectura.

Los experimentos se han dividido en dos. Primero se han realizado los estudios sobre los dos parámetros referenciados siempre en esta Tesis, el rendimiento y la memoria consumida. En segundo lugar se presentan una serie de simulaciones típicas de mantenimiento y el funcionamiento del sistema a lo largo del tiempo.

Gracias a estos experimentos se ha comprobado dos hechos muy positivos de cara a un mayor realismo de las simulaciones. Por supuesto, el hecho de contar con un dispositivo háptico ha hecho que estas tareas se realicen de una forma muy parecida a la real y sintiendo las sensaciones que se obtendrían en un mantenimiento sobre un motor real. Estas sensaciones están limitadas por la falta de los últimos 3 grados de libertad actuados en el sistema. El otro punto a favor es el contar con un sistema 3D que ofrezca visión estereoscópica al usuario. Es una necesidad que no se puede pasar por alto si se quiere realizar simulaciones lo más cercanas a la realidad.

6.5 REFERENCIAS

Amundarain, A., Borro, D., García-Alonso, A., Gil, J.J., Matey, L. y Savall, J., "Virtual Reality for Aircraft Engines Maintainability", *Proceedings of Virtual Concept 2002*, pp. 60-65. Biarritz, Francia. 9-10 Octubre 2002.

Gregory, A., Lin, M.C., Gottschalk, S., y Taylor, R., "Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback", *Computational Geometry – Theory & Applications 15*, pp. 269-285. 2000.

Hudson, T.C., Lin, M.C., Cohen, J., Gottschalk, S. y Manocha, D., "V-COLLIDE: Accelerated Collision Detection for VRML", *Proceedings of the second symposium on Virtual Reality Modeling Language (VRML'97)*, pp. 119-125. 1997.

Mark, W.R., Randolph, S.C., Finch, M., Van Verth, J.M., y Taylor II, R.M., "Adding Force Feedback to Graphics Systems: Issues and Solutions", *Proceeding of the ACM SIGGRAPH'96 - Computer Graphics*, New Orleans, Louisiana, pp. 447-452. Agosto 1996.

McNeely, W.A., Puterbaugh, K.D., y Troy, J.J., "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", *Proceedings of the ACM SIGGRAPH'99 - Computer Graphics,* Los Angeles, California, USA, pp. 401-408. Agosto 1999.

Savall, J., Borro, D., Gil, J.J. y Matey, L., "Description of a Haptic System for Virtual Maintainability in Aeronautics", *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2887-2892, EPFL, Lausanne, Switzerland. Septiembre (30) – Octubre (4) 2002.

CAPÍTULO 7

CONCLUSIONES Y FUTURAS LÍNEAS DE INVESTIGACIÓN

7.1 CONCLUSIONES

El objetivo de la presente Tesis ha sido el estudio del problema de las colisiones para su aplicación dentro de sistemas que puedan simular tareas de mantenimiento en cualquier modelo de maquinaria. Como consecuencia de la investigación realizada para llevar a cabo dicha tarea, se han realizado una serie de contribuciones al campo de las colisiones geométricas que se presentan a lo largo de los siguientes párrafos.

- Se ha redactado una exhaustiva clasificación de problemas tipo dentro del campo de las colisiones. Esta clasificación distingue, enumera, y describe los posibles factores que influyen en el problema de las intersecciones. Como ya se conoce, cada método existente depende del tipo de problema a resolver. Gracias a este estudio, es posible identificar las características de ese problema y localizar los trabajos ya realizados en esa área.

- Se ha realizado un estudio teórico y experimental sobre dos métodos muy usados en la bibliografía: la enumeración espacial y la subdivisión jerárquica. Este estudio ha sido validado con una comparativa entre cuatro métodos implementados en esta Tesis: voxels, octrees y dos métodos híbridos que combinan los dos

anteriores. Este estudio asienta las bases para el desarrollo del método propuesto.

- El método final propuesto en esta Tesis mejora los estudios realizados en el anterior punto en el aspecto del coste computacional. Este método introduce una pequeña mejora en la *narrow phase* que lo distingue de anteriores trabajos con este tipo de particionamiento. La principal diferencia con la gran mayoría de trabajos previos es su aplicación en escenarios compactos y masivos de millones de polígonos alcanzando la máxima precisión geométrica disponible. Aún en estas condiciones, el *frame rate* de las colisiones supera con creces los primeros objetivos impuestos de 10 Hz.

- La desventaja del uso de la enumeración espacial es el desaprovechamiento de zonas espaciales vacías que se traduce en un gasto de memoria innecesario. Por ello, se ha implementado una técnica de *hashing* cuya función *Hash* ofrece tan buenos resultados que logra ahorrar un 70% de la memoria de voxels sin repercusión significativa en el rendimiento.

- Se ha analizado con detalle el otro gran problema existente en el uso de técnicas enumerativas: la elección del tamaño del voxel. Como resultado se ha obtenido una solución analítica que encuentra una aproximación al tamaño óptimo de voxel tomando como entrada el par de modelos que representan a la maqueta estática y a la herramienta. Esta solución ha sido validada con un conjunto numeroso de experimentos en los que se comprueba el buen comportamiento de la estimación.

8 El sistema necesita dar una salida al exterior a través de un dispositivo de reflexión de fuerza por lo que ha sido necesario atacar el problema de la respuesta de colisión. Ésto ha derivado en una serie de algoritmos para el cálculo de la normal de contacto y la penetración entre dos poliedros generales. Se demuestra que apenas introduce carga computacional en el rendimiento global de las colisiones.

- Ligado a lo anterior, se ha estudiado la integración de estos algoritmos junto con los del módulo de Control para el cálculo final de las fuerzas a restituir. El sistema se ha usado en el periodo enero-abril 2003 a razón de 22 horas a la semana aproximadamente y según sus usuarios la sensación táctil conseguida gracias a los algoritmos es

suave, estable y sin fuerzas bruscas. Esta dedicación concierne tanto a la utilización del sistema con la restitución de fuerzas como a su uso con un 6-DOF SpaceMouse.

- El módulo de Colisiones se ha integrado dentro de un sistema multidisciplinar llamado REVIMA. El sistema global logra simular, mediante técnicas de realidad virtual, tareas de mantenimiento y ensamblaje en modelos de motores aeronáuticos. El módulo de Colisiones se comporta de forma interactiva en todas las simulaciones, incluso a la hora de montar y desmontar elementos de las maquetas virtuales.

- El algoritmo sólo depende de la geometría de la escena por lo que el código implementado ofrece un alto grado de modularización que se ha aprovechado para integrarlo en otros proyectos.

- Los resultados experimentales han sido satisfactorios, comprobándose que en tareas normales de mantenimiento y gracias a la restitución de esfuerzos que evita grandes penetraciones, el *frame rate* de las colisiones se mantiene entre 30 y 500 Hz. Asimismo, como validación se han presentado varios ejemplos típicos de mantenimiento y que afrontan diferentes dificultadas cada uno de ellos: el desensamblaje de un tornillo, una tuerca enroscada en un tubo y uno de los elementos más problemáticos como puede ser un tubo.

- Gracias también a los experimentos, se ha podido comprobar que el método propuesto es escalable en el tamaño de las maquetas virtuales, lo que quiere decir que apenas se ve afectado por el número de polígonos. El único condicionante lo supone la memoria RAM disponible. Se han analizado modelos de hasta 3.6 millones de triángulos sin agotar la memoria disponible (2 GB).

- El sistema REVIMA ha sido implantado en la empresa aeronáutica ITP (Industria de Turbopropulsores) con buena aceptación por parte de operarios y diseñadores quienes están utilizando el sistema para diferentes metas:

  - Validación de los modelos de CAD generados.

  - Formación de los operarios para las tareas de mantenimiento.

  - Estudios de ergonomía, secuencias de ensamblaje/desensamblaje y tiempos en las tareas de accesibilidad.

- Finalmente, los diferentes estudios realizados a lo largo de esta Tesis han dado pie a la publicación de varios artículos y al envío de otros pendientes de aceptación.

8.1 FUTURAS LÍNEAS DE INVESTIGACIÓN

Como continuación del trabajo descrito en esta Tesis, en los siguientes párrafos se proponen algunas de las posibles líneas en las que se podría seguir investigando:

- El método propuesto depende en gran medida del número de polígonos del objeto móvil. Sería interesante estudiar algún método alternativo que, manteniendo el acceso rápido conseguido con los voxels, permita discriminar rápidamente zonas de la herramienta que están alejadas del contacto.

- Se ha comprobado que para un consumo de memoria similar, la geometría del voxel no influye en el rendimiento del método. Sin embargo se podrían realizar más experimentos para averiguar el límite del ratio entre los lados del paralelepípedo.

- El consumo de memoria sigue siendo una desventaja a pesar del ahorro considerable utilizando las técnicas de *hashing*. Una aportación clara es la búsqueda de nuevas técnicas para reducir aún más ese consumo. Esto puede ir por la línea de la mejora de lo existente o por el contrario implementar algún método híbrido que combine la enumeración espacial con alguna otra técnica.

- Se puede seguir investigando y realizando experimentos con la fórmula de la función de coste z para seguir validándola. El parámetro que más le afecta es el área de los polígonos de los modelos extremadamente simplificados. Aunque son *worst-cases* para la búsqueda del óptimo, de cara al rendimiento terminan siendo *best-cases* por lo que investigar en esta línea beneficiaría únicamente en la generalización de la fórmula para todos los posibles casos.

- En un futuro se pretende añadir restitución de pares al sistema, es decir, añadir los 3 grados de libertad actuados que le faltan al

dispositivo (investigación actualmente en curso) por lo que habría que diseñar nuevos algoritmos en la respuesta de colisión que contemplaran este añadido.

- El sistema REVIMA utiliza una arquitectura dual para el PC de simulación. Resultaría más conveniente separar en dos PCs el módulo de Colisiones y el resto de la aplicación. El beneficio sería doble: por un lado el coste de dos PCs sería menor al PC dual actual (menos procesadores y menos memoria en cada PC). Por otro lado se tendría un módulo de Colisiones totalmente independiente que gracias a la interfaz adecuada se podría añadir en paralelo a cualquier otro sistema (pensando sobre todo en sistemas de RV o en simuladores).

- Las técnicas de enumeración espacial tienen alto grado de paralelismo. Otra posible línea sería paralelizar el módulo de Colisiones en previsión de tener algún día varios procesadores dedicados exclusivamente a este cómputo.

ANEXO A

CÁLCULO DEL MÍNIMO DE LA FUNCIÓN DE COSTE

Se parte de la función de coste z descrita en (A.1) que modela el comportamiento del algoritmo de colisiones.

$$z = n_m \cdot p_{nc} \cdot v_o + n_m \cdot (1 - p_{nc}) \cdot v_o \cdot f_e \cdot r_t \qquad \text{(A.1)}$$

Donde n_m es el número de polígonos del objeto móvil, p_{nc} es la probabilidad de que un triángulo del objeto móvil no esté en colisión con algún voxel lleno de geometría y r_t es el ratio entre el tiempo de testeo triángulo-triangulo y el tiempo de verificación de voxel.

Los parámetros v_o (voxels objetivo) y f_e (número de facetas por voxel), tanto para los voxels cúbicos como para los paralelepípedos (denotados por los superíndices *c* y *p* respectivamente), vienen definidos en las ecuaciones (A.2) y (A.3).

$$v_o^c = \left(\frac{a_m}{\delta} + 2 \right)^3, \quad v_o^p = \left(\frac{3Na_m}{X+Y+Z} + 2 \right)^3 \qquad \text{(A.2)}$$

$$f_e^c = \frac{\delta^2 \sqrt{3}}{2A_e}, \quad f_e^p = \frac{\sqrt{(XY)^2 + (XZ)^2 + (YZ)^2}}{2A_e N^2} \qquad \text{(A.3)}$$

Donde a_m es la arista media de los triángulos del objeto móvil, A_e es el área media de los triángulos del objeto estático y *X*,*Y*,*Z* son las dimensiones del volumen contenedor del objeto estático.

Para calcular el mínimo de la función de coste *z* hay que derivar primero dicha función respecto de la variable dependiente, δ o *N* según el caso. Pero antes se pueden realizar algunas simplificaciones. Primero, el factor n_m es una constante que multiplica a los dos sumandos por lo tanto no afecta a la hora de buscar el mínimo de la función. Segundo, según el heurístico encontrado, esta función trabaja bien cuando p_{nc}=0.99 y r_t=100 por lo que el factor $(1-p_{nc})r_t$ es la unidad. La función z a minimizar queda entonces expresada por la ecuación (A.4).

$$z = 0.99 \cdot v_o + v_o \cdot f_e \tag{A.4}$$

Sustituyendo v_o y f_e en la ecuación (A.4), las ecuaciones finales (A.5) y (A.6) representan a las funciones de coste para cada una de las dos geometrías distintas de voxel.

$$z_c = 0.99 \cdot \left(\frac{a_m}{\delta} + 2\right)^3 + \frac{\left(\frac{a_m}{\delta} + 2\right)^3 \delta^2 \sqrt{3}}{2A_e} \tag{A.5}$$

$$z_p = 0.99 \cdot \left(\frac{3Na_m}{X+Y+Z} + 2\right)^3 + \frac{\left(\frac{3Na_m}{X+Y+Z} + 2\right)^3 \sqrt{X^2Y^2 + X^2Z^2 + Y^2Z^2}}{2N^2A_e} \tag{A.6}$$

Las derivadas respectivas de cada una de estas funciones se presentan a continuación.

$$\dot{z}_c = \frac{2.97\left(\frac{a_m}{\delta} + 2\right)^3 a_m}{\delta^2} - \frac{3\left(\frac{a_m}{\delta} + 2\right)^2 a_m \sqrt{3}}{2A_e} + \frac{\left(\frac{a_m}{\delta} + 2\right)^3 \delta\sqrt{3}}{A_e} \tag{A.7}$$

$$\dot{z}_p = \frac{8.91\left(\frac{3Na_m}{X+Y+Z} + 2\right)^2 a_m}{X+Y+Z} - \frac{\left(\frac{3Na_m}{X+Y+Z} + 2\right)^3 \sqrt{X^2Y^2 + X^2Z^2 + Y^2Z^2}}{N^3A_e} + \frac{9\left(\frac{3Na_m}{X+Y+Z} + 2\right)^2 a_m \sqrt{X^2Y^2 + X^2Z^2 + Y^2Z^2}}{2N^2A_e(X+Y+Z)} \tag{A.8}$$

Si se resuelven estas dos ecuaciones para las incógnitas δ y N, la expresión que calcula el nivel óptimo de voxelizado viene dada por (A.9) (únicamente se da la solución cúbica debido al considerable tamaño que presenta la solución paralelepípeda).

$$\begin{aligned}\delta &= 0.083 \cdot a_m \\ &+0.017\left(92595.436 \cdot a_m A_e + 125 \cdot a_m^3 + 90 \cdot \sqrt{105858 \cdot a_m^2 A_e^2 + 2857.884 \cdot a_m^4 A_e}\right)^{(1/3)} \\ &+\frac{0.417 \cdot a_m^2}{\left(92595.436 \cdot a_m A_e + 125 \cdot a_m^3 + 90 \cdot \sqrt{105858 \cdot a_m^2 A_e^2 + 2857.884 \cdot a_m^4 A_e}\right)^{(1/3)}}\end{aligned} \tag{A.9}$$

ANEXO B

ARTÍCULOS GENERADOS

En este anexo se incluyen los artículos generados hasta el momento gracias al estudio e investigación de la presente Tesis (se citan tanto los artículos aceptados como los pendientes de aceptación). A continuación se enumeran estos artículos y en las siguientes páginas se presentan los artículos aceptados en su formato final.

Borro, D., Amundarain, A., Gil, J.J., Savall, J., García-Alonso, A., y Matey, L., "Tracking System with Force Feedback for Aircraft Engines Maintainability", enviado a *IEEE Computer Graphics & Applications* en abril de 2002.

Savall, J., Borro, D., Gil, J.J. y Matey, L., "Description of a Haptic System for Virtual Maintainability in Aeronautics", *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2887-2892, EPFL, Lausanne, Suiza. Septiembre (30) - Octubre (4), 2002.

Amundarain, A., Borro, D., García-Alonso, A., Gil, J.J., Matey, L. y Savall, J., "Virtual Reality for Aircraft Engines Maintainability", *Proceedings of Virtual Concept 2002*, pp. 60-65. Biarritz, Francia. 9-10 Octubre 2002. (Este mismo artículo ha sido aceptado en la revista *Mécaniques et Industries* 2003).

Amundarain, A., Aguinaga, I., Borro, D., García-Alonso, A., y Matey, L., "Realidad virtual para el mantenimiento de motores de avión", *Proceedings of the CEIG'03*. 2003.

Borro, D., Gil, J.J., García-Alonso, A., y Matey, L., "Haptic Rendering for new Large Haptic Devices", enviado a *ACM Transactions on Graphics* en abril de 2003.

Borro, D., García-Alonso, A., y Matey, L., "Approximation of Optimal Voxel Size for Collision Detection in Maintainability Simulations within Massive Virtual Environments", enviado a *Computer Graphics Forum* en mayo de 2003.

Proceedings of the 2002 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
EPFL, Lausanne, Switzerland • October 2002

Description of a Haptic System for Virtual Maintainability in Aeronautics

Joan Savall[1], Diego Borro[1], Jorge J. Gil[2], Luis Matey[1]

[1]CEIT, San Sebastián, Spain, jsavall@ceit.es, dborro@ceit.es, lmatey@ceit.es
[2]University of Navarra, San Sebastián, Spain, jjgil@tecnun.es

Abstract

This paper describes a Haptic system for maintainability simulation in Aeronautics, called REVIMA (Virtual Reality for Maintainability). In this project a software-hardware tool is designed and built to realistically simulate assembly-disassembly operations. It also helps to perform accessibility, interference and maintainability analysis by using virtual reality techniques without physical mock-ups. The system gives the user a reliable and realistic response. In order to achieve these requirements, the device has a workspace similar to the size of a turbo-engine. In addition this workspace can be placed in different positions to study ergonomics aspects of the simulated tasks.

1. Introduction

In the field of Aeronautics the term Maintainability is defined as "the ability of an element to keep in service or to be returned to adequate status in order to develop its function, after being maintained at conditions previously established, using the personnel, the means and adequate procedures"[1].

One of the most relevant aspects of maintainability concerns man and tool accessibility task analysis, which is undertaken in order to calculate paths and assembly-disassembly sequences and times.

Design based on electronic mock-up is widely used in the creation of engine externals (piping, harnesses and installations) by the aeronautics industry. Pipes and harness are routed over these parts and accessories are installed by means of a workstation network. This allows a group of designers to work quasi-concurrently over an assembly, copying and automating the original process. This technology is known in the industry as DMU/DPA (Digital Mock Up / Digital Pre-Assembly).

DPA/DMU technology has overcome the need for a hard mock-up for design purposes, significantly decreasing time-to-market and thereby saving money. However, nowadays the use of a physical mock-up is mandatory in order to evaluate the maintainability of externals during the development stage. Although these mock-ups can be used for other applications, the ultimate purpose of the construction is to check the maintainability. The expenses of these mock-ups led ITP to research an alternative using haptics.

Figure 1: *Maintenance operation on an aircraft-engine*
(Photograph Courtesy of Rolls Royce)

ITP is the exclusive supplier of low-pressure turbines for Rolls-Royce engines of greater than 35,000lbs of thrust - primarily the Trent engine family. It is also the Spanish participant in the EJ200 engine for the Typhoon Eurofighter, and earlier this year became a 13.6% shareholder in the TP400 engine programme for the A400M European military transport aircraft.

Dressings and Maintainability design areas have always aspired to do away with -partially or totally- the physical mock-up, at least during the development phase. During production, the first production engine serves as a physical mock-up.

The main aim of this project is thus to develop a haptic device that can be used as a tool to predict the maintainability of an aircraft engine. One of the main advantages of this development would be that mock-ups were no longer needed for this purpose, leading to important cost savings in the development of a new engine.

0-7803-7398-7/02/$17.00 ©2002 IEEE

2. Description of the system

REVIMA is a haptic system developed to check the maintainability of aircraft engines. The system has been created from scratch by CEIT Applied Mechanical Department. This is a multidisciplinary development that includes, amongst others, the following disciplines: mechanical design, control theory, computer graphics, computational geometry and human-machine interaction.

The research involved in the project concerns two main areas: mechanical design and software development. Both of them deal with important challenges since system maintenance simulation needs to be very close to reality.

One of the main targets of the mechanical design was that the workspace of the device should match that of an aircraft engine. At the same time, any haptic device had to have low inertia. Both requirements have been achieved by combining mechanical design with significant sensible use of a force sensor. The need of large workspace was established by ITP to perform ergonomic studies. This design is explained in section 3.

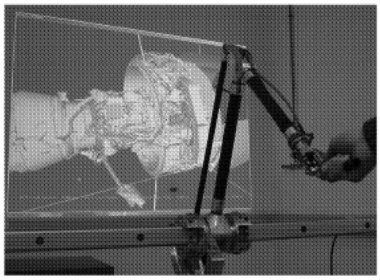


Figure 2: *Photo of the CEIT LHIfAM in a virtual maintainability operation on a CAD aircraft engine model*

In turn, software development has involved the integration of a fast control loop that reflects force to the operator, the evaluation of collisions, and the visualization of the scene. The two last tasks are especially difficult because of the enormous size of the model (more than 2 million triangles). Section 4 describes this integration.

3. Haptic interface

The Large Haptic Interface for Aeronautic Maintainability (LHIfAM) which has been developed, is an example of nonportable force-feedback hardware.

The most prevalent forms of force feedback interfaces in use today, are desk-grounded masters [2]. The system that this paper presents is floor-grounded due to the large workspace needed for the maintainability application. In fact, the basic workspace of the haptic interface occupies a cylindrical sector, which corresponds to a wide work area of a virtual 3D aircraft engine full-scale mock-up.

The dimensions of the basic cylindrical work-space are: internal radius, 242 mm; external radius, 772 mm; depth 1500mm; angle, 120º.

The system provides force feedback in three translational degrees of freedom while three additional orientations are measured, but not actuated, by a special wrist. The 6 DOF are obtained as follows:

The kinematic configuration of the three translational degrees of freedom is PRR, consisting of one prismatic and two revolute joints. A variant of a parallelogram mechanism is used, mounted on a commercial linear motion-rolling guide to obtain the largest displacement (about 1.5 metres).

A Roll-Pitch-Roll wrist provides the three orientations. These are measured by high-resolution encoders (2048 cpr). A classic design of the RPR wrist has been compacted to obtain a robust and light (less than 140gr) device to measure the orientation of the virtual tool.

The end-effector is cylindrical-shaped and can be held like a pen (for increased dexterity) or like a handle (for increased power). It is designed to be interchangeable between different shapes to reproduce several ways of holding operation tools.

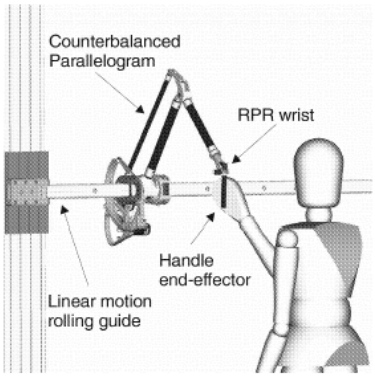


Figure 3: *Main parts of the haptic interface.*

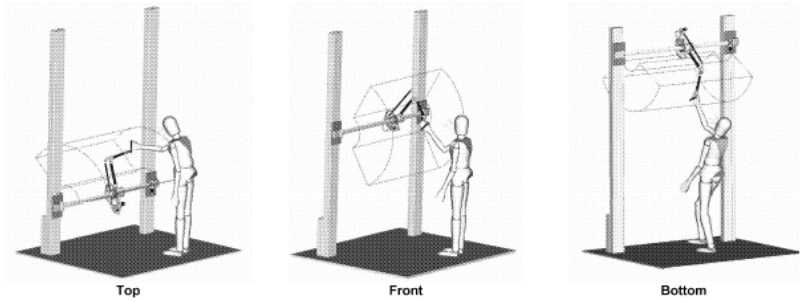


Figure 4: *Different locations of the workspace*

DC brushed motors with encoders transmit torques to the linkages through pre-tensioned cable reductions. To obtain the linear motion of the prismatic joint, a special cable transmission has been developed and tested, providing high performance.

The maximum providing continuous force capability of the device is about 16 N, while the limit of the peak force is about 70 N. The nominal resolution is 0.02 mm at the end effector.

An interesting design feature of the LHIfAM is that its workspace can be relocated: to reproduce different maintainability operations and check different situations from an ergonomic point of view, the basic cylindrical sector can be reconfigured as shown in Figure 4. The linear-motion-guide is supported by two columns, which allow the system to be placed at different heights, depending on the required maintainability task.

This is possible, without decreasing its dynamic properties, thanks to the special characteristic that the mechanism presents. The centre of gravity coincides with the pivotal point over the linear-guide for any position of the mechanism. To obtain this counterbalanced system, the DC motors are specially arranged on the parallelogram structure, which is also built using advanced structural materials.

4. System Architecture

The software of REVIMA is based on C++ and OpenGL Graphical Library. It has been developed using Microsoft Visual C++ and Microsoft Windows 2000 OS. This has represented a challenge because Windows 2000 is not a real time OS but it has advantages of cost, easy support and management, and software development tools.

The system runs in 2 PCs. One (control PC) is in charge of executing the control loop to command the LHIfAM. The other (simulation PC) runs the main module, the Graphical User Interface (GUI), the collision solver and the graphics engine. Both PCs are interconnected through an Ethernet LAN network using the UDP network protocol. This type of architecture is also used in [3] and [4].

The computer that controls the haptic interface is a 233 MHz Pentium II CPU. A dual processor (two 866 MHz Pentium III Xeons with an Intense3D Wildcat 4210) runs the simulation. Collision detection is executed in one dedicated processor of the simulation PC.

The scheme of the system architecture is presented in Figure 5.

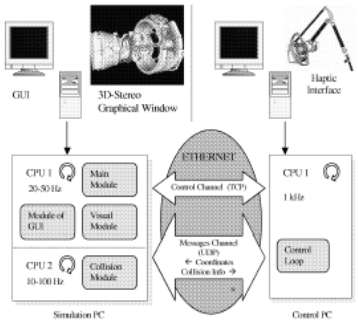


Figure 5: *System Architecture*

The simulation PC has two monitors: one shows the GUI and the other will contain the 3D virtual scene. The simulation PC's software has been structured in

independent modules encapsulated in DLLs to improve their development and support.

The software modules mentioned earlier are described below.

4.1 Simulation PC modules

The main module synchronizes several events: the GUI, the 3D scene visualization, collision detection and the reception of messages from the Control PC. This is achieved using a standard defined interface.

The visual engine needs to exceed 20 frames per second to achieve the application's requirements and obtain a good interaction with the user. This module uses several culling techniques and graphical database optimizations.

This module receives periodically, at a frequency of 1 kHz, the user position from the control PC. Since this frequency is higher than the visual module's rate, only the last position received is taken into account.

Due to the high frequency of the control loop, collision detection should have as fast a rate as possible. Efficiency has been preferred to saving memory. Therefore, algorithms have been chosen on the basis of spatial partitioning by means of voxels [5]. These algorithms use large amount of memory but are very fast (the access to the voxels is direct as against the search system of some hierarchical methods). In addition, the Boeing Company has experience in projects employing this kind of massive virtual environment [6].

For maintainability and accessibility, faithful reproduction/representation of the contact is critical. This module obtains good results because the structure stored reaches the triangle level, i.e. the maximum level accordi

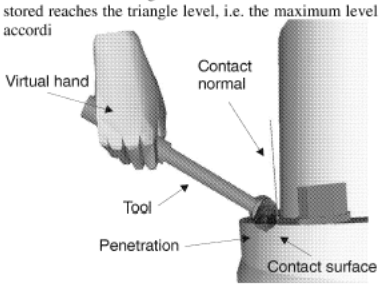


Figure 6: *Graphical representation of the collision information*

Another essential part of the collision module is the computation of the necessary information to obtain the most realistic force possible throughout the LHIfAM. All the computational effect is expended in the collision module and only minimal information for the control loop is sent off minimizing the network traffic.

This computational cost arises from calculating a direction or contact normal and a penetration. With these values, the LHIfAM can obtain a direction and magnitude for the feedback force.

Figure 6 shows a visual example of this computation.

Besides these two values, the simulation PC sends the user-position, employed to calculate the penetration, through the network to the control PC. This value is necessary for the algorithm of the control PC.

4.2 Communications

There are several reasons to choose the architecture of two PC's connected by Ethernet. The most obvious relates to efficiency. To control the LHIfAM, a control frequency of about 1 kHz is proposed. Since an application of this kind needs to run with real-time priority, it can have problems if other modules are running in the same machine at the same time.

Other reasons include:

- Reuse of the haptic in any application using the defined standard protocol.
- Disposition of a control PC to connect different devices to the application REVIMA.

As previously mentioned, the two PC's communicate by UDP. The disadvantage of this protocol compared with TCP is the lack of reliability of the messages received. In this kind of real time application this isn't a disadvantage because the loss of a packet is unusual in a point-to-point connection via Ethernet. The sender has a much higher frequency than the receiver, which takes the last packet, received.

These messages are normally are exchanged across the network. The system also has a second communication "control" channel, which it used for very specific control messages. This channel uses the TCP protocol because reliability is required rather than speed. Examples of this kind of message include: the remote execution of the control loop from the simulation PC, messages to take and release different pieces of assembly, notification from the control PC to the simulation PC of problems with the LHIfAM as the excessive heat of the motors, etc.

4.3 Control PC

The control loop, located in the control PC, has a sampling period of 1 kHz. According to the studies of Shimoga [7] the majority of authors choose one sample frequency of either 500 Hz or 1 kHz.

This control module acquires the position of the robot manipulator and sends that information to the simulation PC.

The collision forces are calculated according to the actual user-position, the last collision information received (note that the collision solver runs at a slower frequency) and the contact model in use. Figure 7 shows the parameters received from the simulation PC: the contact normal **n**, the penetration x and the tool position **p**.

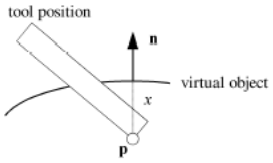


Figure 7: *Collision information*

The contact model used in REVIMA is a simple spring [8]. The collision force is directly proportional to the penetration of the virtual tool in the environment. The stiffness, K of the model is selected experimentally to be as high as possible, whilst maintaining stability. This stiffness is quite low (1 kN/m) because the workspace of the device is very large. Joint stiffness (in Nm/rad) is divided by the square of the arm length at the end-point (in N/m). A different model that includes virtual damping and that permits higher stiffness is currently being studied.

A friction model following based on that described by Salisbury et al. [9] is also implemented in REVIMA.

Since the control loop runs faster than the collision module, several strategies must be implemented in order to avoid brusque changes of the contact force. There are three problems that should be taken into account.

The first one is the delay that exists in the collision information. When a collision message arrives in the control PC, its information is only true for a previous position of the user, not the actual. Knowing the position of the user where that information is true, $\mathbf{p}_{collision}$, and assuming that the contact normal, $\mathbf{n}_{new}$, has no change, the variation of the collision penetration, Δx_{delay}, can be estimated by projecting the difference between actual and collision positions on the contact normal. This variation is added to the received collision penetration, $x_{collision}$, to estimate the actual penetration that must be taken into account, x_{new}.

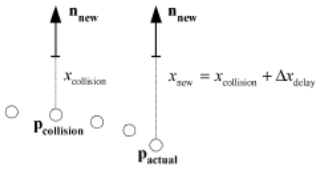


Figure 8:*Processing the collision information*

The second problem is what to do if there is no new collision information in a control iteration. This arises if the collision module is slower than the control loop. In this case, the contact normal is maintained and the penetration is modified by projecting the difference between actual and previous positions on the contact normal. This is represented in Figure 9.

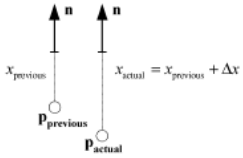


Figure 9: *Incrementing the penetration without collision information*

The last problem is how to solve the great changes in normal direction and penetration, which can appear every time that new collision information is received. Mark et al. [4] developed a method that calculates n intermediate penetrations, i.e. intermediate planes along normal direction in their algorithm- in order to restitute a smoother force along the n following sampling periods. In REVIMA a similar intermediate iteration is used, but also n intermediate normal directions are calculated, as is represented in Figure 10.

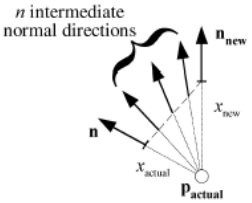


Figure 10: *Intermediate normal directions*

The combination of the strategies described in this section permits sufficiently stiff and smooth touch

simulation even in the presence of several contact points or great changes in the contact surface.

Another important factor that concerns the control loop is the fact that a motor is used in order to aid the user in his/her movement. The result of this strategy is a decrease in the apparent inertia that the user feels.

To achieve this decrease, the force exerted by the user is measured and filtered. The control loop restores a force K_f times greater, so the apparent inertia is decreased $1+K_f$ times. In REVIMA a reduction of 6 times the apparent inertia of the system has been implemented.

5. Conclusions

The main conclusion of this work is that haptics can be directly applied to the industry and that these systems may lead to important cost savings.

The system has shown that it is possible to have a device with a workspace as large as an aircraft engine with low apparent inertia. This matching of workspaces and the ability to relocate its spatial position are especially useful in performing ergonomic studies.

Another interesting characteristic of this system is that CAD models used in design phase are also useful for simulation studies without preprocessing.

The employed architecture of two different control loops (a slow one for collision detection and a fast one for force restitution) has proven to be a good approach to treat the problem of managing large CAD models in real time.

There are two completely developed systems that are being used and validated by ITP.

Future research includes the development of software and hardware for a haptic device with 6 actuated degrees of freedom.

6. Acknowledgments

The authors would like to thank to ITP and Sener for promoting and funding the application REVIMA. The development of this application has been partially financed by the Basque Government. (Project number CI01TP03)

We would like to especially thank Iker Aguinaga, Aiert Amundarain, Emilio Sánchez, Angel Rubio and Jaime Rubí for their support and ideas in the development of this work.

7. References

[1] B. Blanchard, S. D. Verma and E. L. Peterson: *Maintainability*, John Wiley & Sons Inc., New York, USA, 1995.

[2] T. H. Massie and J. K. Salisbury. "The phantom haptic interface: A device for probing virtual objects". *In ASME Winter Annual Meeting, Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, Chicago, IL, Nov. 1994.

[3] T. V. Thompson II, D.D. Nelson, E. Cohen and J.M.Hollerbach, "Maneuverable NURBS Models Within a Haptic Virtual Environment" *Proceedings of the 1997 ASME International Mechanical Engineering Congress and Exhibition,* Dallas, DSC-Vol 61, pp. 37-44, 1997.

[4] W. R. Mark, S. C. Randolph, M. Finch, J. M. Van Verth and R. M. Taylor II, "Adding Force Feedback to Graphics Systems: Issues and Solutions", *Proceeding of SIGGRAPH 96*, In Computer Graphics Proceedings, New Orleans, Louisiana, August, 1996, pp. 447-452.

[5] A. García-Alonso, N. Serrano and J. Flaquer. "Solving the Collision Detection Problem", *IEEE Computer graphics and applications*, Vol .13, nº 3, pp. 36-43, 1994.

[6] W. A. McNeely, K. D. Puterbaugh and J. J. Troy, "Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling", *SIGGRAPH - Computer Graphics Conference Proceedings*, pp. 401-408, 1999.

[7] K. Shimoga, "Finger Force and Touch Feedback Issues in Dextrous Telemanipulation", *Proceedings of NASA-CIRSSE International Conference on Intelligent Robotic Systems for Space Exploration*, NASA, Greenbelt, MD, September 1992.

[8] G. C. Burdea, *Force and Touch Feedback for Virtual Reality*, John Wiley and Sons, Inc., New York, 1996.

[9] K., Salisbury, D., Brock, T., Massie, N. Swarup and C., Zilles, "Haptic Rendering: Programming Touch Interaction with Virtual Objects" *Proceedings 1995 Symposium on Interactive 3D Graphics*, pp. 123-130, Monterey, California, 1995.

Virtual Concept 2002 *Biarritz - France* *October, 9-10*

Virtual Reality for aircraft engines maintainability

A. Amundarain[3], D. Borro[3], A. García-Alonso[2], J. J. Gil[3], L. Matey[1], J. Savall[3]

(1) : University of Navarra
Lardizábal 13, E-20018 San Sebastián
(34) 943 219877 / (34) 943 311442
E-mail : lmatey@tecnun.es

(2) : University of the Basque Country
Lardizábal 1, E-20018 San Sebastián
(34) 943 015104 / 943 219306
E-mail : agalonso@si.ehu.es

(3) : CEIT (Centro de Estudios e Investigaciones Técnicas de Guipúzcoa)
Lardizábal 15, E-20018 San Sebastián
(34) 943 212800 / (34) 943 213076
E-mail : {aamundarain, dborro, jjgil, jsavall}@ceit.es

Abstract : REVIMA is a virtual reality system for maintainability simulation in Aeronautics. It comprises both hardware and software developments, plus system integration. REVIMA required the design of a new haptic system. It is used both to track hand movements and to return force feedback that provides the sensation of working with a physical mock-up. The main software modules are: image generation, collision detection and control. System integration is based on two LAN connected PCs that share the different tasks and data.

The visualization module has been built using low-cost graphic systems, and we have thoroughly analyzed this problem to achieve a drawing frame rate acceptable for simulation analysis. The models comprise more than two thousand different elements that require about two million polygons to describe their shapes. Different organization strategies have been tested in order to achieve the real simulation goal. We also present the different visualization algorithms we have used.

Keywords : virtual reality, real time visualization, maintainability, haptic.

1- Introduction

In the field of Aeronautics the term Maintainability is defined as "the ability of an element to keep in service or to be returned to adequate status in order to develop its function, after being maintained at conditions previously established, using the personnel, the means and adequate procedures" [1].

One of the most relevant aspects of maintainability concerns man and tool accessibility task analysis, which is undertaken in order to calculate paths and assembly-disassembly sequences and times. Design based on electronic mock-up is widely used in the creation of engine externals (piping, harnesses and installations) by the aeronautics industry. Pipes and harness are routed over these parts and accessories are installed by means of a workstation network. This allows a group of designers to work quasi-concurrently over an assembly, copying and automating the original process. This technology is known in the industry as DMU/DPA (Digital Mock Up / Digital Pre-Assembly).

DPA/DMU technology has overcome the need for a hard mock-up for design purposes, significantly decreasing time-to-market and thereby saving money. However, nowadays the use of a physical mock-up is mandatory in order to evaluate the maintainability of externals during the development stage. Although these mock-ups can be used for other applications, the ultimate purpose of the construction is to check the maintainability. The expenses of these mock-ups led ITP (Industria de TurboPropulsores) to research an alternative using haptics. ITP is the exclusive supplier of low-pressure turbines for Rolls-Royce engines of greater than 35,000lbs of thrust - primarily the Trent engine family. It is also the Spanish participant in the EJ200 engine for the Typhoon Eurofighter, and earlier this year became a 13.6% shareholder in the TP400 engine programme for the A400M European military transport aircraft.

The tracking system provides a workspace similar to the size of a turbo-engine. The whole system gives the user a realistic feedback. Even more, as we will explain later on, the reachable workspace can easily be shifted to different positions and orientations. This allows the simulation of different relative positions between the model and the operator, to study ergonomic aspects of the simulated tasks.

Using our device the user movements are the same ones that are done when testing physical mock-ups. This fact provides an enhanced sense of real manipulation and can lead to important reduction in costs in the development of new aircraft engines.

2- Description of the system

REVIMA is a system developed to check the maintainability of aircraft engines. The system has been created from scratch by CEIT Applied Mechanical Department. This is a multidisciplinary development that includes, amongst others, the following disciplines: mechanical design, control theory, computer graphics, computational geometry and human-machine interaction.

The research involved in the project concerns two main areas: mechanical design and software development. Both of them deal with important challenges since system maintenance simulation needs to be very close to reality.

One of the main targets of the mechanical design was that the workspace of the haptic device should match that of an aircraft engine (see Fig. 1). At the same time, any haptic device had to have low inertia. Both requirements have been achieved by combining mechanical design with significant sensible use of a force sensor. The need of large workspace was established by ITP to perform ergonomic studies.

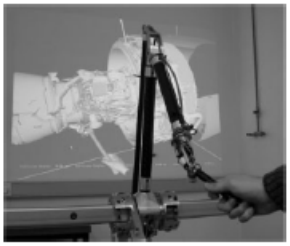


Fig. 1 : Photo of the CEIT haptic in a virtual maintainability operation on a CAD aircraft engine model

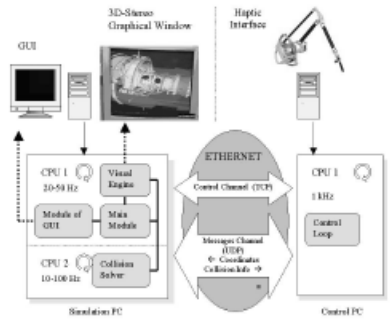
In turn, software development has involved the integration of a fast control loop that reflects force to the operator, the evaluation of collisions, and the visualization of the scene. The two last tasks are especially difficult because of the enormous size of the model (more than 2 million triangles). Section 3 describes this integration.

3- System architecture

REVIMA is a complex project that involves software (computer graphics, geometry computational, control) and hardware (mechanics) disciplines. The software has been developed using Microsoft Visual C++ and OpenGL. It runs on Microsoft Windows 2000. Its architecture is based in two PCs. The first one (control PC) executes the control module to command the haptic. The second one (simulation PC) runs the main, the Graphical User Interface (GUI), the collision solver and the graphics engine modules. Both PCs are connected through an Ethernet LAN network.

The control PC has a 233 MHz Pentium II. The simulation PC has two 866 MHz Pentium III Xeon processors with an Intense3D Wildcat 4210. To achieve the maximum frame rate in the visualization and collision modules, those tasks have been distributed between the two processors. Fig. 2 shows the system architecture.

The simulation PC has two video outputs: one contains the GUI, the other one displays the 3D virtual scene. This second display can be connected to a monitor, a screen projector or both.



Interdependencies between modules

Video output

Fig. 2 : System Architecture of REVIMA

The communication between two PCs uses both UDP and TCP protocols. The UDP protocol is used in the normal and general case: with position messages and collision messages.

The positions messages send from control PC to simulation PC when the Control Module detects position or orientation changes. When the Collision Solver detects interferences, this module sends collision messages to Control Module. Even though, an UDP channel does not guarantee reliability, it is the most appropriate one for a real time application that makes use of a point-to-point connection. The system is not affected if it occasionally misses one of these messages.

When REVIMA needs a reliable communication channel, it uses the TCP protocol. Examples of this kind of messages are: starting the control module from the simulation PC, messages to grasp or release pieces from the assembly, notification from the control PC to the Simulation PC such as problems with the haptic interface, i.e., excessive heat of the motors, etc.

4- Visualization Module

The visualization module generates images that replace the physical viewing of the mock-up. Our aim was to develop a program, where the virtual aircraft engine can be displayed in

an interactive frame rate, more than 10 fps. The necessary steps to achieve an interactive frame rate have followed three ways:

1. Find the most suitable structure for storing the scene.
2. Analyze the most optimized graphics commands.
3. Develop visualization techniques.

Before analyzing the visualization module itself, we shall present some experiences related to the source data. The system receives CAD data structures, without element limitation and based in nodes. This information is received loading a text file based in the "product structure" format. This file defines a structure where is stored the information that places spatially the model and defines the operational features of each element of the model.

The structure defined in the file is a hierarchical one (see Fig. 3). The structure has a root node, which can have an unlimited number of child nodes. Each node of the structure provides information for operational purposes, indicating which is its behavior in the simulation operations. A position matrix also is contained in the nodes that place spatially its descendents in the scene.

The leaf nodes structure contains the name of a graphic file that defines the actual geometry and topology of one part. These files can be VRML, GAF or STL format

The aircraft engine digital models used in the virtual environments are very complex, with a great density of faces per element. An engine consists of around 2000 elements. The sizes of the elements are different. Some elements have few polygons as the bolts that consist of around a hundred polygons. On the other hand some elements are very big, more than a hundred thousand polygons are necessary to define them.

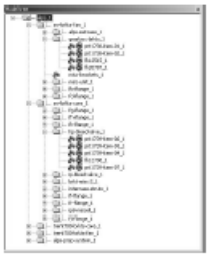


Fig. 3 : Input Hierarchical structure

4.1- The most suitable structure

It is very important to find the most suitable structure for storing the information of the scene [2].

Due to the structure that is read from the files, in our first design we chose a sloppy n-ary space partitioning tree [3] to store all the geometrical information. This structure does not impose a fixed number of child nodes and the bounding volumes of tree nodes of the same tree level are not necessarily disjoint. These features make suitable this structure in relation to the one that is received.

After working with this structure, the frame rate achieved, less than 3 fps, was not as good as required. The problem was found in the input structure. This first structure was built following the operational structure, instead of geometrical properties. As a consequence, the visualization techniques that must be applied at the rendering process and which will be explained later, were not good enough.

The next step was to build a hierarchical structure where the scene was stored following spatial sorting. This structure was totally independent in relation to the input structure. Anyway both structures should be maintained, because the input structure is necessary for the simulation operations.

The chosen hierarchical structure was an Octree (Fig. 4). This structure has an important disadvantage. The objects of the scene must be split up to suit the structure. On the other hand it has a valuable advantage: It supplies valuable spatial information about the scene.

When we traverse this structure to make the visibility computations, the spatial information supplied is useful. However, we are still studying the problem in order to find automatically which is the optimal decomposition level for an arbitrary model.

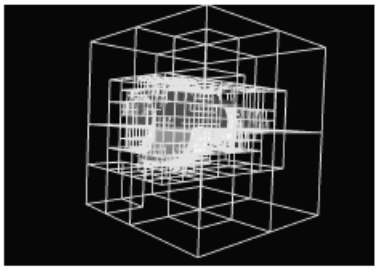


Fig. 4 : An Aircraft Engine Stored in an Octree Structure

4.2- Representation techniques

The rendering had to be done using the OpenGL API. So, it was important to find the fastest representation techniques that OpenGL could offer.

Two graphic cards have been used to render. One of the cards belongs to the family of the Wildcat GPU and the other one was built up on the geForce3 chips of nVidia. The most interesting discovery is that in both cards the most effective representation techniques were different.

In the case of the Wildcat, the use of display lists was the most effective technique. The test we performed showed that using a display for each part was the most optimized method, reaching twice bigger fps than using immediate mode commands.

However this technique is not as fast as the use of the vertex arrays for the other cards. The reason of this difference is that the graphic cards based on the geForce3 chip, allow storing vertex information in video memory. This way the access to this information is fastest, and the rendering process is accelerated considerablely.

4.3- Visualization methods

The system has implemented several visualization methods. These methods when rendering the scene, try to use the minimum number of polygons.

The program has implemented simplification algorithms. We have implemented the quadric error metrics algorithm [4]. The input data specifies the degree of simplification for each element when the model is charged. The degree of simplification does not change during the simulation operations, because the entire scene has not enough depth to use the LOD technique [5]. The use of LODs also supposes to employ twice the necessary memory, which was not available.

When the scene is displayed, all the elements must not be rendered. Some elements are not visible due to several reasons and other elements are not enough important because of their tiny contribution to the final image.

An element may not be visible for several reasons: it is outside of the viewing pyramid, it is hidden by other elements or the user has marked it as not visible. Also the faces whose normal points away from the viewpoint are not visible.

The elements whose projection is less than a threshold value are not rendered because it is considered that their contribution to the final image is not worthy. It is very important to identify these elements before calling the commands that render them.

The use of efficient hierarchical structures is very important to identify these elements. Bounding volumes of all the nodes of the structure are calculated. The use of these volumes makes easier the necessary calculus. The bounding volumes are spheres because they are the simplest and fastest for calculus purposes [6]. With a collision test between the bounding volume and the viewing pyramid can be known if the elements inside the bounding volume are or are not inside the viewing volume. If the volume is totally outside the viewing pyramid, all the elements are not visible; on the other hand if the volume is inside all the elements are visible. If both volumes collide, there is not enough information and the process follows in the next level of the hierarchical structure. In order to identify if an element contribution to the image is important, the same method is employed. In this case the bounding volume projection area is calculated.

4.4- Occlusion methods

An algorithm that finds which objects are not visible has been also developed. Although we must stress that most of the algorithms used nowadays are not valid within our application due to specific features of our virtual environment.

A lot of algorithms are based in an important pre-process step [7]. These algorithms usually precalculate the objects that are hidden from different possible camera positions. Therefore, at any moment the hidden objects are known just knowing the camera position. These methods are not valid for our program, because the user can vary the visibility state of the elements in run time; therefore the occlusion information obtained in the pre-process step loses all its value.

Either, the methods based in replacing part of the scene by images are not also valid [8]. The scene is not depth enough to replace some elements by an image without producing visual artifacts.

The working scene does not also fit to the methods that divide the scene in cells and portals [9]. These methods are used in architectural environments.

4.5- Occlusion method implemented

Analyzing the main features of the working scene we concluded that the method that best fits is the one based in hierarchical occlusion maps [10].

This method calculates a set of possible occluders in a pre-process step. The set of occluders are different for different viewpoints. Good occluders for each point are those whose projection area is big and those that are usually near from the viewpoint.

When the set of occluders is projected to the screen, the area of its projection is made opaque. An occlusion map is a two-dimensional array. The information stored in an occlusion map indicates which part of the image has become opaque. Once this occlusion map is obtained, applying the average operator to rectangular blocks of pixels, we get occlusion map at different resolutions.

An estimation buffer is also built at every frame, which requires determining the depth value in every pixel where the occluders have been projected. This operation is very expensive, so, instead of calculating the exact value, depth estimations are calculated.

Once the information about the projected area and the depth is obtained the rendering step begins. The following two tests are performed for each element. The screen projection of the element is tested against the occlusion maps to see whether it is completely within the opaque area of the occlusion map. Also is compared against the depth information to determine if it is behind the occluders. If the element is overlapped and behind the occluders, then, it is occluded.

As we needed to improve the throughput of this technique we studied how to save computations, we realized that an important feature is that the aircraft engine virtual models have a cylindrical shape.

Another important property of our program is a special navigation system. To make easier the simulation operations the camera must be located on a cylindrical surface, looking at the center of the engine all the time.

Using these characteristics properly, the step where the depth information was calculated may be avoided. The shape of the model and the navigation mode makes very useful the plane

that is perpendicular to the viewpoint and crosses the center of the shape.

When we are going to calculate the occlusion map, the occluders projected are those that are between the camera and this plane. Then, an element is considered hidden if the occlusion map overlaps its projection and it is behind the plane.

In the Fig. 5 can be seen how many elements are considered as hidden for the different positions a camera can take.

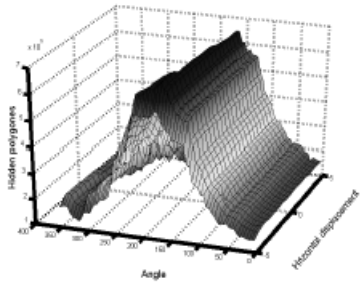


Fig. 5 : Number of Objects ($x10^5$) occluded for different camera positions

5- Collision Module

The collision module sends messages to the control module when a collision is detected among the objects. The control PC with this information calculates a force feedback that is applied by the haptic interface, inhibiting some user movements. This is a simplification of the problem. In section 6 the problem of penetration and control is analyzed.

It should be noticed that the collision problem we are tackling with has two specific characteristics. The first one has a geometric nature. There are a reduced number of mobile objects (the hand and the grabbed tool or engine part) while the rest of the scene is a large set of static objects. Therefore, the mobile objects are only about a 0.5% of the total number of facets in the scene. The second one is the relative position between the static and mobile geometry: the facets of the mobile objects are usually close to a subset of static facets.

These two characteristics are common to other environments whose maintainability must be analyzed, and have a notable influence in the algorithms required when checking for collisions and penetrations. This environment is quite different from others that also deal with the collision problem: path finding, robotics, vehicles, virtual walks, even mechanism design.

The algoritm used is based on uniform spatial grid decomposition: voxels [11]. Each voxel can point to a list of facets that intersect with its volume.

Instead of building the voxel decomposition for each static object, we make only one structure, treating all the static objects as a unique solid [12], [13].

Here we present a similar method of [12]. We named it *MF* (Mobile with Facets). This method doesn't use voxels to describe the mobile objects; it uses only facets. The *MF* algorithm uses only two levels of accuracy: interferences among mobile facets and static voxels; and interferences among mobile and static facets.

In the first one, for each mobile facet it computes the subset of static voxels intersected by that mobile facet. If the subset of voxels has some facet, in a second level of accuracy, the algorithm takes the facets contained in those voxels and searches for interference among each pair of facets. We use an algorithm from Möller [14] to detect an intersection between two triangles.

Besides the detection problem, the collision solver also computes a collision response. This response is two values, the normal contact and the penetration between two 3D objects, and the collision module sends them to control module.

6- Control Module

The control loop, located in the control PC, has a sampling period of 1 kHz. Most authors choose one sample frequency of 500 Hz or 1 kHz, according to the studies of Shimoga [15].

This module acquires the position and orientation of the tracking device and sends this information to the simulation PC.

Collision forces are calculated depending on the actual user position, the last collision information received from the simulation PC and the contact model in use. Fig. 6 shows the data that make up the collision information: the contact normal **n**, the penetration x and the tool position **p**.

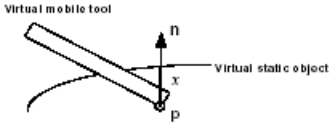


Fig. 6 : Collision information

The contact model used in REVIMA is a simple spring [16]. The collision force is directly proportional to the penetration x of the virtual tool in the environment. It also has been implemented a friction model that follows the one described by Salisbury et al. [17].

Since the control loop runs faster than the collision module, we have considered the following three problems that generate sudden changes in the contact force.

The first one is the delay that exists in the collision information. When a collision message arrives at the control

PC, its information is only true for a previous position of the user, not the actual.

The second problem is what to do when there is not new collision information for new control iteration. This happens because collision analyses are slower than the control loop.

The last problem is how to solve the great changes in normal direction and penetration, which can appear every time that new collision information is received.

The numerical strategies we have implemented to solve these problems permit a sufficient stiff and smooth touch simulation even in presence of several contact points or great changes in the contact surface.

7- Conclusions

The main conclusion of this work is that virtual reality systems with force feedback based on low cost hardware can be directly applied to the industry and that these systems may lead to important reduction in costs.

The system has shown that it is possible to have a device with a workspace as large as an aircraft engine.

Another interesting characteristic of this system is that CAD models used in the design phase are also useful for simulation studies without manual or interactive preprocessing.

The used architecture and visualization policy has proven to be a good approach to treat the problem of managing large CAD models in real time. The specific characteristics of this problem allow us speeding up occluding computations.

There are two complete developed systems that are being used and validated by ITP.

8- Acknowledgments

The authors would like to thank to ITP and Sener for promoting and finding the application REVIMA. This application has been partially financed by the project of the Basque Government, number CI01TP03.

We would like to especially thank Iker Aguinaga, Emilio Sánchez, Angel Rubio and Jaime Rubí for their support and ideas in the development of this work.

9- References

[1] Blanchard B., Verma S.D. and Peterson, E.L. Maintainability. John Wiley & Sons Inc., New York, USA, 1995.

[2] Meißner M., Bartz D., Hüttner T., Müller G. and Einighammer J. Generation of Subdivision Hierarchies for Efficient Occlusion Culling of LargePolygonal Models. Technical Report WSI-99-13, Department of Computer Science, University of Tübingen, 1999.

[3] Bartz D., Meißner M. and Hüttner T. OpenGL-assisted Occlusion Culling for Large Polygonal Models. Computer and Graphics, Vol.23, No. 5, 1999.

[4] Garland M. and Heckbert P. Surface Simplification Using Quadric Error Metrics. Computer Graphics (SIGGRAPH 97 proceedings), 209-216, 1997.

[5] Aliaga, D., Cohen, J., Wilson, A., Baker, E., Zhang, H., Erikson, C., Hoff, K., Hudson, T., Stuerzlinger, W., Bastos, R., Whitton, M., Brooks, F., and Manocha, D. "MMR: An Interactive Massive Model Rendering System Using Geometric and Image-Based Acceleration," Symposium on Interactive 3D Graphics '99 Proceedings, 199-206, 237, 1999.

[6] Bartz D., Klosowski J.T. and Staneker D. k-DOPs as Tighter Bounding Volumes for Better Occlusion Performance. ACM SIGGRAPH, Conference Abstracts and Applications, page 213, 2001.

[7] Saona-Vazquez C., Navazo I. And Brunet P. The visibility Octree. A Data Structure for 3D Navigation. Computer & Graphics 23 pp. 635-643, 1999.

[8] Aliaga D. and Lastra A. Automatic Image Placement to Provide a Guaranteed Frame Rate. Proceedings of ACM SIGGRAPH, pp. 307-316, August 1999.

[9] Luebke D. and Georges C. Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets. In ACM Interactive 3D Graphics Conference, Monterrey, CA, 1995

[10] Zhang H., Manocha D., Hudson T. and Hoff K. Visibility Culling Using Hierarchical Oclusion Maps. In Proc. Of ACM Siggraph, 1997.

[11] García-Alonso A., Serrano N. and Flaquer J. Solving the Collision Detection Problem. IEEE Computer Graphics and Applications, 13(3): 36-43, May 1994.

[12] Held M., Klosowski J.T. and Mitchell J.S.B. Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs. Proceedings of the Seventh Canadian Conference on Computer Geometry, Vol. 3, pp. 205-210, Québec City, Québec, Canada. Agosto 1995.

[13] McNeely W. A., Puterbaugh K. D. and Troy J. J. Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling. Proceedings of the ACM Siggraph, pp. 401-408. Los Angeles, California, USA. August 1999.

[14] Möller T. and Haines E. Real-Time Rendering. A K Peters Ltd., 1999.

[15] K. Shimoga. Finger Force and Touch Feedback Issues in Dextrous Telemanipulation: A Survey. Proc. of NASA-CIRSSE International Conference on Intelligent Robotic Systems for Space Exploration, NASA. 159-178, Greenbelt, September 1992.

[16] Burdea G.C. Force and Touch Feedback for Virtual Reality, John Wiley and Sons, Inc., New York, 1996.

[17] Salisbury K., Brock D., Massie T., Swarup N. and Zilles C. Haptic Rendering: Programming Touch Interaction with Virtual Objects. Proceedings 1995 Symposium on Interactive 3D Graphics. 123-130, Monterey, California, 1995.

Realidad virtual para el mantenimiento de motores de avión

A. Amundarain†, I. Aguinaga†, D. Borro†, A. Garcia-Alonso‡, L. Matey†.

† CEIT (Centro de Estudios e Investigaciones Técnicas de Gipuzkoa)
Lardizabal 15, E-20018, San Sebastian
Tlf: 943 219 877 Fax: 943 311 442.
e-mail: {aamundarain, iaguinaga, dborro, lmatey}@ceit.es

‡ Universidad del Pais Vasco
Lardizabal 1, E-20018 San Sebastian
Tlf: 943 015104 Fax: 943 219306. e-mail: agalonso@si.ehu.es

Resumen

Exponemos un sistema de realidad virtual para la simulación de procesos de mantenimiento en aeronáutica que se ha desarrollado en el marco del Proyecto REVIMA. Hemos desarrollado un hardware nuevo (háptico) y resuelto diversos problemas al elaborar el software que integra el sistema. El nuevo háptico se emplea para realizar el seguimiento del movimiento de las manos, y al mismo tiempo devuelve una fuerza que provoca la sensación de estar trabajando con una maqueta física. Los principales módulos de software son: el módulo de generación de imágenes, el módulo de colisión y el módulo de control. La integración del sistema está basada en dos ordenadores conectados mediante una LAN que comparten diferentes tareas y datos.

El módulo de visualización ha sido implementado empleando sistemas gráficos de bajo coste y mediante un estudio exhaustivo se ha logrado una frecuencia válida para las operaciones de simulación. Los modelos consisten en más de dos mil elementos diferentes y emplean alrededor de dos millones de polígonos para describir su geometría.

Este trabajo presenta una breve descripción del conjunto y desarrolla de modo específico las técnicas de visualización que se han desarrollado.

Palabras clave: Realidad Virtual, visualización en tiempo real, mantenimiento, háptico.

1 Introducción

En el campo de la aeronáutica el término mantenibilidad se define como: "la habilidad de un elemento de mantenerse en servicio o volver al estado adecuado para que desarrolle su función, después de haber sido mantenido en condiciones previamente establecidas, empleando el personal, los medios y los procedimientos adecuados" **[1]**.

Uno de los aspectos de mayor interés de la mantenibilidad concierne al análisis de accesibilidad del hombre y la herramienta, donde se acomete el cálculo de vías de acceso, de secuencias de ensamblaje y desensamblaje y de tiempos. El diseño basado en maquetas electrónicas esta siendo empleado de forma habitual en la creación de las partes externas del motor (tuberías, guarniciones e instalaciones) por la industria aeronáutica.

La tecnología DMU/DPA (Digital Mock Up / Digital Pre Assembly) ha logrado que el empleo de maquetas físicas no sea necesario en la fase de diseño, reduciendo considerablemente el tiempo necesario para comercializar el producto y por consiguiente ahorrando dinero. Sin embargo, hoy en día todavía es necesario el empleo de maquetas físicas, para poder evaluar la mantenibilidad de las partes externas en la etapa de desarrollo [***Figura 1***]. Aunque estas maquetas pueden ser empleadas en otras aplicaciones, la razón principal para la construcción de estas maquetas suele ser la evaluación de la mantenibilidad.

Figura 1 Operación de mantenimiento en un motor de avión

El coste de estas maquetas ha conducido a ITP (Industria de Turbopropulsores) a investigar alternativas empleando hápticos.
Por tanto el objetivo principal de este proyecto es desarrollar un sistema de realidad virtual que permita predecir la mantenibilidad de un motor de avión, simulando las tareas necesarias para ello. Al llevar a cabo este proyecto se busca que las maquetas físicas dejen de ser necesarias para los estudios de mantenibilidad, logrando una importante reducción de costes en el desarrollo de motores de avión.

2 Descripción del sistema

REVIMA es un sistema háptico desarrollado para comprobar la mantenibilidad de los motores de avión. El sistema ha sido desarrollado desde cero por el departamento de mecánica del CEIT. La consecución del proyecto ha supuesto un desarrollo multidisciplinar donde han tomado parte, entre otros, las siguientes áreas: Diseño mecánico, teoría de control, gráficos por ordenador, geometría computacional, interfase hombre-máquina...

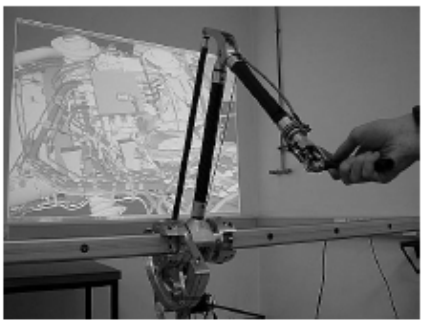


Figura 2 REVIMA realizando una operación de mantenimiento

La investigación realizada en el proyecto se puede dividir en dos áreas principales: Diseño mecánico y desarrollo de software. Ambos han supuesto importantes retos debido a que la simulación de la mantenibilidad exige estar muy cerca de la realidad.
Uno de los objetivos del diseño mecánico era que el espacio de trabajo del háptico coincidiese con las dimensiones de un motor de avión, manteniendo como todo

sistema háptico, una inercia baja. Esto se ha logrado combinando el diseño mecánico con el uso adecuado de sensores de fuerza. ITP ha establecido como condición que el sistema háptico tenga esas dimensiones, para que se puedan realizar estudios de ergonomía. El sistema ofrece reflexión de fuerza en los tres grados de libertad de translación, mientras que se pueden medir las orientaciones pero no se actúa sobre ellas. Esto permite simular el empleo de cualquier tipo de herramienta a la hora de realizar las operaciones de mantenibilidad.
Por otro lado el desarrollo del software ha supuesto la integración de un lazo rápido de control que calcula las fuerzas aplicadas por el sistema háptico, un evaluador de colisiones y la visualización de la escena. Las grandes dimensiones de los modelos (más de dos millones de polígonos) dificultan en gran medida las dos últimas tareas.

3 Arquitectura del sistema

REVIMA es un complejo proyecto en el que se han integrado nuevos equipos (hardware mecánico) y sistemas software (gráficos por ordenador, geometría computacional, control). El software ha sido desarrollado empleando Visual C++, OpenGL y Microsoft Windows 2000. Su arquitectura se basa en dos PCs [***Figura 3***]. En el primero, el PC de control, se ejecuta el módulo de control que sirve para gobernar el háptico, mientras que en el segundo, el PC de simulación, se ejecutan el módulo principal, el GUI (interfaz gráfica de usuario), el módulo de colisiones y el motor gráfico. Ambos PCs están conectados mediante una red Ethernet LAN.
El PC de control tiene un procesador Pentium II a 233 MHz y mientras el de simulación tiene dos procesadores Pentium III Xeon a 866 MHz con una tarjeta gráfica Intense 3D Wildcat 4210. Para obtener el mejor rendimiento en los módulos de visualización y de colisiones, ambas tareas han sido distribuidas entre los dos procesadores. El PC de simulación tiene dos salidas de video; una sirve para mostrar la interfaz de usuario y la otra para visualizar la escena tridimensional. La escena puede ser representada tanto en estéreo activo como pasivo. La segundo salida puede conectarse a un monitor o a una pantalla con proyector.
La comunicación entre ambos PCs se realiza empleando los protocolos UDP y TCP. El protocolo UDP se emplea cuando se envían mensajes de posición y de colisión. Los mensajes de posición los envía el PC de control al PC de simulación en cada ciclo del lazo de control. El PC de simulación únicamente recoge el último mensaje enviado antes de empezar a renderizar. Todos los mensajes enviados mientras el PC de simulación representa la escena son desechados El módulo de colisión se pone en contacto con el PC de Control mediante mensajes de colisión

cuando se detecta una interferencia, de forma que el háptico pueda ofrecer la respuesta adecuada. Aunque un canal UDP no garantiza la fiabilidad en el envío de datos es el más adecuado para una aplicación en tiempo real cuando se emplea una conexión punto a punto, debido a su rapidez.

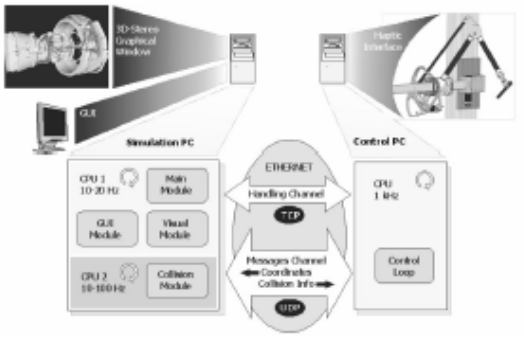


Figura 3 Arquitectura del sistema REVIMA

Cuando REVIMA necesita una comunicación fiable entonces emplea el canal de comunicación TCP. Este canal de comunicación es necesario cuando se quiere asegurar la recepción del mensaje, como en los siguientes casos: En el proceso de puesta en marcha de los módulos del sistema, cuando se selecciona o deselecciona alguna pieza, cuando existen problemas con el sistema háptico como el sobrecalentamiento de los motores...

4 Módulo de visualización

El módulo de visualización genera imágenes para reemplazar la visión física de la maqueta. El objetivo de este módulo es crear un motor gráfico que permita visualizar los motores de avión virtuales [***Figura 4***] a una frecuencia interactiva.
Antes de realizar una descripción detallada del módulo de visualización, procederemos a presentar las características de los datos que se han de visualizar. Los datos que se reciben son estructuras de datos de CAD, basados en nodos y sin límite de elementos. Esta información se recibe cargando un fichero de texto con el

formato "product structure". En este fichero define una estructura jerárquica, en la que se almacena la información que posiciona espacialmente los elementos del motor e indica sus características funcionales.

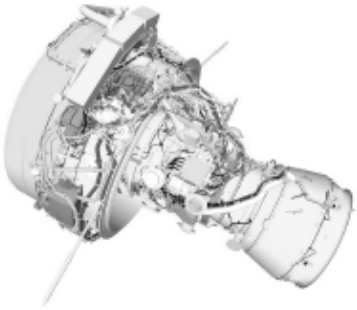


Figura 4 Vista de un motor de avión virtual

La estructura empieza a partir de un nodo raíz, y de cada nodo de la estructura puede colgar un número ilimitado de nodos. Cada nodo posee información operacional que indicará el comportamiento de cada nodo durante la simulación. Mediante una matriz de posicionamiento que se almacena en cada nodo se posiciona espacialmente cada nodo y los nodos que cuelgan de él.

Los nodos finales de la estructura almacenan la información geométrica de los elementos del motor. Esta información puede almacenarse en diversos formatos: VRML, GAF y STL.

Los modelos de avión empleados son muy complejos, con una densidad muy alta de caras por elemento. Un motor suele constar del orden de dos mil elementos. El tamaño de los elementos difiere de forma considerable. Se pueden encontrar tornillos definidos con tan solo cien polígonos, mientras que otros, como la carcasa, pueden alcanzar los cien mil polígonos.

El principal problema del modulo de visualización consiste en que los modelos virtuales de avión están compuestos de un número demasiado alto de polígonos para poder representarlos a una frecuencia interactiva. Es necesario por lo tanto desarrollar técnicas de visualización que permitan logran la frecuencia deseada. Se ha partido de técnicas ya implementadas y estas se han amoldado a las características de nuestro escenario, teniendo en cuenta que cualquier elemento del motor del avión puede ocultarse y moverse a voluntad del usuario, lo que influye en los cálculos de visibilidad.

Una de las formas más habituales para reducir el número de polígonos a representar es la simplificación de la geometría de los objetos **[2]**. El problema en este caso es que en este proceso un objeto pierde precisión y la calidad de imagen se deteriora. Como se necesita una visión muy detallada de los objetos a la hora de realizar las operaciones de mantenimiento, no es muy adecuado simplificar los objetos. De esta forma, debido a las características del programa, no se recomienda simplificar la geometría, aunque se le ofrece al usuario la opción de poder hacerlo mediante la configuración del programa. Se podría usar la técnica de los niveles de detalle **[3]** (LOD) para solventar este problema, permitiendo observar de forma detallada los objetos cercanos y simplificados los que se encuentran más lejanos. El inconveniente de emplear LOD-s consiste en que requiere el uso del doble de cantidad de memoria. Este incremento en la memoria no es asimilable debido a que el módulo de colisiones también hace un uso importante de ella.
A la hora de representar, se consideran los distintos elementos como entidades indivisibles. Es decir, se representa el objeto en su totalidad o no se representa, pero no se puede representar únicamente unos polígonos del objeto. Esto se hace para optimizar las llamadas al sistema gráfico, mejorando de este modo el rendimiento.
Una forma de acelerar la frecuencia de refresco es la identificación de los objetos no visibles antes de mandarlos a renderizar. Esta tarea se centrará en identificar los objetos que se encuentren fuera del volumen de visión y los que se encuentren ocultos por otros elementos de la escena. Muchos polígonos no suelen ser visibles debido a que su cara frontal da la espalda al punto de vista. No se estudia la identificación de estos polígonos debido a que tratamos los elementos como una entidad indivisible y la mayoría de las tarjetas gráficas actuales identifican estos polígonos por hardware.

4.1 Frustum Culling

Los objetos que se encuentran fuera del volumen de visión se identifican de forma sencilla; mediante un test de colisión entre el volumen contenedor del objeto y el volumen de visión se puede saber si el objeto está totalmente fuera del volumen de visión. En esta etapa también analizamos cuantos píxeles comprende la proyección de los volúmenes contenedores, si el valor que se obtiene es menor que cierto límite preestablecido, se decide no renderizar el objeto debido a que su aportación a la imagen final se considera mínima.
Para optimizar ese proceso se decidió hacer uso de una estructura jerárquica, empleando la estructura jerárquica que se recibe en la entrada de datos. Mediante la estructura jerárquica se pueden identificar los objetos que se encuentran fuera del volumen realizando un número menor de operaciones, pero vimos que no se obtenía una mejoría perceptible debido a que la operación necesaria para identificar

los objetos que se encuentran ocultos es muy rápida. El volumen contenedor empleado al principio fue la esfera. La ventaja de la esfera es que su geometría permite realizar los cálculos con menor coste, pero sin embargo es un tipo de volumen que no se adecua muy bien a los objetos. Debido a esto, muchos objetos que no son visibles, no son identificados como tales debido a que su esfera contenedora se encuentra dentro del volumen de visión. Debido a las características de nuestros modelos virtuales se decidió emplear un nuevo volumen contenedor [***Figura 5***]. Se calcula el convex hull bidimensional en las direcciones Y y Z y en el eje X se encuentran los puntos más distantes. Se uso un volumen contenedor similar a un cilindro con eje en X y en vez de un círculo se ajusta a la forma del objeto mediante un convex hull. Mediante este volumen envolvente los cálculos son más costosos, pero se logra identificar un número mucho mayor de elementos ocultos. Este volumen envolvente, también optimiza los cálculos en el método de oclusión culling que se describirá a continuación. En las pruebas realizadas, el número de objetos identificados ha aumentado entre el 30 y el 50% dependiendo de la situación del motor de avión respecto al volumen de visión.

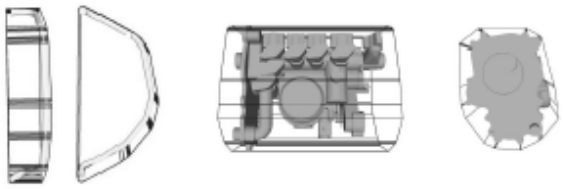


Figura 5 Vistas frontal y lateral de dos objetos con su volumen envolvente basado en convex hull bidimensional.

4.2 Oclusión culling

A la hora de identificar los objetos ocultos por los diferentes elementos que constituyen la escena, se ha realizado un amplio estudio sobre los métodos de oclusión desarrollados hasta ahora. Entre los diferentes métodos existentes se ha juzgado que el método de mapas de oclusión jerárquicas (HOM) **[4]** es el más adecuado al problema que se trata de resolver. La principal razón por la que se ha seleccionado este método es debido a su generalidad, pues se adecua a toda clase de modelos, y no solamente a escenas arquitectónicas **[5][6]** o escenas con grandes polígonos ocultadores **[7]**. Una vez seleccionado este método se le han introducido nuevas modificaciones para que se adecue a las características de nuestra escena.

Primero describiremos las características principales del método de partida, para luego resaltar las modificaciones que se han introducido.
HOM primeramente ha de seleccionar los objetos con mayor capacidad de ocultación, los ocultadores, desde cada punto de vista. Esto se puede realizar en una etapa de preproceso o mientras se representa la escena empleando la coherencia temporal. Una vez que se han seleccionado los ocultadores, estos se proyectan en un buffer auxiliar. Para la proyección se desactiva toda iluminación y se dibuja todo en blanco. De esta forma se obtiene un mapa de oclusión que nos indica que áreas de la pantalla serán ocupadas por los elementos con mayor probabilidad de ocultar a los demás.
También es necesario obtener una estimación de la profundidad de los elementos proyectados. Dependiendo de las necesidades se puede realizar una estimación de profundidad exacta o una aproximada. Una vez que se tiene el mapa de oclusión, se crean mapas a menor resolución a partir de ella, obteniendo una representación jerárquica de los mapas de oclusión. Una vez que se obtienen los mapas de oclusión y la información de profundidad [***Figura 6***], se analiza si la proyección de los volúmenes contenedores de los objetos está oculta por los mapas y si estos objetos se encuentran a mayor profundidad, en el caso de que la respuesta sea afirmativa se sabe que ese objeto se encuentra oculto.

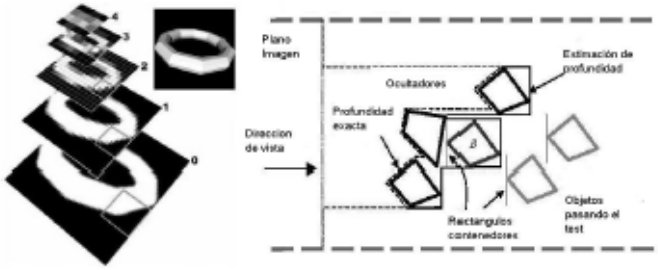


Figura 6 A la izquierda mapa de oclusión jerárquica y a la derecha posible estimaciones de profundidad. Con trazado discontinua una estimación exacta y con trazado continuo una estimación aproximada.

Debido a que el cálculo de los mapas de oclusión y de la información de profundidad necesita una cantidad de tiempo no despreciable, para que el método resulte efectivo se han de identificar una cantidad considerable de objetos ocultos.

La técnica se convierte en efectiva, independiente de la cantidad de objetos identificados, si se paraliza en dos procesadores; en un procesador se ejecutan los pasos para calcular los mapas y en el otro procesador se identifican los objetos ocultos haciendo uso de los mapas.
En nuestro caso, la cantidad de objetos ocultos identificados no es lo suficientemente grande para que el método sea efectivo y no se tienen dos procesadores disponibles para paralelizar el proceso, de forma que se ha hecho uso de las características específicas de nuestras escenas para convertir la técnica en efectiva.
Las modificaciones se han introducido a partir de dos características de nuestro programa. Una característica es inherente a los modelos que se han de representar, pues al ser motores de avión, tienen una forma básica que se puede asemejar a la de un cilindro. La siguiente característica es una consecuencia del sistema de navegación desarrollado para que el usuario pueda realizar las operaciones de mantenimiento con el háptico sin preocuparse de controlar la cámara. Para ello se ha desarrollado un sistema de navegación que se encarga de que la herramienta de trabajo siempre se encuentre enfocada. Para tener un enfoque adecuado se obliga que la dirección de la cámara apunte constantemente hacia el eje del motor.

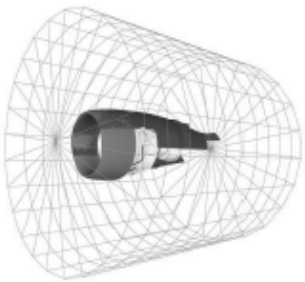


Figura 7 División en sectores cilíndricos de la escena

La primera modificación introducida al método de HOM es en el cálculo de los ocultadores. Los ocultadores se calculan en una etapa de preproceso. Se calcula una serie de ocultadores para las diferentes regiones del espacio donde se puede encontrar el punto de vista. Debido a que la cámara apunta siempre hacia el eje del motor, se ha comprobado que lo más adecuado es dividir la escena en sectores cilíndricos [***Figura 7***], y calcular para cada sector cilíndrico los objetos con mayor capacidad de ocultación.

Al no poder paralelizar el proceso, se debía de encontrar una forma que permitiese reducir el tiempo necesario para calcular los mapas de oclusión y la información de profundidad. Debido a las dos características anteriormente citadas se observó que se podía prescindir del cálculo de la información de profundidad. Si se divide el cilindro por un plano perpendicular a la dirección de visión y que pasa por el eje [***Figura 8***], se sabe que la mayoría de los objetos ocultos estarán detrás de este plano.

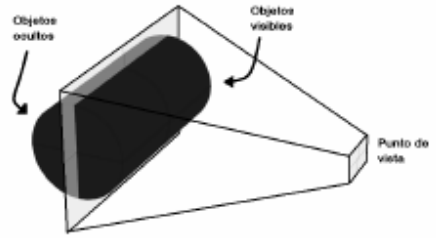


Figura 8 Plano que divide el motor entre posibles objetos ocultos y objetos visibles

Para identificar correctamente los objetos que, estando detrás del plano, estén ocultos, se debe de tener cuidado a la hora de calcular los mapas de oclusión. En la creación de estos mapas no deben contribuir las partes de los ocultadores que se encuentren detrás del plano perpendicular. Esto se logra empleando un volumen de visión diferente a la hora de proyectar los ocultadores en el buffer auxiliar. Este volumen de visión es similar al que se emplea al representar el modelo virtual con la excepción de que el plano lejano del volumen de visión coincide con el plano perpendicular. De esta forma se logra que sólo contribuyan al mapa la parte de los ocultadores que se encuentra por delante del plano, y las otras partes son eliminadas empleando las capacidades de clipping de la tarjeta gráfica.

En las pruebas se ha observado que el plano que pasa por el eje x no es el plano con el que se obtienen mejores resultados. Si se acerca el plano levemente hacia el punto de vista se observa que se pueden identificar una cantidad superior de objetos que están ocultos. Esto es debido a dos razones. Una razón consiste en las características de la proyección en perspectiva. También es debido a que se consideran objetos no visibles únicamente aquellos que están detrás del plano. Si el plano se acerca al punto de vista habrá más objetos que serán candidatos a estar ocultos. Sin embargo si se acerca el plano, los mapas de oclusión serán peores debido a que habrá menos objetos para crearlos.

Mediante la aplicación de las diversas técnicas descritas, se logra aumentar la frecuencia en un 40% de media, alcanzando unos valores que permiten realizar las operaciones de mantenimiento.
Aunque se ha limitado el uso de la técnica descrita a un entorno específico, está técnica se puede generalizar a entornos donde se pueda dividir la escena mediante un plano que defina una zona donde los objetos tengan una importante probabilidad de ser visibles y otra zona donde la probabilidad que los objetos se encuentren ocultos sea alta.

5 Módulo de colisión, control e interfaz háptico

El módulo de colisiones envía mensajes al módulo de control cuando detecta colisiones entre los objetos. Gracias a esta información el PC de control gobierna la respuesta del háptico, restituyendo fuerza en caso de contacto.
El algoritmo empleado esta basado en una descomposición de la escena en una malla de volúmenes uniformes, donde a su unidad básica se le llama voxel **[8]**.
En vez de descomponer en voxels cada objeto estático, se crea una única estructura considerando todo los objetos estáticos como un único sólido **[9]**. Los objetos móviles son descritos mediante sus facetas.
Como primer paso, para cada faceta móvil se calcula con que voxels está colisionando. Si los voxels calculados tienen facetas, se consigue un mayor grado de exactitud calculando si la faceta móvil interseccíona con las facetas que se encuentran en los voxels.
Además de detectar colisiones, se ha de poder informar al módulo de control sobre las características de la colisión para que el háptico pueda aplicar la fuerza adecuada. La información consiste en la dirección de la colisión y en la penetración entre los cuerpos que estén colisionando.
Mediante el módulo de control se informa al PC de simulación de los cambios de posición y orientación del háptico.
El lazo de control situado en el PC de control tiene una frecuencia de muestreo de 1 KHz **[10]**. Debido a que está frecuencia es superior al del módulo de colisiones, se han implementado diferentes estrategias numéricas para lograr un tacto suave y rígido. Estas estrategias están basadas en la interpolación de los valores que son enviados por el módulo de colisiones **[11]**.
El interfaz háptico desarrollado y construido, es un ejemplo de hardware de reflexión de fuerzas fijo. El sistema se encuentra anclado al suelo, por las dimensiones del espacio de trabajo. El espacio de trabajo es un sector cilíndrico cuyas dimensiones permiten trabajar a escala 1:1 con el modelo virtual del avión.

Las dimensiones del espacio de trabajo son las siguientes: radio interno, 242mm; radio externo 772mm; profundidad 1500mm y ángulo 120º.

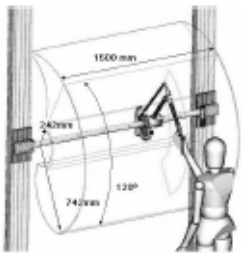


Figura 9 Dimensiones del espacio de trabajo

6 Conclusiones

Hemos desarrollado un sistema de realidad virtual con reposición de fuerzas empleando hardware de bajo coste. Estos sistemas pueden ser empleados directamente en industria, lo que puede conducir a un importante ahorro en los costes de mantenimiento. Hoy en día hay dos sistemas desarrollados completamente que han sido validados y están siendo empleados por la industria. Estos sistemas han servido para evaluar de forma satisfactoria numerosos casos de mantenibilidad al desarrollar las partes externas del motor.

El módulo de visualización ha logrado representar los modelos virtuales a una frecuencia interactiva. Se han adecuado las técnicas de visualización a las características especificas de los modelos y al sistema de navegación empleado.

Se ha empleado un volumen envolvente de los objetos que optimiza sustancialmente las técnicas de visualización.

Se ha integrado el sistema de visualización con un sistema háptico cuyas dimensiones son equiparables al tamaño de un motor de avión. Esta característica permite trabajar con el modelo virtual a escala 1:1, posibilitando realizar estudios ergonómicos muy útiles.

Una caracteristica interesante es el empleo de los modelos de CAD que se emplean en la fase de diseño, sin necesidad de realizar ninguna adaptación manual..

Se ha comprobado que la estructura basada en tres procesadores, en la que se reparten los procesos es adecuada para manejar modelos CAD suficientemente

detallados en tiempo real. Los procesos principales son: el lazo de control rápido requerido para restituir fuerzas, el proceso de visualización y el proceso de evaluación de colisiones y penetración.
Se han medido las mejores que se pueden conseguir empleando nuevas técnicas de frustum y oclusión culling.

7 Referencias

[1] B. Blanchard, S.D. Verma, y E.L. Peterson, "Maintainability", John Wiley & Sons Inc., New York, USA, 1995.

[2] D. Luebke, "A Survey of Polygonal Simplification Algorithms", UNC Department of Computer Science Technical Report TR97-045. December 1997.

[3] E. Puppo and R. Scopigno, "Simplification, LOD and Multiresolution Principles and Applications", In Proceedings of EUROGRAPHICS 97. CG Forum, vol. 16, no. 3, 1997.

[4] H. Zhang, D. Manocha, T. Hudson y K. Hoff, "Visibility Culling Using Hierarchical Occlusion Maps", In Proc. Of ACM Siggraph, 1997.

[5] D. Luebke y C. Georges, "Portals and Mirrors: Simple, Fast Evaluation of Potentially Visible Sets", In ACM Interactive 3D Graphics Conference, Monterrey, CA, 1995.

[6] T. A. Funkhouser, "Database Management for Interactive Display of Large Architectural Models", Graphics Interface, pages 1-8, May 1996.

[7] S. Coorg y S. Teller, "Real-Time Occlusion Culling for Models with Large Occluders", Proc. 1997 Symposium on Interactive 3D Graphics, pp. 83-90.

[8] A. Garcia-Alonso, N. Serrano, J. Flaquer, "Solving the Collision Detection problem", IEEE Computer Graphics and Aplications, 13(3):36-43, May 1994.

[9] W. A. McNealy, K. D. Puterbaugh y J.J. Troy, "Six Degree of Freedom Haptic Rendering Using Voxel Sampling", Proceedings of the ACM Siggraph, pp. 401-408, Los Angeles, California, USA, Agosto 1999.

[10] K. Shimoga, "Finger Force and Touch Feedback Issues in Dextrous Telemanipulation: A Survey", Proceedings of NASA-CIRSSE International Conference on Intelligent Robotic Systems for Space Exploration, NASA. 159-178, Greenbelt, Setiembre 1992.

[11] J. Savall., D.Borro, J.J. Gil. y L. Matey, "Description of a Haptic System for Virtual Maintanability in Aeronautics", Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2887-2892, EPFL, Lausanne, Switzerland, September 2002.

ANEXO C

OTRAS PUBLICACIONES

Además de los artículos científicos generados gracias al estudio de esta Tesis, el sistema REVIMA, en el cual se ha integrado toda la investigación realizada sobre las colisiones, ha sido publicado también en una revista interna de la empresa aeronáutica ITP (Industria de TurboPropulsores). A continuación se cita la publicación y se presenta en su formato final.

REVIMA: Realidad Virtual. *Al Vuelo* (comunicación interna). ITP (Industria de TurboPropulsores). No. 42, octubre 2002.


alvuelo
ITP
COMUNICACIÓN INTERNA · Nº 42 · Octubre de 2002
REVIMA:
Realidad Virtual
EPI gestionará el TP400-D6
Panorámica de Farnborough
Zamudio estrena nave

Tecnología

Carlos Tarazona, Informática Técnica

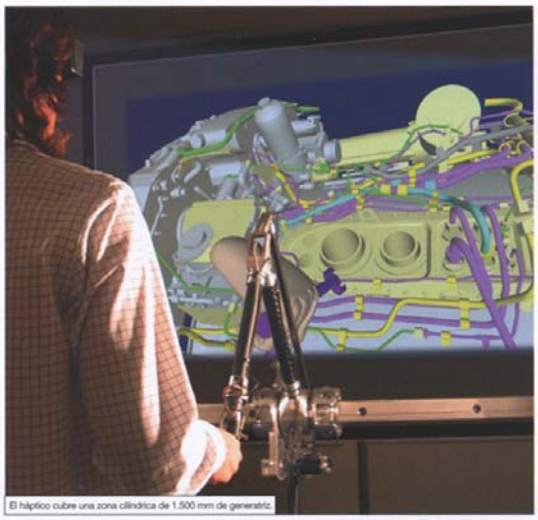
Simula una zona de trabajo 1:1 en un motor de avión

Proyecto REVIMA:

realidad virtual aplicada a la mantenibilidad de componentes externos

Revima proiektuak Errealitate Birtualako teknikak ezartzea helburutzat jotzen du. Funtzean, honek suposatzen du motoreen kanpoko osagaiak robotikaren medio neurtzea mantenimendu aldetik.

El Proyecto REVIMA (1999-2002) tiene como finalidad la aplicación de técnicas de Realidad Virtual a la evaluación de la *mantenibilidad* de componentes externos de motores de aviación, en especial mediante el uso de dispositivos robóticos para la simulación de las sensaciones de pesos y colisiones. El fin último es evitar la construcción, durante la fase de desarrollo, de prototipos físicos cuyo único propósito sea el estudio de la *mantenibilidad*.



El háptico cubre una zona cilíndrica de 1.500 mm de generatriz.

Todo empezó allá por 1998 cuando el CEIT, centro tecnológico experto en robótica que colaboraba desde hacía tiempo con ITP en otras áreas de investigación, hizo una presentación a los departamentos de DPA e ILS, entonces en Ajalvir, ofreciendo la posibilidad de aplicación de técnicas de Realidad Virtual para el trabajo con maquetas electrónicas de componentes externos de motores.

A mediados de 1998 empezó a popularizarse un pequeño dispositivo táctil de sobremesa denominado PHANTOM, que proporcionaba pequeñas fuerzas en el dedo o la mano del operador. El reto estaba planteado: ¿Era posible construir una herramienta integrada formada por un visualizador 3D + un dispositivo táctil de uso industrial para el estudio de *mantenibilidad* de componentes externos, que proporcionara al usuario un comportamiento cercano al real y que permitiera un estudio intuitivo de estos problemas?

El problema al que se quería dar solución era el de la eliminación de la construcción de prototipos físicos para la evaluación de *mantenibilidad*, teniendo en cuenta que el diseño había sido realizado con herramientas CAD. Con este fin, se acometió por parte la Unidad de I+D de Ingeniería y Tecnología, desde 1999 y hasta la fecha, el proyecto REVIMA, con la colaboración de Sener.

Un dispositivo táctil o *háptico* se puede describir como un sistema mecánico para interaccionar con sistemas que en la práctica no se pueden tocar por estar alejados, por ser peligrosos, por ser inaccesibles o simplemente por no existir en la realidad sino como modelo de ordenador. El desarrollo de *hápticos* efectivos ha demostrado ser una tarea complicada, lo cuál hemos podido comprobar durante el desarrollo del proyecto: las extremidades humanas tienen muchos grados de libertad y las sensaciones

táctiles incluyen muchos tipos de estímulos e incorporan un rango dinámico muy amplio de respuestas.

El dispositivo tiene seis grados de libertad, **produciéndose respuesta de fuerza de hasta 1 kg en los tres ejes**

Estos dispositivos pueden considerarse, pues, pequeños robots dotados de sensores y actuadores, que responden con una fuerza determinada a las acciones del utilizador.

Durante 2001 se ha diseñado y construido la primera versión del *háptico*. Dicho dispositivo cubre una zona cilíndrica de 1.500 mm de generatriz, un radio de 350 a 650 mm, con un ángulo abarcado de 120º. La idea es simular una zona de trabajo a escala 1:1 en un motor de avión. El dispositivo tiene 6 grados de libertad, produciéndose respuesta de fuerza de hasta 1 Kg en los 3 ejes. En este primer *háptico* no se van a simular pares, por lo que el usuario deberá *aprender a vivir* en un mundo "sin pares".

Un **resultado** práctico

"Un primer resultado práctico de REVIMA es la utilización del propio software en PC's de sobremesa de altas prestaciones por parte del personal de la sección de Mantenibilidad del departamento de ILS. Actualmente hay 2 puestos instalados, que son réplicas del sistema de la "Sala de Realidad Virtual", donde se ha sustituido el *háptico* por un ratón de 6 grados de libertad. Este sistema permite la evaluación de los casos menos complejos en los estudios de accesibilidad, así como el aprendizaje del propio sistema REVIMA, recurriendo al *háptico* en los casos donde es necesario mayor *realismo*. Estos estudios antes se realizaban mediante los visualizadores de las herramientas CAD, necesitando del orden de varias horas para realizar un estudio no considerado definitivo. Estos estudios se hacen ahora en cuestión de minutos, y si es necesario el uso del haptic, en menos de una hora se puede tener el escenario simulado. Se espera en los próximos meses perfeccionar la técnica, mediante el ajuste del háptico para que la evaluación de la mantenibilidad sea una disciplina establecida en ITP sin necesidad de prototipo físico.

No obstante, todavía queda mucho camino por recorrer. Estamos entrando en un territorio totalmente nuevo, como es la simulación táctil de un entorno industrial, y día a día surgen nuevas ideas y aspectos que se deben mejorar. Esperamos seguir con esta línea de investigación en el futuro cercano. Creo que este proyecto ayudará a que ITP se convierta en una empresa puntera en el sector."

Es el primer *háptico* de este tamaño que se construye en Europa, y aun en el mundo, es difícil encontrar dispositivos que cubran áreas semejantes, en un entorno industrial y con una aplicación práctica.

El montaje definitivo se llevó a cabo en abril de 2002.

Para la instalación del *háptico* se ha habilitado en las oficinas de San Fernando una "Sala de Realidad Virtual". El montaje definitivo se llevó a cabo en abril de 2002, estando la instalación actualmente en fase de validación. En el momento de escribir este artículo se está instalando una pantalla 3D y 2 proyectores en el techo para tener una imagen estéreo pasiva en la pared frontal de gran tamaño, y para evitar la fatiga que produce el uso prolongado de las gafas activas, con el monitor estéreo actualmente operativo. ■

Para la instalación del háptico se ha habilitado en las oficinas de San Fernando una "Sala de Realidad Virtual"