UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ÁREA DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN MATEMÁTICA Y COMPUTACIÓN

ALGORITMO DE DETECCIÓN DE COLISIONES DE OBJETOS VOLUMÉTRICOS BASADO EN EL MODELO DE VISUALIZACIÓN Y MANIPULACIÓN DE VOLÚMENES EVM

Autor:
Lic. Kiara Alexandra Ottogalli Fernández

Tutor:
Dr. Jorge Ernesto Rodríguez Rojas

Bárbula, Abril de 2014

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ÁREA DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN MATEMÁTICA Y COMPUTACIÓN

ALGORITMO DE DETECCIÓN DE COLISIONES DE OBJETOS VOLUMÉTRICOS BASADO EN EL MODELO DE VISUALIZACIÓN Y MANIPULACIÓN DE VOLÚMENES EVM

Autor:
Lic. Kiara Alexandra Ottogalli Fernández

Trabajo presentado ante el Área de Estudios de Postgrado de la Universidad de Carabobo para optar al Título de Magíster en Matemática y Computación.

Bárbula, Abril de 2014

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ÁREA DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN MATEMÁTICA Y COMPUTACIÓN

ALGORITMO DE DETECCIÓN DE COLISIONES DE OBJETOS VOLUMÉTRICOS BASADO EN EL MODELO DE VISUALIZACIÓN Y MANIPULACIÓN DE VOLÚMENES EVM

Autor:
Lic. Kiara Alexandra Ottogalli Fernández

Aprobado en el Área de Estudios de Postgrado de la Universidad de Carabobo por Miembros de la Comisión Coordinadora del Programa:

_____

_____

_____

Bárbula, Abril de 2014

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ÁREA DE ESTUDIOS DE POSTGRADO
MAESTRÍA EN MATEMÁTICA Y COMPUTACIÓN

VEREDICTO

Nosotros, Miembros del Jurado designado para la evaluación del Trabajo de Grado titulado: **ALGORITMO DE DETECCIÓN DE COLISIONES DE OBJETOS VOLUMÉTRICOS BASADO EN EL MODELO DE VISUALIZACIÓN Y MANIPULACIÓN DE VOLÚMENES EVM**, presentado por: **Lic. Kiara Alexandra Ottogalli Fernández** para optar al título de **Magíster en Matemática y Computación**, estimamos que el mismo reúne los requisitos para ser considerado como: **Aprobado**.

_____

_____

_____

Bárbula, Abril de 2014

A Dios,
a mis padres, Isabel y Danilo, y
a mi novio Daniel.

RECONOCIMIENTOS

Primero que todo a Dios, por estar allá arriba iluminando siempre mi camino, y haberme dado la comprensión y fortaleza necesaria para seguir adelante frente a todos los obstáculos.

A mis padres, Isabel y Danilo, por darme la vida, el amor, la educación y los valores que formaron la base de la persona que soy hoy en día, y por el apoyo que me brindaron en todo momento, incluso sin entender lo que estaba haciendo, lo cual ha sido parte de lo que me llevó a seguir adelante. No existen palabras para expresar lo agradecida que estoy por todo lo que me han dado. Los amo.

A mi novio Daniel, por siempre estar a mi lado en los buenos y malos momentos, por ayudarme en todo, incluso a encontrar bugs en el código que yo misma no encontraba, por ser mi amor, mi amigo y mi soporte, quien me dio la fortaleza para seguir adelante incluso en los momentos más difíciles. Cuando no veía la luz al final del camino, tú fuiste mi linterna. Te amo.

A mi amigo Amadís, por ser un súper amigo, quien me ha brindado su amistad incondicionalmente, por haberme ayudado y apoyado en todo este proceso de desarrollar este trabajo, y por estar siempre pendiente.

A mi amigo Jesús, por haberme ayudado tanto en este proceso y tener siempre un buen consejo que dar. Sin ti no hubiera podido llegar hasta aquí.

A mi tutor académico Jorge, por haberme dado la oportunidad de desarrollar este proyecto tan bonito, por haberme guiado siempre por el mejor camino desde el primer momento, por sus aportes, orientación y sobretodo paciencia.

A la Universidad de Carabobo, por ser mi Alma Mater, mi segundo hogar.

A todas aquellas personas que no he mencionado pero, que de alguna forma me han acompañado en mi camino.

¡Muchas gracias!

ÍNDICE GENERAL

CAPÍTULO III

MARCO METODOLOGICO

CAPÍTULO IV

RESULTADOS

INDICE DE FIGURAS

INDICE DE TABLAS

RESUMEN

ALGORITMO DE DETECCIÓN DE COLISIONES DE OBJETOS VOLUMÉTRICOS BASADO EN EL MODELO DE VISUALIZACIÓN Y MANIPULACIÓN DE VOLÚMENES EVM

Autor:
Lic. Kiara Alexandra Ottogalli Fernández

Tutor:
Dr. Jorge Ernesto Rodríguez Rojas

Bárbula, Abril de 2014

El modelo de los vértices extremos es un modelo orientado a la visualización y manipulación de datos volumétricos que proporciona un conjunto aceptable de operaciones de edición, de análisis y morfológicas, sin embargo, no contaba con una operación para la detección de colisiones, la cual es fundamental para las aplicaciones que necesitan realizar un análisis de contacto entre objetos estáticos o en movimiento. En vista de la importancia de la detección de colisiones en cualquier modelo que permita la visualización y manipulación de datos volumétricos, se desarrolló un algoritmo de detección de colisiones el cual complementa el conjunto de operaciones del modelo de los vértices extremos. El diseño se hizo en base al algoritmo para las operaciones booleanas regularizadas, particularmente orientado a la operación de intersección, la cual permite verificar si dos objetos colisionan, pero requiere gran cantidad de tiempo de cómputo, lo cual la hace ineficiente. La implementación del algoritmo fue realizada tomando en cuenta dos estructuras de datos distintas (ABC-*Sorted* y *Trie-Trie*), sobre las cuales fueron realizadas las pruebas. Las pruebas realizadas revelaron que el algoritmo desarrollado es mejor que la operación de intersección para la detección de colisiones, desde un 74% para objetos pequeños, hasta un 82% para objetos grandes. Asimismo, las pruebas revelaron que el algoritmo presenta un mejor comportamiento sobre la estructura ABC-*Sorted*. Por otro lado también se demuestra que el algoritmo de detección de colisiones permite utilizar el EVM como volumen envolvente multiresolución y así disminuir los falsos positivos.

Palabras clave: Algoritmo de detección de colisiones, modelo de los vértices extremos, EVM.

INTRODUCCIÓN

La visualización de volúmenes es un área en el campo de la visualización científica que comprende una serie de técnicas y modelos que permiten representar, visualizar y manipular conjuntos de datos volumétricos con la finalidad de asistir al investigador en el proceso de análisis e interpretación de los mismos.

Para realizar la visualización de volúmenes se han desarrollado diferentes técnicas, sin embargo la mayoría están orientadas exclusivamente a la visualización sin tomar en cuenta la manipulación e interacción con los objetos representados a partir de los datos volumétricos. En la búsqueda de atender esta carencia han sido propuestos diversos modelos de representación que se encargan de integrar la visualización y manipulación de conjuntos de datos volumétricos, entre los cuales se encuentra el Modelo de los Vértices Extremos o Extreme Vertices Model (EVM).

El Modelo de los Vértices Extremos o Extreme Vertices Model (EVM) es un modelo orientado a la visualización y manipulación de datos volumétricos el cual proporciona un conjunto aceptable de operaciones de edición (transformaciones geométricas y operaciones booleanas), de análisis (cálculo de área, volumen, perímetro, etc. y detección de componentes conexas) y morfológicas (erosión, dilatación y esqueletonización). Sin embargo, cuando se piensa en sistemas de visualización complejos con varios objetos en movimiento, existen ciertas operaciones indispensables para la correcta representación y manipulación de dichos objetos, y que no han sido estudiadas a profundidad sobre este modelo. Entre estas operaciones se encuentra la detección de colisiones.

La detección de colisiones es una operación booleana que se encarga de identificar si dos o más objetos en una escena se tocan o se intersectan, lo cual la hace una operación fundamental para las aplicaciones que necesitan realizar un análisis de contacto entre objetos estáticos o en movimiento, como por ejemplo un sistema de simulación quirúrgica.

En vista de la importancia de la detección de colisiones en cualquier modelo que permita la visualización y manipulación de datos volumétricos, el Centro Multidisciplinario de Visualización y Cómputo Científico (CEMVICC) de la Facultad de Ciencias y Tecnología (FaCyT) de la Universidad de Carabobo (UC) ha decidido desarrollar un algoritmo que se encargue de la detección de colisiones entre dos objetos provenientes de conjuntos de datos volumétricos, basado en el Modelo de los Vértices Extremos. Para este propósito, se diseñará e implementará una nueva estructura de datos para la representación de los objetos modelados con EVM y se usará el lenguaje de programación C++ con OpenGL (Open Graphics Library) bajo el sistema operativo Linux en su distribución Ubuntu.

El trabajo especial de grado se encuentra estructurado en 4 capítulos. El Capítulo I habla sobre el problema de investigación, contiene el planteamiento y formulación del problema, los objetivos generales y específicos, y la justificación del problema. En el Capítulo II se aborda el marco teórico referencial, el cual contiene los antecedentes, bases teóricas y definición de términos básicos. El Capítulo III contiene el marco metodológico. En el Capítulo IV se presenta el algoritmo de detección de colisiones para el EVM y las pruebas sobre su funcionamiento. Finalmente se presentan las conclusiones, recomendaciones y referencias bibliográficas.

CAPÍTULO I

EL PROBLEMA

Planteamiento del Problema

La visualización científica es una disciplina en la cual se aplican métodos gráficos, con la finalidad de generar representaciones visuales que ayuden a la interpretación de datos científicos. Estas representaciones visuales permiten una mejor percepción y comprensión de procesos físicos, conceptos matemáticos y otros fenómenos cuantificables (Fosdick, L., Jessup, E., Schauble, C. y Domik, G., 1996: 295), los cuales, posiblemente, dificultan su comprensión sin una visualización adecuada.

Rodríguez, J. (2004: vii) afirma que un área de investigación muy importante dentro del campo de la visualización científica es la visualización de volúmenes, la cual se encarga de la representación, manipulación y visualización de conjuntos de datos volumétricos. Esta área de investigación tiene muchas aplicaciones en diferentes campos como la medicina, la exploración sísmica, la dinámica de fluidos, el estudio de fenómenos atmosféricos, la exploración petrolera, entre otros.

En el mismo estudio, Rodríguez, J. afirma que la mayoría de las técnicas desarrolladas en el área de visualización de volúmenes, han sido concebidas únicamente para propósitos de visualización, sin tomar en cuenta la necesidad de interactuar con el objeto gráfico que representa el conjunto de datos volumétricos visualizados. Para solucionar esto, se han propuesto diferentes modelos de

representación o *frameworks*, entre ellos el *Data Conversion Approach* e *Independent Representation Approach*, que permiten integrar la visualización y la manipulación de datos volumétricos.

Un modelo orientado a la visualización y manipulación de datos volumétricos debe permitir la visualización de los datos y al mismo tiempo contar con un conjunto de operaciones que faciliten la interacción y la modificación de los objetos gráficos usados para desplegar los datos visualmente (Rodríguez, J., 2004: 38).

El Modelo de los Vértices Extremos o Extreme Vértices Model (EVM) es definido por Rodríguez, J. como un modelo de representación basado en pseudo-poliedros ortogonales que permite la obtención de un modelo de fronteras para la visualización de datos volumétricos, y que, al mismo tiempo, cuenta con un conjunto de operaciones que facilitan la interacción y modificación de los objetos gráficos en base a los cuales se representan los datos visualmente.

Rodríguez, J., (2004: 84) demuestra que el EVM define un esquema de representación de datos volumétricos competitivo con sus contrapartes *Octree* y *Semiboundary* en términos de memoria y disponibilidad de acceso a los datos volumétricos, esto mediante el uso de estructuras de datos eficientes para el almacenamiento y manejo de los mismos. Además, el EVM es adecuado para el renderizado de superficies y puede ser utilizado para filtrar, editar y analizar los objetos volumétricos de manera eficiente mediante el uso de algoritmos diseñados propiamente para el EVM.

Las operaciones que han sido desarrolladas para el EVM pueden ser clasificadas en tres tipos: (1) operaciones de edición, entre las cuales se encuentran las transformaciones geométricas (rotación, translación y escalado) y operaciones booleanas (unión, intersección y diferencia), (2) operaciones de análisis (cálculo de área, volumen, perímetro y detección de componentes conexas) y (3) operaciones morfológicas (erosión, dilatación y esqueletonización) (Rodríguez, J., 2004: 87).

Sin embargo, Rodríguez, J. (2004: 152) explica que se debe tomar en cuenta que el conjunto de operaciones que han sido desarrolladas para el EVM es limitado. Existen ciertas operaciones consideradas como indispensables para la manipulación de objetos estáticos o en movimiento que no han sido estudiadas en profundidad para el EVM, entre ellas, es de especial interés la detección de colisiones entre objetos volumétricos, ya que permite establecer si dichos objetos entran en contacto.

La detección de colisiones es una operación booleana que se encarga de identificar si dos o más objetos en una escena se tocan o se intersectan. Esta operación es fundamental para aquellas aplicaciones que implican el análisis de contacto entre objetos estáticos y/o en movimiento, entre las cuales se incluyen los sistemas de simulación quirúrgica, simulaciones físicas, robótica, prototipos virtuales, diseño asistido por computadoras, manufactura, ensamblaje y des-ensamblaje, planificación de movimientos y animación, entre otras (Ericson, C., 2005:1).

El desarrollo de un método para la detección de colisiones de objetos volumétricos modelados mediante el EVM busca, mediante el uso de estructuras de datos sencillas en cuanto a su construcción y manipulación, y el lenguaje de programación adecuado, completar el conjunto de operaciones del modelo, logrando

así incrementar la robustez del mismo como modelo de visualización y manipulación de objetos volumétricos.

De acuerdo a lo planteado anteriormente se propone el desarrollo de un algoritmo para la detección de colisiones, fundamentado en una nueva estructura de datos que permita mejorar el conjunto de operaciones de manipulación de datos volumétricos del EVM.

Formulación del Problema

Ahora bien, ¿Cómo se puede efectuar algorítmicamente la detección de colisiones entre dos objetos volumétricos en el modelo de representación EVM?

Objetivos de la Investigación

Objetivo General

Desarrollar el algoritmo de detección de colisiones entre dos objetos volumétricos para incorporarla al conjunto de operaciones de manipulación de volúmenes del modelo de representación EVM.

Objetivos Específicos

– Analizar el modelo de representación de volúmenes EVM.

– Diseñar la estructura de datos adecuada para el almacenamiento de objetos volumétricos modelados mediante el EVM.

– Crear un algoritmo que permita realizar la detección de colisiones entre dos objetos volumétricos modelados mediante el EVM.

– Realizar pruebas con el algoritmo creado.

– Evaluar el algoritmo creado.

Justificación del Estudio

En el trabajo realizado por Rodríguez, J. (2004: 152) se propone la creación de diferentes operaciones de manipulación de objetos volumétricos para el EVM, entre ellas se encuentra el cómputo de esqueletos lineales para sólidos, el cálculo del género de los poliedros y la detección de colisiones. Por su parte, Pérez-Aguila, R. (2006: 177) explica teóricamente cómo puede ser realizada la detección de colisiones entre objetos volumétricos modelados con el EVM tomando como base las operaciones booleanas regularizadas definidas para el modelo, sin embargo, no presenta un algoritmo para tal fin. La creación de un algoritmo para la detección de colisiones entre objetos volumétricos representados mediante el EVM responde a las propuestas hechas en estos trabajos.

La creación de un algoritmo para la detección de colisiones complementaría la gama de operaciones existentes en el modelo de visualización y modificación de

objetos volumétricos EVM, por lo cual contribuye desde el punto de vista práctico, a la completitud de un modelo que ha demostrado ser competitivo para la visualización y manipulación de datos volumétricos (Rodríguez, J., 2004: 128), con el cual se podrían desarrollar aplicaciones que necesiten el análisis intensivo de contacto entre diferentes objetos de la escena.

Desde el punto de vista científico, el desarrollo de un algoritmo para detectar colisiones entre objetos volumétricos entre EVMs permitiría alcanzar mejoras en la investigación y desarrollo de nuevas aplicaciones de visualización científica para el estudio de fenómenos cuantificables que necesiten el análisis de contacto de objetos volumétricos.

Debido a que Perez-Aguila, R. (2006: 88) presenta en su trabajo una estructura de datos que no permite realizar el ordenamiento de un objeto modelado con el EVM, se implementará una estructura de datos más sencilla que permita realizar tanto la detección de colisiones como dicho ordenamiento, debido a que es una operación indispensable para poder transformar el objeto en un B-Rep y realizar su renderizado. Por estas razones, se programará una librería que implementará la estructura de datos para el manejo de objetos modelados con el EVM, la cual será ofrecida bajo la licencia GPL (General Public License), lo cual permitirá su libre estudio, modificación, expansión y optimización por parte de la comunidad científica.

El desarrollo de un algoritmo de detección de colisiones entre objetos volumétricos modelados con el EVM, contribuiría en diferentes áreas, a la realización de sistemas de simulación que requieran el estudio de contacto entre objetos. Por ejemplo, en el área médica se podría usar el EVM como base para la creación de sistemas de simulación quirúrgica, en los cuales se necesita el estudio del contacto

entre diferentes objetos volumétricos para simular con cierto grado de realismo y exactitud ciertas situaciones, entre ellas, el contacto de un bisturí con una parte del cuerpo humano virtual, el comportamiento de las articulaciones, o verificar si una prótesis dental tiene el tamaño correcto para una persona específica antes de proceder a la implantación real de la misma.

Por otra parte, en el área industrial, puede ser beneficioso el desarrollo de un algoritmo de detección de colisiones para objetos volumétricos basado en el modelo EVM, ya que permitiría realizar aplicaciones de simulación de diferentes fenómenos para ser estudiados previamente a su realización, como por ejemplo, una aplicación que permita verificar si varios objetos pueden ser ensamblados correctamente para su posterior manufactura.

CAPÍTULO II

MARCO TEÓRICO REFERENCIAL

Este capítulo tiene como objetivo situar el problema en el contexto del conocimiento existente, orientando el proceso de investigación, presentando un breve resumen de los antecedentes en el área de la detección de colisiones que han llevado a la realización de este estudio. Asimismo, se tomarán en cuenta las bases teóricas que serán utilizadas en la creación de un algoritmo de detección de colisiones para el modelo EVM. Finalmente, se definirán algunos términos básicos cuyo conocimiento es necesario para el estudio.

Antecedentes

La detección de colisiones es una de las tareas de mayor costo computacional en el área de visualización científica, en consecuencia ha sido motivo de investigación por varias décadas, situación que ha generado diferentes enfoques para abordar la solución del problema. A continuación, se presentará cronológicamente un resumen del estado del arte con respecto a la detección de colisiones entre objetos volumétricos.

El trabajo de investigación titulado **Image-based Collision Detection and Response between Arbitrary Volume Objects**, realizado entre la Universidad Grenoble, INRIA (*Institut National de Recherche en Informatique et en Automatique*) y LJK-CNRS (*Laboratoire* Jean Kuntzmann – *Centre National de la Recherche Scientifique*) por F. Faure, S. Barbier, J. Allard y F. Falipou (2008) propone un

método que se basa en la minimización de volúmenes basada en imágenes para realizar la detección de colisiones entre objetos delimitados por superficies triangulares, la cual puede ser aplicada tanto en objetos deformables como no deformables sin necesidad de pre-cómputo. Este algoritmo se encarga de encontrar los pares de *Bounding Boxes* (BBs) que se superponen, luego la detección de una intersección se realiza en la unidad de procesamiento gráfico o *Graphic Processing Unit* (GPU), donde los polígonos son rasterizados en tres direcciones ortogonales y almacenados en capas de imágenes o *Layered Depth Images* (LDI) para luego detectar las colisiones basado en la profundidad de los pixeles y su orientación normal. El aporte de este trabajo radica en que el método propuesto reduce los volúmenes sobre los cuales se hace la prueba de detección de colisiones en la GPU, sin importar si el objeto es deformable o no. Esta reducción de los volúmenes sobre los cuales se harán las pruebas de colisión puede ser lograda mediante la aplicación del modelo de los vértices extremos, ya que reduce la complejidad del objeto volumétrico, y por lo tanto debe reducir la complejidad de la detección de colisiones.

H. Jang y J. Han (2008) realizaron un trabajo en la Universidad de Korea, titulado **Fast Collision Detection using the A-Buffer**, en el cual presentaron un algoritmo para la detección de colisiones entre múltiples objetos volumétricos. Este algoritmo no requiere preprocesamiento y puede manejar objetos en movimiento, incluyendo objetos deformables y rompibles e incluso puede calcular las colisiones de un objeto consigo mismo. El algoritmo propuesto realiza una prueba de superposición entre los *Axis Aligned Bounding Boxes* (AABBs) de dos objetos en la unidad central de proceso o *Central Processing Unit* (CPU). Si se detecta una intersección, se traslada al GPU para calcular los conjuntos potencialmente en colisión o *Potentially Colliding Sets* (PCSs). Cada PCS es un conjunto de dos triángulos, uno del primer objeto y otro del segundo objeto. Una vez calculados los PCSs, con asistencia del CPU se realiza la prueba de intersección de triángulos para obtener los puntos de intersección. Este

algoritmo se diferencia de sus predecesores, CULLIDE y flujos de AABBs en que un PCS tiene solo dos triángulos, mientras dichos métodos trabajan dos o más triángulos. Este trabajo aborda la posibilidad de utilizar un proceso compuesto entre la CPU y la GPU de la máquina, además de incluir la posibilidad de procesar flujos de AABBs para mejorar el tiempo de la detección de colisiones. De este trabajo se toma la técnica del PCS y los flujos de AABBs, de una forma más general, adaptada al modelo de los vértices extremos.

S. Kim, S. Redon y Y. J. Kim (2008) realizaron un trabajo titulado **Continuous Collision Detection for Adaptive Simulations of Articulated Bodies** entre la Universidad Ewha Womans y el INRIA, en el cual presentaron un algoritmo de detección de colisiones para la simulación adaptativa de cuerpos articulados. Este algoritmo se realiza en dos pasos, en el primer paso se realiza el descarte mediante el uso de los AABBs de los cuerpos, para obtener solamente aquellos que potencialmente colisionan, los cuales podrían estar dentro de un mismo cuerpo articulado, para luego, en el segundo paso calcular el contacto exacto mediante jerarquías de *Oriented Bounding Boxes* (OBBs). Este trabajo tiene como aporte la introducción de la detección de colisiones que pueden ocurrir sobre un mismo objeto volumétrico en caso de que esto ocurra.

J. Jiménez y R. Segura (2008) presentaron un trabajo realizado en la Universidad de Jaén, titulado **Collision Detection between Complex Polyhedra**, en el cual representan poliedros complejos mediante una cobertura simplicial, cuyos elementos llamados simplejos son clasificados mediante una descomposición espacial llamada Tetra-Tree y una jerarquía de volúmenes envolventes o *Bounding Volume Hierarchy* (BVH) basada en tetraedros y esferas. El aporte de este trabajo es que la representación presentada permite reducir la cantidad de pruebas de intersección entre

los poliedros complejos. De este trabajo se aplica la reducción de las pruebas de intersección entre los objetos volumétricos.

X. Zhang y Y. J. Kim (2007) realizaron en la Universidad Ewha Womans un trabajo titulado **Interactive Collision Detection for Deformable Models using Streaming AABBs**, en el cual presentan un algoritmo en el cual emplean paralelismo para disminuir el tiempo de cómputo de la detección de colisiones mediante el procesamiento de flujos de datos. El algoritmo se desarrolla en tres fases. En la primera fase se organiza un flujo de AABBs y se guarda su información en el GPU. En la segunda fase se realiza una prueba global de superposición de los AABBs, de encontrarse alguna intersección se realiza la prueba entre todas las posibles combinaciones de sub-objetos entre los dos modelos y se guardan los resultados booleanos, que se codifican para mejorar el tiempo de envío de los datos del GPU al CPU, finalmente se realiza la prueba de intersecciones entre los triángulos de ambos modelos. En la tercera fase se actualizan los flujos de AABBs de los objetos deformados. El aporte de este trabajo a la investigación es que realiza varias pruebas de detección de colisiones entre los AABBs de los objetos, y de esta forma, reducen el tiempo de detección de colisiones. De este trabajo se utiliza la técnica de dividir el proceso de detección de colisiones en varias fases, partiendo de la más sencilla para realizar descartes rápidos entre los volúmenes mediante el uso de AABBs.

T. Larsson y T. Akenine-Möller (2006) realizaron entre las universidades Mälardalen y Lund un trabajo titulado **A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection**, en el cual presentaron una nueva BVH dinámica basada en AABBs para lograr realizar una rápida y generalizada detección de colisiones entre objetos rompibles en movimiento. El proceso de detección de colisiones mostrado en este trabajo se realiza en dos fases. La primera fase es de

actualización, la cual comienza desde los nodos activos más internos de un tiempo anterior, construye el volumen envolvente o *Bounding Volume* (BV) mínimo a partir de la información de la superficie, lo cual es llamado ajuste de los nodos, mientras que los nodos marcados como no activos en la última fase de detección de colisiones se invalidan. La segunda fase es la fase de detección de colisiones, en la cual se recorren de forma paralela los pares de BVHs para ordenar las primitivas que podrían colisionar. Este trabajo aporta una nueva jerarquía de AABBs que permite ser recorrida de forma paralela para ir descartando los AABBs que no colisionan y así, reducir el trabajo a realizar. De este trabajo se intuye la posibilidad de aplicar el modelo de los vértices extremos para la obtención del BV mínimo a partir de la superficie de los volúmenes, con la finalidad de realizar la detección de colisiones.

R. Pérez-Águila (2006) en su tesis doctoral, **Orthogonal Polytopes: Study and Application**, presentada en la Universidad de las Américas de México, realiza un estudio sobre la representación de polítopos pseudo ortogonales en n dimensiones mediante el modelo de los vértices extremos. En este trabajo Pérez-Águila presenta algunas aplicaciones prácticas del modelo, entre las cuales propone su utilidad para realizar la detección de colisiones, y presenta de manera general una idea sobre como pueden ser aplicadas las operaciones regularizadas del modelo para este fin, lo cual es un aporte que se usará en la creación del algoritmo de detección de colisiones entre objetos volumétricos modelados mediante el modelo de los vértices extremos.

P. Liu, X. Shen, N. Georganas y G. Roth (2005) realizaron un trabajo en la Universidad de Ottawa, titulado **Multi-resolution Modeling and Locally Refined Collision Detection for Haptic Interaction**, en el cual introducen una nueva forma de BVH llamada *Active Bounding* Trees (AB-tree), que está basada en la BVH de AABBs. En esta nueva representación jerárquica, cada nodo tiene tres campos

adicionales: (1) Un apuntador a una lista de índices, (2) Un apuntador al nodo padre y (3) El estatus del nodo (activo, inactivo o deformado). El algoritmo de detección de colisiones presentado en este trabajo cuenta con dos fases, una fase de pre-procesamiento y una fase en tiempo de ejecución. En la fase de pre-procesamiento se codifica el AB-Tree en mallados *Space Partitioned Multi-resolution* (SPM), lo cual aligera el proceso de construcción de la BVH. En la segunda fase, se refinan dinámicamente los mallados en las áreas que posiblemente colisionan y se reajustan los AB-Trees de los modelos para luego pasar a calcular la detección de colisiones entre pares de objetos. El aporte de este trabajo es una nueva jerarquía que permite un pre-procesamiento que disminuye su tiempo de construcción, y también permite un refinamiento de las áreas que se determina son de interés.

H. K. Kim, L. Guibas y S. Y. Shin (2005) realizaron un trabajo en conjunto por Samsung Electronics, Universidad Stanford y el Instituto de Ciencia y Tecnología Avanzada de Korea, titulado **Efficient Collision Detection Among Moving Spheres with Unknown Trajectories**, en el cual presentaron un algoritmo para la detección de colisiones entre múltiples esferas con trayectorias desconocidas, aplicando el concepto de envoltura variante en el tiempo o *time-varying bound*. Este algoritmo calcula el BV de cada esfera dentro de la simulación, que en principio es una esfera con el mismo centro y radio de la esfera original, pero a medida en que transcurre el tiempo va aumentando su radio. Cuando se detecta una colisión entre dos de las esferas en el ambiente, se regresa el BV a su estado original y se vuelve a comprobar si las esferas colisionan. Kim et al. lograron demostrar que el algoritmo permite realizar la simulación de miles de esferas en movimiento con trayectorias desconocidas a una tasa interactiva. Este trabajo hace un aporte importante para la detección de colisiones entre miles de objetos en una escena mediante una técnica que varía el tamaño del BV que envuelve a los objetos que posiblemente colisionan.

B. Heidelberger, M. Teschner y M. Gross (2004) en la Escuela Politécnica Federal de Zúrich realizaron un trabajo titulado **Detection of Collisions and Self-collisions Using Image-space Techniques**, en el cual proponen una nueva técnica para la detección de colisiones basado en el espacio de imagen, que puede ser aplicada para detectar las colisiones entre objetos y de un objeto consigo mismo. Como la técnica propuesta no requiere una etapa de pre-procesamiento también puede ser aplicada en ambientes con objetos deformables. El algoritmo propuesto se compone por tres etapas. En la etapa 1, se calcula el volumen de interés o *Volume of Interest* (VoI), que en el caso de una colisión de un objeto con él mismo, es un AABB del propio objeto; y en el caso de una colisión entre objetos distintos, es la intersección entre los AABBs de los objetos implicados. Si el VoI no es vacío, se procede a la siguiente etapa. En la etapa 2, se calculan las imágenes de profundidad en capas o *Layered Depth Images* (LDI) de los objetos dentro del VoI. En la etapa 3 se realiza la detección de colisiones, en la cual presentan dos aproximaciones, una que utiliza el CPU para generar un conjunto ordenado de LDI, y otra en la cual no se ordena el conjunto de LDI y se utiliza el GPU. Este trabajo aporta una técnica que permite verificar mediante el uso conjunto de la CPU y la GPU la detección de colisiones entre objetos o sobre un objeto consigo mismo.

D. James y D. Pai (2004), realizaron un trabajo titulado **BD-Tree Output-Sensitive Collision Detection for Reduced Deformable Models**, en las universidades Carnegie Mellon y Rutgers, en el cual desarrollaron una BVH llamada *Bounding Deformation Tree* (BD-Tree), la cual es una jerarquía basada en esferas, en la cual no es necesario que una esfera en un nodo del árbol envuelva completamente a las esferas de sus nodos descendientes, pero sí los puntos de la geometría subyacente, en consecuencia, el espacio de las esferas puede ser más pequeño, lo cual es beneficioso cuando se aplica la detección de colisiones en el árbol. Este trabajo introduce la posibilidad de utilizar BVHs en las cuales los nodos de la jerarquía no

deben contener completamente a sus nodos hijo, lo cual hace que la detección de colisiones se vuelva más exacta.

M. Teschner et al. (2004) llevaron a cabo un estudio titulado **Collision Detection for Deformable Objects**, en conjunto con la Universidad de Freiburg, la Escuela Politécnica Federal de Zúrich, WSI/GRIS, la Universidad Tübingen, Universidad de Geneva, GRAVIR/IMAG, INRIA Grenoble, la Universidad Bonn y el Instituto de Computación Gráfica Fraunhofer, en el cual se habla de las técnicas para realizar la detección de colisiones entre objetos deformables. En este estudio presentan las técnicas basadas en el espacio del objeto, entre ellas, las BVHs, los campos de distancia y el particionamiento espacial. Igualmente, presentan las técnicas basadas en el espacio de imagen. Con respecto a las BVHs, Teschner et al., afirman que la aproximación es eficiente siempre y cuando se elija correctamente el BV básico, así como también, la estrategia para actualizar la jerarquía. Con respecto a los campos de distancia, se afirma que no sólo son eficientes para la detección de colisiones sino también para el cálculo de la profundidad de la colisión, lo cual es necesario para el proceso de respuesta de la detección de colisiones. Con respecto al particionamiento espacial, se afirma que la técnica es eficiente en ambientes de simulación con objetos deformables. Finalmente, con respecto a las técnicas de detección de colisiones basadas en el espacio de imágenes se afirma que son eficientes para ambientes con objetos geométricamente complejos. El aporte de este estudio es un resumen de las técnicas existentes hasta el momento para la detección de colisiones entre objetos deformables, en la cual se resumen sus ventajas y desventajas, de manera que es posible tomar en cuenta que técnica es mejor en cada caso, y que se puede mejorar de ésta.

Z. Fan, H. Wan y S. Gao (2003) realizaron en la Universidad Zhejiang un trabajo titulado **IBCD: A Fast Collision Detection Algorithm Based on Image-Space using OBB**, en el cual propusieron un algoritmo que combina diferentes aproximaciones para la detección de colisiones en el espacio del objeto y en el espacio de imágenes para realizar la detección de colisiones de superficies no convexas. Este algoritmo procede en dos fases, una primera fase de pre-procesamiento y otra de detección de colisiones en tiempo de ejecución. En la primera fase, se descompone cada superficie no convexa de un objeto en un conjunto de piezas convexas, las cuales luego son organizadas en un árbol binario jerárquico en el cual cada hoja representa una de esas piezas convexas. Luego, para cada una de esas piezas convexas se construye un OBB para ser utilizado en el proceso de renderizado. En la segunda fase, se eliminan todos los pares de objetos en la escena que no se intersectan, mediante la técnica de *barrido y poda* (*Sweep and Prune*) para luego verificar las colisiones entre pares de objetos, cuyas jerarquías se recorren de forma simultánea hasta verificar si sus OBBs se superponen. Si esto ocurre, las piezas convexas de dichos OBBs se renderizan para luego realizar la detección de colisiones en el espacio de imagen. El aporte de este trabajo es la posibilidad de descomponer un objeto no convexo en sub-objetos convexos para la detección de colisiones, y también la realización de la detección de colisiones de forma compuesta en el espacio del objeto y en el espacio de imágenes.

N. Govindaraju, S. Redon, M. C. Lin y D. Manocha (2003) realizaron un trabajo titulado **CULLIDE: Interactive Collision Detection Between Complex Models in Large Environments using Graphics Hardware** en la Universidad de Carolina del Norte, en el cual presentaron el algoritmo CULLIDE, que utiliza el *hardware* gráfico para detectar colisiones entre múltiples objetos en movimiento, ya sean rígidos, deformables o rompibles en ambientes amplios. Este algoritmo calcula el PCS del conjunto total de objetos en la escena mediante consultas de oclusión en el espacio de

imagen, para luego calcular el PCS de los sub-objetos (*Bounding Boxes*, triángulos o grupos de los mismos) del primer PCS y finalmente realizar una prueba exacta de intersección de triángulos para determinar cuáles triángulos colisionan. Este algoritmo aporta una detección de colisiones basada en el espacio de imágenes que permite reducir la cantidad de comparaciones, disminuyendo el grupo de objetos en un grupo de objetos que probablemente colisionan.

G. Baciu y W. Wong (2003) realizaron un trabajo titulado **Image-Based Techniques in a Hybrid Collision Detector** en la Universidad Politécnica de Hong Kong, en el cual desarrollaron un algoritmo híbrido para realizar la detección de colisiones entre dos objetos volumétricos. Este algoritmo disminuye la carga computacional que implica la detección de una colisión distribuyendo el trabajo en las diferentes etapas del *pipeline* de visualización, descartando la mayor cantidad de pares de triángulos en el espacio del objeto para luego aprovechar el *hardware* gráfico para realizar la detección de colisiones en el espacio de imagen. El algoritmo, llamado HYCODE, primero verifica si existe un plano que separe a los dos objetos en el espacio, si este plano no existe, calcula el AABB más ajustado a cada objeto y lo proyecta sobre un plano, luego calcula la región mínima de superposición o *Minimum Overlapping Region* (MOR) de las proyecciones de los dos AABBs en el plano, para luego realizar la prueba de intersección haciendo uso del Z-Buffer y el Stencil Buffer. Este trabajo representa uno de los primeros aportes que incluyen la utilización del *hardware* gráfico para realizar la detección de colisiones entre objetos.

Bases Teóricas

Para una mejor comprensión de este Trabajo Especial de Grado hay que tomar en cuenta dos puntos importantes sobre los cuales se sustenta el diseño del algoritmo de

detección de colisiones: La detección de colisiones y El Modelo de los Vértices Extremos o *Extreme Vertices Model* (EVM). En el primer punto se explica brevemente en que consiste la detección de colisiones entre objetos, los factores que influyen en el diseño de un algoritmo para la detección de colisiones y las técnicas existentes para la detección de colisiones. En el segundo punto se define el EVM y se presentan todas sus propiedades, las cuales serán usadas en el desarrollo del algoritmo de detección de colisiones.

Detección de Colisiones

Ericson, C. (2004) define la detección de colisiones como una operación booleana que se encarga de identificar si dos o más objetos en una escena se tocan o se intersectan. Es parte del proceso de determinación de colisiones, proceso en el cual no solo se verifica si se genera una colisión, sino que también responde cuándo y dónde se genera una colisión.

La detección de colisiones es muy importante en las áreas de computación gráfica, visualización científica, robótica, animación por computadora, simulaciones de fenómenos físicos, y cualquier área en la que sea fundamental verificar la intersección de dos o más objetos, por estas razones ha sido objeto de estudio por décadas.

Diseño de un Sistema de Detección de Colisiones

Para diseñar un sistema de detección de colisiones, Ericson, C. (2005: 7) expresa que existen diferentes factores que deben ser tomados en cuenta:

1. La representación del dominio de la aplicación:

El diseño de un algoritmo para la detección de colisiones debe tomar en cuenta las diferentes formas de representar un modelo, al igual que considerar ciertos conocimientos específicos de la aplicación a desarrollar que pueden ser utilizados para realizar implementaciones especializadas que consigan mejorar el rendimiento del algoritmo.

Las representaciones de objetos, se pueden clasificar en representaciones poligonales y no poligonales.

La base de una representación poligonal es el polígono (generalmente el triángulo), a partir del cual se forman objetos. Las representaciones no poligonales más comunes son: (a) las superficies implícitas, en las cuales la frontera del objeto viene definida mediante una función implícita, (b) las superficies paramétricas, en las cuales la frontera del objeto viene definida por una ecuación paramétrica, y (c) la geometría constructiva de sólidos o *Constructive Solid Geometry* (CSG), en la cual las primitivas geométricas como esferas, cajas y cilindros se combinan mediante las operaciones de teoría de conjuntos (unión, intersección y diferencia entre otras) para formar nuevos objetos.

2. Los tipos de consultas.

El algoritmo debe tomar en cuenta los tipos de consultas que deben ser respondidas. Entre las consultas se tiene el caso más sencillo, el cual implica responder si dos objetos se tocan o no, encontrar las partes que se intersectan, algún

punto de intersección, todos los puntos de intersección, la profundidad de penetración, los puntos más cercanos y el tiempo en el que ocurrirá una colisión. Igualmente, estas consultas pueden ser exactas, o inexactas dado cierto valor de tolerancia.

3. Los parámetros del ambiente de simulación.

Entre los parámetros de simulación se encuentran el número de objetos, el tamaño de los objetos, si el movimiento de los objetos es simultáneo o secuencial, continuo o discreto, si un objeto puede penetrar otro y si los objetos son flexibles o rígidos.

4. El rendimiento.

Dependiendo del tipo de aplicación se debe tomar en cuenta el tiempo de respuesta del algoritmo y la cantidad de memoria que ocupan las estructuras de datos utilizadas. En las aplicaciones de tiempo real se hace énfasis en mejores tiempos de ejecución, mientras en aplicaciones en las cuales se necesita exactitud en los cálculos el factor tiempo no es tan importante.

5. La robustez.

En la detección de colisiones la robustez del algoritmo es sumamente importante, esto debido a que pueden existir cálculos numéricos y configuraciones geométricas difíciles de manejar que pueden llevar a un mal funcionamiento del algoritmo. Se deben tomar en cuenta dos tipos de problemas: (a) Numéricos, debidos al uso de

variables de precisión finita, y (b) Geométricos, en los cuales se debe asegurar la consistencia geométrica.

6. Fácil implementación y uso.

En este caso se deben tomar en cuenta la complejidad del algoritmo, los casos especiales que pudieran ocurrir, las variables que podrían llevar a errores numéricos y otras limitaciones. Igualmente, se debe evaluar la generalidad del sistema, si puede manejar objetos de diferentes tamaños y el tiempo de pre-procesamiento para la construcción de las estructuras de datos asociadas a la detección de colisiones, factores que pueden ser relevantes en la experimentación y productividad al utilizar el sistema.

Clasificación de las Técnicas para la Detección de Colisiones

Las técnicas para realizar el proceso de detección de colisiones pueden ser clasificadas, dependiendo del espacio en el cual se realice el procesamiento en dos grupos: Técnicas basadas en el espacio del objeto y técnicas basadas en el espacio de imagen.

Las técnicas basadas en el espacio del objeto se basan en la geometría del objeto, y la detección de colisiones se realiza procesando pares de primitivas geométricas, como por ejemplo triángulos. Debido al alto costo computacional que implica realizar una prueba de intersección entre todas las posibles combinaciones de pares de primitivas, se han desarrollado diferentes técnicas para probar si dos objetos potencialmente colisionan. Entre estas técnicas las más relevantes son las basadas en

volúmenes envolventes (BVs: *Bounding Volumes*), campos de distancia y particionamiento espacial.

Para realizar la detección de colisiones es usual envolver cada uno de los objetos en la escena en BVs, entre los cuales se realizan pruebas de descarte para detectar si estos no colisionan. Esto mejora significativamente el desempeño del algoritmo ya que en caso de no existir una colisión no se realizan las pruebas sobre la geometría del objeto en sí.

Ahora bien, de detectarse una posible colisión entre los BV de dos objetos, se deben realizar pruebas entre las primitivas que los componen. Esta situación puede mejorarse al construir una jerarquía de volúmenes envolventes (BVH: *Bounding Volume Hierarchies*), la cual se logra mediante el cálculo, en una etapa de pre-procesamiento, de un árbol de BVs. Para ello se realiza una partición recursiva del conjunto de primitivas que pertenecen a cada BV en dos subconjuntos, a cada uno de los cuales se le calcula su respectivo BV, y se conectan al BV del cual se generaron como hijos. Este proceso continua hasta que las hojas del árbol sean los BVs de las primitivas del objeto.

Para definir una BVH se han propuesto diferentes tipos de BV, entre ellos se encuentran el AABB, el OBB, la esfera, el polítopo discreto orientado o *Discrete Oriented Polytope* (DOP), la *Convex Hull*, *Spherical Shell*, el prisma, el cilíndro y la intersección entre ellos (Teschner, M. et al., 2005).

Para detectar una colisión entre dos objetos, se debe realizar el recorrido desde el nodo raíz hasta los nodos hoja (*top-down*), realizando las pruebas entre los pares de

nodos de ambos árboles para encontrar cuáles BVs se superponen. Si los nodos que se superponen son hojas, entonces se realiza la prueba de intersección entre sus primitivas. Si los nodos que se superponen son una hoja y un nodo interno, se realiza la prueba entre el nodo hoja y los hijos del nodo interno con el cual se intersecta. Si ambos nodos que se superponen son nodos internos, entonces se intenta disminuir la cantidad de cómputo realizando la prueba entre el nodo de menor volumen y los hijos del nodo de mayor volumen.

Los campos de distancia, según Huang, J. et al. (2001), se definen como campos espaciales de distancias escalares a una superficie geométrica o forma. Cada elemento en un campo de distancia especifica su distancia mínima a la superficie del objeto. Estas distancias pueden ser positivas o negativas para distinguir el exterior del interior del objeto, de forma que cuando la distancia es negativa se intuye que es el exterior del objeto y cuando la distancia es positiva se intuye que es el interior del objeto. Mediante la utilización de campos de distancia, se puede realizar la detección de colisiones verificando si la distancia entre los campos es igual a cero, caso en el cual existe una colisión.

El particionamiento espacial consiste en dividir el espacio tridimensional en regiones. Verificando si los objetos se superponen en la misma región del espacio se disminuye la cantidad de pruebas, ya que dos objetos se intersectan si y solo si ocupan la misma región en el espacio. Entre las técnicas de particionamiento espacial se encuentran los *grids* uniformes, los *grids* jerárquicos y los árboles tales como octree, bintree, Kd-tree, entre otros. (Ericson, C., 2004).

Las técnicas basadas en el espacio de imagen hacen uso intensivo de la GPU, y se basan en la rasterización de los objetos de una prueba de colisión en los diferentes

buffers (Z-Buffer, A-Buffer, Stencil-Buffer, entre otros). Estas técnicas no requieren estructuras complejas de almacenamiento para realizar la detección de colisiones y funcionan bien con múltiples objetos deformables en movimiento. Sin embargo, debido a la resolución de los buffers en los cuales los objetos son renderizados, todas las técnicas basadas en el espacio de imagen son aproximadas.

Las técnicas para la detección de colisiones basadas en el espacio de imagen pueden ser realizadas en objetos convexos y no convexos.

Para detectar las colisiones de objetos convexos, una aproximación consiste en ver cada pixel del buffer de renderizado como un rayo perpendicular al plano de visión que se dirige hacia la escena. Cuando un rayo intersecta un objeto, su intersección se puede describir con un intervalo que comienza en el punto por el cual entra el rayo al objeto y termina en el punto por el cual el rayo sale del objeto. Si los intervalos de los objetos se solapan, existe una colisión entre los mismos. Esta prueba necesitaba realizar la lectura del buffer desde la memoria del GPU, operación bastante lenta; sin embargo, existe una nueva aproximación en la cual la misma prueba se puede realizar en el GPU mediante la utilización de consultas de oclusión.

Las consultas de oclusión se realizan en dos pasos. En el primer paso se renderiza el primer objeto en el Z-Buffer, usando una prueba de profundidad menor o igual. La prueba de profundidad se cambia luego a mayor que, se deja de actualizar el Z-buffer y se renderiza el segundo objeto, en este caso si la consulta de oclusión afirma que no existen pixeles visibles del segundo objeto entonces el segundo objeto debe encontrarse completamente enfrente del primer objeto, por lo tanto no colisionan. Si no es así, es necesario realizar el segundo paso en el cual se invierten los roles del primer objeto y el segundo objeto.

El Modelo de los Vértices Extremos (EVM)

El modelo de los vértices extremos o *Extreme Vertices Model* (EVM) constituye un modelo de representación de sólidos basado en pseudo-poliedros ortogonales, en el cual toda la información referente a la geometría y relaciones topológicas referentes a las caras, aristas y vértices del objeto pueden ser obtenidas a partir de su EVM. En este modelo, cualquier pseudo-poliedro ortogonal (OPP) es representado por un subconjunto de sus vértices, los cuales son llamados vértices extremos (Aguilera, A., 1998: 4-36).

En el trabajo de Aguilera, A. (1998: 4-2) se exponen los siguientes conceptos básicos y propiedades del EVM:

Un poliedro ortogonal (OP) es un poliedro en el cual todas sus aristas y caras están orientadas en tres direcciones ortogonales.

Un pseudo-poliedro ortogonal (OPP) es un OP con una frontera no-variedad. Para las siguientes definiciones se considerará a *P* un OPP.

Un borde, orilla o *brink* es el segmento más largo ininterrumpido formado por aristas 2-variedad, colineales y continuas de *P*.

Los vértices extremos de *P*, *EV*(*P*), son los vértices terminales de todos los bordes de *P*. Estos constituyen un subconjunto de todos los vértices de *P*.

Un conjunto finito de puntos en un espacio tridimensional se considera ABC-ordenado o ABC-*Sorted* si se ordena primero según la coordenada A, segundo según la coordenada B y tercero según la coordenada C. Existen por lo tanto seis posibilidades para ordenar un conjunto de puntos: XYZ, XZY, YXZ, YZX, ZXY y ZYX.

El EVM de un poliedro o pseudo-poliedro ortogonal *P*, *EVM*(*P*), es un conjunto ABC-*Sorted* de *EV*(P).

Un plano de vértices (*plv*) de *P*, es el conjunto de todos los vértices que se encuentran en un plano perpendicular al eje principal del *P*, usualmente el eje A.

Un corte o *slice* es la región que se encuentra entre dos planos (o líneas) consecutivos de vértices. *P* puede ser expresado como la unión de todos sus cortes en cierta dirección ortogonal.

Una sección (*S*) es el polígono o conjunto de polígonos resultante de la intersección entre un *slice* de un OPP y un plano o línea ortogonal y perpendicular a cierta dirección ortogonal. $S_k(P)$ se refiere a la *k*-ésima sección del OPP entre el $plv_k(P)$ y el $plv_{k+1}(P)$.

Propiedades del EVM

Sea *P* un OPP con *n* planos de vértices. $S_0(P) = S_n(P) = \varnothing$.

Propiedad 1

Las coordenadas de todos los vértices no extremos se pueden obtener a partir de las coordenadas de los vértices extremos.

Propiedad 2

Las secciones pueden ser calculadas a partir de los planos de vértices y viceversa.

$$\overline{S_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{plv_i(P)} \qquad (1)$$

$$\overline{plv_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{S_i(P)} \qquad (2)$$

donde $\overline{plv_i(P)}$ y $\overline{S_i(P)}$ son las proyecciones de $plv_i(P)$ y $S_i(P)$ sobre un plano principal paralelo a *P*, y $\otimes^*$ representa a la operación XOR regularizada.

Observe que para operar con las proyecciones se necesita obviar la coordenada de los EVs que corresponde al plano de proyección.

Propiedad 3

El cálculo de cualquier plano de vértices puede ser descompuesto en dos términos a los cuales llamaremos diferencia hacia delante (*FD*) y diferencia hacia atrás (*BD*).

Para $i = 1 \ldots n$ se tiene que:

$$\overline{plv_i(P)} = \left(\overline{S_{i-1}(P)} -^* \overline{S_i(P)}\right) \cup \left(\overline{S_i(P)} -^* \overline{S_{i-1}(P)}\right) \qquad (3)$$

Y puede ser descompuesta como sigue:

$$\overline{FD_i(P)} = \left(\overline{S_{i-1}(P)} -^* \overline{S_i(P)}\right) \qquad (4)$$

$$\overline{BD_i(P)} = \left(\overline{S_i(P)} -^* \overline{S_{i-1}(P)}\right) \qquad (5)$$

Propiedad 4

$FD_i(P)$ es el conjunto de las caras del $plv_i(P)$ cuyos vectores normales apuntan a un lado del eje principal perpendicular a $plv_i(P)$, mientras que $BD_i(P)$ es el conjunto de las caras cuyos vectores normales apuntan al lado opuesto.

Propiedad 5

Sean P y Q dos d-D ($d \leq 3$) OPPs, y sean $EVM(P)$ y $EVM(Q)$ sus modelos, entonces,

$$EVM(P \otimes^* Q) = EVM(P) \otimes EVM(Q) \qquad (6)$$

Cabe destacar que las propiedades 3 y 4 son la prueba de que el modelo EVM es un modelo completo de representación de frontera de un objeto (B-Rep).

EVM como Modelo Intermedio de Visualización y Manipulación

Rodríguez, J. (2004: 70) explica que dado que un volumen digital binario es en esencia un conjunto de pseudo-poliedros ortogonales, puede ser representado por el EVM. Dado que el EVM es un modelo conciso de representación y goza de propiedades tales como clausura, orientación y conectividad, puede ser utilizado como un esquema intermedio de representación de datos volumétricos el cual permite visualizar y manipular volúmenes. EVM resulta competitivo a nivel en términos de almacenamiento y accesibilidad con respecto a otras estrategias de representación tales como *Octree* y *Semiboundary* (2004:85).

Por otro lado, dado que existen estrategias que permiten generar una representación volumétrica de objetos geométricos tridimensionales (Kaufman, A. y Shimony, E., 1985: 45), se puede decir que el EVM puede ser usado como un esquema intermedio de representación para dichos objetos.

Para utilizar el EVM como esquema intermedio de visualización y manipulación de objetos se requiere considerar la existencia de un preprocesamiento, en el cual se obtiene el EVM del objeto, y en caso de que el objeto sea geométrico, debe ser obtenida su representación volumétrica o voxelización, y un postprocesamiento, que se realiza para poder realizar la visualización del objeto, en el cual se obtiene su representación de frontera (B-Rep) y se realiza el proceso de teselado en caso de que alguna de las caras obtenidas a partir del EVM sea cóncava.

Sistema de Variables e Hipótesis

Variables

Hernández, R. (2010: 93) define una variable como “una propiedad que puede fluctuar y cuya variación es susceptible de medirse u observarse”. En este trabajo de investigación se tomarán en cuenta las siguientes variables:

Variable Independiente

Cantidad de vértices extremos de un pseudo-poliedro ortogonal. Esta cantidad es un valor entero que se extraerá de cada pseudo-poliedro ortogonal usado como base para probar el algoritmo de detección de colisiones.

Variable Dependiente

Tiempo de ejecución del algoritmo de detección de colisiones entre objetos volumétricos para el Modelo de los Vértices Extremos. Este valor se obtendrá cada vez que se ejecute el algoritmo de detección de colisiones y se medirá en microsegundos.

Hipótesis

Según Tamayo y Tamayo, M. (2003: 31) “Una hipótesis es una proposición que puede ser puesta a prueba para determinar su validez. Siempre lleva a una prueba empírica; es una pregunta formulada de tal modo que se puede prever una respuesta de alguna especie”. De acuerdo a los estudios realizados por Rodríguez, J. (2004: 64), “El EVM es un modelo de sólidos completo (no ambiguo): Es un modelo implícito de representación de la frontera de un objeto”, por lo tanto puede ser utilizado para representar el volumen envolvente más ajustado al volumen de interés, y en consecuencia, puede ser utilizado en el proceso de detección de colisiones.

Por la razón descrita anteriormente, surge la siguiente hipótesis de investigación:

Hi: El tiempo de ejecución del algoritmo de detección de colisiones entre dos objetos volumétricos modelados mediante el EVM se incrementa en función a la cantidad de vértices extremos de los objetos sobre los cuales se aplica el algoritmo.

Definición de Términos Básicos

Para efectos de este trabajo se consideraran los términos definidos a continuación:

AABB (*Axis Aligned Bounding Box*): Es una caja rectangular de seis lados (en 3D, de cuatro lados en 2D) que se caracteriza por tener sus caras orientadas de tal manera que sus vectores normales son siempre paralelos a los ejes del sistema de coordenadas dado (Ericson, C., 2005: 77).

Arista: Es cada segmento de línea que delimita un polígono (Ericsson, C., 2005: 56).

BV (*Bounding Volume*): Es un volumen simple que encapsula uno o más objetos de naturaleza compleja (Ericson, C., 2005: 75).

BVH (*Bounding Volume Hierarchy*): Es un árbol jerarquico de BVs (Ericson, C., 2005: 235).

Cara: Es un polígono que forma parte de un objeto gráfico tridimensional. (Stevens, R., 1993: 87).

Normal: Perpendicular a una superficie (Stevens, R., 1993: 146).

OBB (*Oriented Bounding Box*): Es una caja rectangular parecida a la de un AABB pero con una orientación arbitraria (Ericson, C., 2005: 101).

Pixel: Es un único elemento de una pantalla discreta (Stevens, R., 1993: 162).

Poliedro: Es un objeto tridimensional cerrado, compuesto de polígonos, en el cual cada arista es compartida por dos polígonos y cada vértice es compartido por dos o más polígonos (Stevens, R., 1993: 165). Es la contraparte tridimensional de un polígono. Es una región delimitada y conexa de espacio en forma de un sólido de múltiples caras (Ericson, C., 2005: 62).

Poliedro Ortogonal: Es un poliedro en el cual todas sus caras están orientadas en tres direcciones ortogonales (Rodríguez, J., 2004: 63).

Polígono: Es una figura cerrada de *n* lados, definida por un conjunto ordenado de tres o más puntos en el plano de tal manera que cada punto se conecta al siguiente (y el ultimo se conecta al primero) mediante un segmento de línea. (Ericson, C., 2005: 56).

Polígono Convexo: Es un polígono cuyos segmentos de línea entre cada par de puntos dentro del polígono, se encuentran también completamente dentro del polígono (Ericson, C., 2005: 57).

Polítopo: Es un poliedro convexo (Ericson, C., 2005: 62).

Pseudo-Poliedro Ortogonal: Es un poliedro ortogonal con una frontera *non-manifold* (Rodríguez, J., 2004: 63).

Raster: Es la disposición de píxeles en forma de un arreglo bidimensional o grid en un medio de salida digital (Stevens, R., 1993: 175).

Rasterizar: Es la acción de convertir una imagen en píxeles para su visualización en un medio de salida digital (Stevens, R., 1993: 176).

Renderizar: Es la acción de convertir un modelo gráfico en un arreglo de píxeles para su visualización en un medio de salida digital (Stevens, R., 1993: 179).

Simplejo: Un *d*-simplejo es la envolvente convexa de $d + 1$ puntos linealmente independientes en un espacio *d*-dimensional. Un simplejo es un *d*-simplejo para un *d* dado. Por ejemplo, el 0-simplejo es un punto, el 1-simplejo es un segmento de línea, el 2-simplejo es un triángulo y el 3-simplejo es un tetraedro (Ericson, C., 2005: 62).

Tiempo de Ejecución de un Algoritmo: Es la cantidad de unidades de tiempo que consume la ejecución de un algoritmo. Puede ser medido a priori o a posteriori (Florez, R., 2005:16).

Vector: Es una línea recta definida por las coordenadas de sus extremos (Stevens, R., 1993: 213).

Vector Normal: Es un vector unitario perpendicular a una superficie en un punto determinado (Stevens, R., 1993: 146).

Vértice: Es un punto que marca la intersección de dos o más aristas de un polígono u otro objeto gráfico (Stevens, R., 1993: 214).

Vértices Extremos: Son los vértices terminales de todos los bordes de un Pseudo-Poliedro Ortogonal, estos constituyen un subconjunto de todos los vértices del mismo (Aguilera, A., 1998: 4-4).

CAPÍTULO III

MARCO METODOLÓGICO

En este capítulo se incluyen el tipo, modo y diseño de la investigación, seguido de las técnicas e instrumentos para el análisis de los datos obtenidos, además de los procedimientos que serán utilizados para llevar a cabo este estudio.

Tipo y Modalidad de la Investigación

El presente estudio cumple con las características de una investigación científica básica, la cual, según Tamayo y Tamayo, M. (2003: 43) “tiene corno objeto el estudio de un problema destinado exclusivamente al progreso o a la simple búsqueda del conocimiento”. Con este estudio se pretende resolver el problema de la detección de colisiones entre objetos volumétricos representados con el modelo de los vértices extremos mediante la creación de un algoritmo, generando así nuevos conocimientos en el área de la computación gráfica, específicamente en la visualización de volúmenes. Esta operación no ha sido desarrollada para el EVM, y busca complementarlo, ya que el mismo ha demostrado ser eficiente.

Se enfoca bajo la modalidad exploratoria y explicativa. Según Hernández, R. (2010: 79) la modalidad exploratoria se aplica “cuando el objetivo es examinar un tema o problema de investigación poco estudiado… o no se ha sido abordado antes”. Mientras que, Hernández, R. (2010: 84) afirma que los estudios explicativos centran su interés en “explicar por qué ocurre un fenómeno y en qué condiciones se manifiesta, o por qué se relacionan dos o más variables”. Este trabajo de

investigación es exploratorio ya que busca generar conocimientos en el área de visualización y manipulación de objetos volumétricos mediante la creación de un algoritmo de detección de colisiones para el modelo de los vértices extremos. Igualmente, es explicativo ya que se buscará explicar el efecto que tiene el aumento o disminución de vértices extremos de los modelos en el tiempo de ejecución del algoritmo desarrollado.

Diseño de la Investigación

Gómez, M. (2006: 61) establece que la investigación cuantitativa se enfoca particularmente en "cuantificar y aportar evidencias a una teoría que se tiene para explicar algo. Se asocia con los experimentos donde se manipulan variables". Este estudio aplicará el método cuantitativo, ya que tanto la cantidad de vértices extremos de los objetos de interés como el tiempo de ejecución del algoritmo de detección de colisiones entre objetos volumétricos constituyen variables cuantificables que serán presentadas de manera numérica en los resultados del estudio.

Asimismo, debido a la existencia de las variables antes descritas, las cuales deben ser tomadas en cuenta para estudiar el desempeño del algoritmo de detección de colisiones entre objetos volumétricos, en este proyecto se usará un diseño experimental con variables, en el cual se manipulara la cantidad de vértices extremos para observar como esto afecta el tiempo de ejecución del algoritmo.

En esta investigación se pretende mejorar el modelo de los vértices extremos mediante la construcción de un algoritmo para la detección de colisiones basado en el modelo EVM, para luego medir su desempeño en función de las variables definidas.

Según Hernández, R. (2010: 137), “Los experimentos puros son aquellos que reúnen los dos requisitos para lograr el control y la validez interna: 1. Grupos de comparación (manipulación de la variable independiente); 2. Equivalencia de los grupos”. De acuerdo a esto, la variable independiente, que es la cantidad de vértices extremos, será manipulada en un ambiente controlado para verificar su efecto sobre la variable dependiente, que es el tiempo de ejecución del algoritmo. Luego estas variables serán analizadas y permitirán establecer una serie de conclusiones con respecto al desempeño del algoritmo. Los resultados de esta de investigación serán resumidos en tablas y gráficos para una mejor interpretación.

Población y Muestra

La población, definida según Tamayo y Tamayo, M. (2003: 176) es la “totalidad de un fenómeno de estudio, incluye todas las unidades de análisis o entidades de población que integran dicho fenómeno y que debe cuantificarse para un determinado estudio”, de acuerdo a esto, la población en este estudio está constituida por todos los algoritmos de detección de colisiones entre objetos volumétricos.

De esta población sólo se tomará una muestra. Según Hernández, R. (2010: 173), una muestra “es un subgrupo de la población de interés sobre el cual se recolectarán datos, y que tiene que definirse o delimitarse de antemano con precisión, éste deberá ser representativo de dicha población”. En esta investigación la muestra comprende una única unidad de análisis o caso de estudio, que es el algoritmo de detección de colisiones para el modelo de los vértices extremos, sobre el cual se harán los experimentos.

Técnicas e Instrumentos de Recolección de Datos

Sabino, C. (1992: 155) afirma que “se hace necesario definir las técnicas de recolección necesarias para construir los instrumentos que nos permitan obtener los datos de la realidad”, en este sentido, para realizar la recolección de datos se utilizará el método de observación científica. La palabra observación, según Tamayo y Tamayo, M. (2003: 183) “hará referencia explícitamente a la percepción visual y se emplea para indicar todas las formas de percepción utilizadas para el registro de respuestas tal como se presentan a nuestros sentidos”. En este estudio los datos sobre la cantidad de vértices extremos y el tiempo de ejecución del algoritmo de detección de colisiones serán captados con ayuda del computador.

En cuanto a las técnicas de procesamiento de los datos, se tomarán en cuenta los métodos de tabulación y graficación para organizar los datos obtenidos en los experimentos realizados. Según Sabino, C., la tabulación consiste en “hacer tablas, listados de datos que los muestren agrupados y contabilizados” (1992: 140) y la graficación consiste en “expresar visualmente los valores numéricos que aparecen en los cuadros” (1992: 188), lo cual garantiza una comprensión directa y global de la información numérica. Con el método de tabulación será posible organizar los resultados referentes a los tiempos de ejecución del algoritmo desarrollado con respecto a los diferentes casos de prueba, así como también será posible organizar los datos comparativos entre el algoritmo de detección de colisiones para el modelo de los vértices extremos y otro algoritmo de detección de colisiones entre objetos volumétricos. Con el método de graficación se mostrará la relación entre la cantidad de vértices extremos y el tiempo de ejecución del algoritmo de una forma más sencilla para su comprensión.

Análisis de los Datos

Se realizará un análisis cuantitativo de los datos que según Sabino, C. (1992: 172), “se efectúa, naturalmente, con toda la información numérica resultante de la investigación”, la cual en este trabajo de investigación, luego del procesamiento, se encontrará organizada en tablas y gráficas.

Para cada una de las tablas y gráficas se debe estudiar el comportamiento del tiempo de ejecución del algoritmo con respecto a la cantidad de vértices extremos presentes en los objetos volumétricos, con la finalidad de establecer una relación matemática entre ambas.

Procedimientos Previstos

Para crear un algoritmo de detección de colisiones entre dos objetos volumétricos modelados mediante EVM, que es el objetivo de este proyecto de investigación, se realizarán los siguientes procedimientos:

1. Analizar el modelo de representación de volúmenes EVM: En este punto se analiza el EVM con la finalidad de conocer a fondo los conceptos principales, propiedades y herramientas del modelo, para tomarlas como base para el desarrollo del algoritmo de detección de colisiones.

2. Diseñar la estructura de datos adecuada para el almacenamiento de objetos volumétricos modelados mediante el EVM: Una ver analizado el EVM se pasa al

diseño de una estructura de datos adecuada para el almacenamiento de los objetos volumétricos modelados mediante el EVM. El EVM de un objeto se obtiene a partir de un objeto volumétrico voxelizado, por lo tanto también se diseñará la estructura de datos adecuada para almacenar dicho objeto.

3. Crear un algoritmo que permita realizar la detección de colisiones entre dos objetos volumétricos modelados mediante el EVM: Para crear el algoritmo se utilizará el lenguaje de programación C++, y se programará de acuerdo al paradigma de programación orientada a objetos. En un principio se crearan las clases auxiliares necesarias para la construcción de la clase EVM, luego se creará para esta clase un método de detección de colisiones.

4. Realizar pruebas con el algoritmo creado: Una vez creado el algoritmo de detección de colisiones entre dos objetos modelados mediante el EVM, es necesario crear un programa principal, con la finalidad de realizar pruebas sobre el algoritmo. Se harán dos pruebas: (1) Prueba de desempeño, en la cual se obtendrán datos que permitan establecer una relación entre la cantidad de vértices extremos y el tiempo de ejecución del algoritmo, y (2) Prueba comparativa, donde se obtendrán datos que permitan establecer una comparación entre el algoritmo de detección de colisiones creado con alguna técnica similar.

5. Evaluar el algoritmo creado: Finalmente, una vez recopilados los datos de las pruebas, se evaluará el algoritmo en cuanto a su desempeño y comparación con otra técnica similar.

CAPÍTULO IV

RESULTADOS

En este capítulo se presentan los detalles sobre el diseño, implementación, pruebas y evaluación del algoritmo de detección de colisiones entre objetos volumétricos representados mediante el modelo de los vértices extremos, y los resultados correspondientes a las pruebas de desempeño del mismo. Para la presentación de algoritmos se utilizará el lenguaje pseudoformal NASPOO (Ottogalli, K., Martínez, A. y León, L., 2010).

Diseño del Algoritmo de Detección de Colisiones entre EVMs

Para diseñar el algoritmo de detección de colisiones entre objetos volumétricos representados mediante el modelo de los vértices extremos se tomó en cuenta la redefinición hecha por Rodríguez, J. (2004: 67) de la especificación operacional dada inicialmente en Aguilera, A. (1998: 4-28), en la cual el EVM de un pseudo-poliedro ortogonal n-dimensional ($n \leq 3$), es representado mediante un tipo de dato abstracto que almacena el EVM ABC-Ordenado de un objeto volumétrico, el cual cuenta con una serie de operaciones generales para su manipulación.

Operaciones Generales del EVM

El tipo EVM cuenta con una serie de operaciones generales, las cuales se asumirá que son aplicadas sobre una instancia de EVM, la cual se denotará como P. Para la

definición de este tipo se toma en cuenta que existen los tipos Vértice, que representa un punto tridimensional, Dimensión, un tipo enumerado que define si el EVM es 3D, 2D o 1D, y Operación, un tipo enumerado que define las operaciones booleanas válidas para un EVM ($\otimes$, $\cup$, $\cap$ y $-$).

Sean *P*, *Q*, *plv* y S_j instancias de EVM, *v*, v_i, y v_f instancias de Vértice, *dim* una instancia de Dimensión, que representa la dimensión de *P*, *coord* un valor Real que representa una coordenada de un Vértice, y *op* una instancia de Operación, se tienen las siguientes operaciones generales para el EVM:

func construir(Dimensión: dim): EVM

Retorna un EVM vacío, *P*, de dimensión *dim*.

func isEmpty(): **Lógico**

Retorna verdadero en caso de que *P* sea vacío y falso en caso contrario.

proc putExtremeVertex(Vertice: v)

Añade un vértice extremo, *v*, a *P*.

func readPlv(): EVM

Obtiene el siguiente OPP (*dim*-1)-dimensional de *P*.

proc putPlv(EVM: plv)

Añade un OPP (*dim*-1)-dimensional, *plv*, a *P*.

proc readBrink(**ref** Vertice: vi, vf)

Obtiene el par actual de vértices extremos v_i y v_f de P.

proc putBrink(Vertice: vi, vf)

Añade dos vértices extremos consecutivos v_i y v_f a P.

func endEVM(): **Lógico**

Retorna verdadero si se ha alcanzado el final de P, o falso en caso contrario.

func getCoord(): **Real**

Obtiene la coordenada común A si P es 2D o B si P es 1D.

proc setCoord(**Real**: coord)

Cambia el valor de la coordenada A si P es 2D o B si P es 1D, de todos los vértices extremos de P a coord. Si coord = 0, realiza la proyección de P.

func operador ⊗(EVM: Q): EVM

Aplica la operación or exclusivo (xor) a los vértices de P y Q, y retorna el conjunto de vértices extremos resultante.

func getSection(EVM: plv; **Entero**: dim): EVM

Retorna la siguiente sección de un OPP cuya sección previa es P y cuyo corte actual es *plv*.

func getPlv(EVM: Sj; **Entero**: dim): EVM

Retorna el siguiente corte de un OPP entre dos secciones consecutivas, S_i y S_j.

func operation(EVM: Q; **Entero**: dim; Operation: op): EVM

Realiza la operación booleana *op* entre dos objetos *dim*-dimensionales, *P* y *Q*.

Algoritmo de Detección de Colisiones

La detección de colisiones consiste en detectar si dos (o más) objetos se intersectan (Ericson, 2004: 1). Aguilera, A. (1998: 4-16) define un conjunto de operaciones booleanas regularizadas para el modelo de los vértices extremos, entre las cuales se encuentra la intersección entre dos objetos. En su trabajo, Aguilera A. presenta un algoritmo general para realizar las operaciones booleanas regularizadas (1998: 5-2), sobre el cual Rodríguez, J. propuso dos aproximaciones que permiten lograr una aceleración basada en la utilización de la región de solapamiento o región crítica entre los objetos (2004: 91). Pérez-Águila, R. (2006: 200) por su parte, presenta entre las aplicaciones del modelo de los vértices extremos, como se puede realizar la detección de colisiones usando como base la operación booleana regularizada de intersección entre objetos.

Esta aproximación es acertada, sin embargo, requiere el cálculo completo de la intersección entre los objetos para responder si existe una colisión o no, esto implica una sobrecarga innecesaria en tiempo de procesamiento y memoria que puede ser evitada tomando en cuenta que para saber si dos objetos colisionan, basta con calcular si se intersectan en al menos un borde.

Tabla 1. Algoritmo de la operación collision1D.

```
01  func collision1D(ref EVM: Q): EVM
02  var
03      EVM: R
04      Entero: ip, iq
05  inicio
06      R.construir(instancia.dim)
07      mientras(R.isEmpty() ∧ ¬instancia.endEVM() ∧ ¬Q.endEVM()) hacer
08          si(instancia.getExtremeVertex(ip).c() > Q.getExtremeVertex(iq).c())
                entonces
09              si(ip mod 2 = 1) entonces
10                  si(instancia.getExtremeVertex(ip-1).c() <
                        Q.getExtremeVertex(iq).c()) entonces
11                      R.putExtremeVertex(Q.getExtremeVertex(iq))
12                  fsi
13              fsi
14              iq ← iq + 1
15          sino
16              si(instancia.getExtremeVertex(ip).c() < Q.getExtremeVertex(iq).c())
                    entonces
17                  si(iq mod 2 = 1) entonces
18                      si(Q.getExtremeVertex(iq-1).c() <
                            instancia.getExtremeVertex(ip).c()) entonces
19                          R.putExtremeVertex(instancia.getExtremeVertex(ip))
20                      fsi
21                  fsi
22                  ip ← ip + 1
23              sino
24                  si((ip mod 2 = 0 ∧ iq mod 2 = 0) ∨ (ip mod 2 = 1 ∧ iq mod 2 = 1))
                    entonces
25                      R.putExtremeVertex(instancia.getExtremeVertex(ip))
26                  fsi
27                  ip ← ip + 1
28                  iq ← iq + 1
29              fsi
30          fsi
31      fmientras
32      retornar(R)
33  ffunc
```

Fuente: Elaboración propia.

Tabla 2. Algoritmo de la operación collision.

```
01  func collision(EVM: Q): EVM
02  var
03      Lógico: fromP, fromQ        { banderas para los plv's fuente }
04      Real: coord                 { la coordenada común de plv }
05      EVM: sP, sQ                 { secciones actuales de P y Q }
06      EVM: plv                    { plano (línea) de vértices }
07      EVM: R, sRprev, sRcurr      { R y dos de sus secciones }
08  inicio
09      si (instancia.dim = 1) entonces
10          retornar(instancia.collision1D(Q))
11      sino
12          sP.construir(instancia.dim-1)
13          sQ.construir(instancia.dim-1)
14          sRcurr.construir(instancia.dim-1)
15          instancia.nextObject(Q, coord, fromP, fromQ)
16          instancia.improve(Q, coord, fromP, fromQ, plv, sP, sQ, R)
17          mientras (R.isEmpty() ∧
                ¬instancia.endEVM() ∧ ¬ Q.endEVM()) hacer
18              si (fromP) entonces
19                  plv ← instancia.readPlv()
20                  sP ← sP.getSection(plv)
21              fsi
22              si (fromQ) entonces
23                  plv ← Q.readPlv()
24                  sQ ← sQ.getSection(plv)
25              fsi
26              sRprev ← sRcurr
27              sRcurr ← sP.collision(sQ)     { Llamada recursiva }
28              plv ← sRprev.getPlv(sRcurr)
29              plv.setCoord(coord)
30              R.putPlv(plv)
31              instancia.nextObject(Q, coord, fromP, fromQ)
32          fmientras
33          retornar(R)
34      fsi
35  ffunc
```

Fuente: Elaboración propia.

Tabla 3. Algoritmo de la operación collide.

```
01  func collide(EVM: Q): Lógico
02  inicio
03          retornar(¬instancia.collision(Q).isEmpty())
04  ffunc
```

Fuente: Elaboración propia.

Por esta razón, para diseñar el algoritmo de detección de colisiones para el modelo de los vértices extremos, se tomó como base el algoritmo general para realizar las operaciones booleanas regularizadas presentado por Aguilera, A., mejorado mediante la aproximación basada en planos de vértices presentada por Rodríguez, J., el cual se adaptó para inducir una salida adelantada al detectar la primera intersección entre los bordes de los objetos.

El algoritmo presentado en la Tabla 2 es una versión modificada del algoritmo presentado por Aguilera, A. (1998: 5-2) para realizar operaciones booleanas entre dos instancias de EVM.

La líneas 9-10 muestran que en caso de que los objetos sean unidimensionales, es decir, cuando los objetos estén formados únicamente por bordes, se debe aplicar la operación collision1D presentada en la Tabla 1, la cual retorna un EVM formado por un solo punto en caso de que haya solapamiento entre los bordes de los objetos, o un EVM vacío en caso contrario. Esta operación es una versión modificada de la intersección para objetos unidimensionales, ya que en vez de retornar todos los bordes resultantes de la intersección, retorna únicamente el primer borde en el cual ésta ocurre.

En la línea 16 se encuentra la operación improve, la cual aplica la aproximación basada en planos de vértices presentada por Rodríguez, J. (2004: 91). La operación descarta aquellos *plvs* que se encuentran del lado izquierdo de la región crítica; cuando el inicio de ésta se encuentra, la operación termina y se continúa con el algoritmo.

Como se puede ver, en la línea 17 se modificó la condición del ciclo mientras para que verifique en un principio si *R* es vacío, si no lo es, significa que al menos se detectó un solapamiento entre los bordes de *P* y *Q*, y por lo tanto se detectó una colisión.

En este algoritmo, como se puede notar en las líneas 27-30, al ocurrir una colisión a nivel unidimensional se retorna un EVM que únicamente contiene un borde, que es el primer borde en el cual *P* y *Q* se intersectan, lo cual implica que a nivel bidimensional, este borde constituye la sección actual del EVM resultante calculado, *R*, y dado que la sección anterior a esta colisión debe ser vacía, este mismo borde se convierte en un *plv* que es agregado a *R*. De esta manera, *R* deja de ser vacío y se retorna, produciendo que a nivel tridimensional se repita lo ocurrido a nivel bidimensional, y la operación termine.

Entre las líneas 32 y 33 originalmente se procesaban aquellos *plvs* que se encontraban del lado derecho de la región crítica, sin embargo, como no son relevantes para la operación simplemente se descartan.

En la Tabla 3 se presenta el algoritmo encargado de la detección de colisiones entre *P* y *Q*. En la línea 3 realiza una llamada al Algoritmo 1, que retorna un EVM

que puede ser vacío en caso de que no ocurra una colisión, o puede tener un borde en caso de que si ocurra. Si el algoritmo *collision* retorna un EVM vacío, la operación isEmpty retornará verdadero, en cuyo caso el algoritmo *collide* retornará la negación de esta valor, es decir, falso ya que no ocurre una colisión. Si el algoritmo *collision* retorna un EVM no vacío, la operación isEmpty retornará falso, en cuyo caso el algoritmo *collide* retornará verdadero, ya que se ha detectado una colisión.

Implementación del Algoritmo de Detección de Colisiones entre EVMs

Para la implementación del algoritmo de detección de colisiones entre objetos volumétricos representados mediante el modelo de los vértices extremos, se utilizó el lenguaje de programación C++ bajo el paradigma de programación orientada a objetos, y además fueron desarrolladas una serie de librerías de soporte para el manejo de las unidades básicas de construcción de objetos.

Librerías de Soporte

Las librerías de soporte desarrolladas fueron las siguientes:

- point3d: Una librería para el manejo de puntos tridimensionales.
- line3d: Una librería para el manejo de líneas tridimensionales.
- vector3d: Una librería para el manejo de vectores tridimensionales.
- plane: Una librería para el manejo de planos.

– object: Una librería para el manejo de objetos geométricos basados en triángulos. Esta librería se usa en la conversión de EVM a B-Rep para realizar la visualización de un EVM.

– AABB: Una librería para el manejo de los AABBs obtenidos a partir de objetos.

– voxel: Una librería para el manejo de voxeles.

– vObject: Una librería para el manejo de objetos volumétricos.

Implementación del Tipo EVM

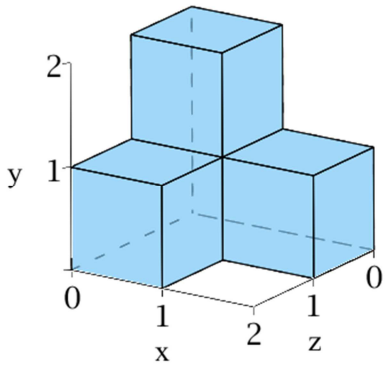
A nivel computacional, se pueden definir diversas estructuras de datos que permitan representar un EVM. En este trabajo se utilizaron dos estructuras diferentes sobre las cuales la operación de detección de colisiones será evaluada: (1) la estructura ABC-*Sorted*, propuesta por Aguilera A. (1998: 4-28), la cual fue implementada de la forma más cercana posible a la definición de EVM, y (2) la estructura *Trie-Tree*, propuesta por Pérez-Águila, R. (2006: 127).

ABC-*Sorted*

La estructura de datos llamada ABC-*Sorted* para representar el EVM de un OPP denotado como *P*, almacena la secuencia ordenada del conjunto de vértices extremos de *P*, *EV*(*P*). El orden de *EV*(*P*) se realiza primero según la coordenada A, luego según la coordenada B y finalmente según la coordenada C. Un ejemplo de este tipo, correspondiente al OPP de la Figura 1, es el siguiente:

ABC-*Sorted*$_3$(*P*) = {(0, 0, 0); (0, 0, 2); (0, 1, 1); (0, 1, 2); (0, 2, 0); (0, 2, 1); (1, 0, 1); (1, 0, 2); (1, 1, 0); (1, 1, 2); (1, 2, 0); (1, 2, 1); (2, 0, 0); (2, 0, 1); (2, 1, 0); (2, 1, 1)}

Figura 1. Un pseudo-poliedro ortogonal.



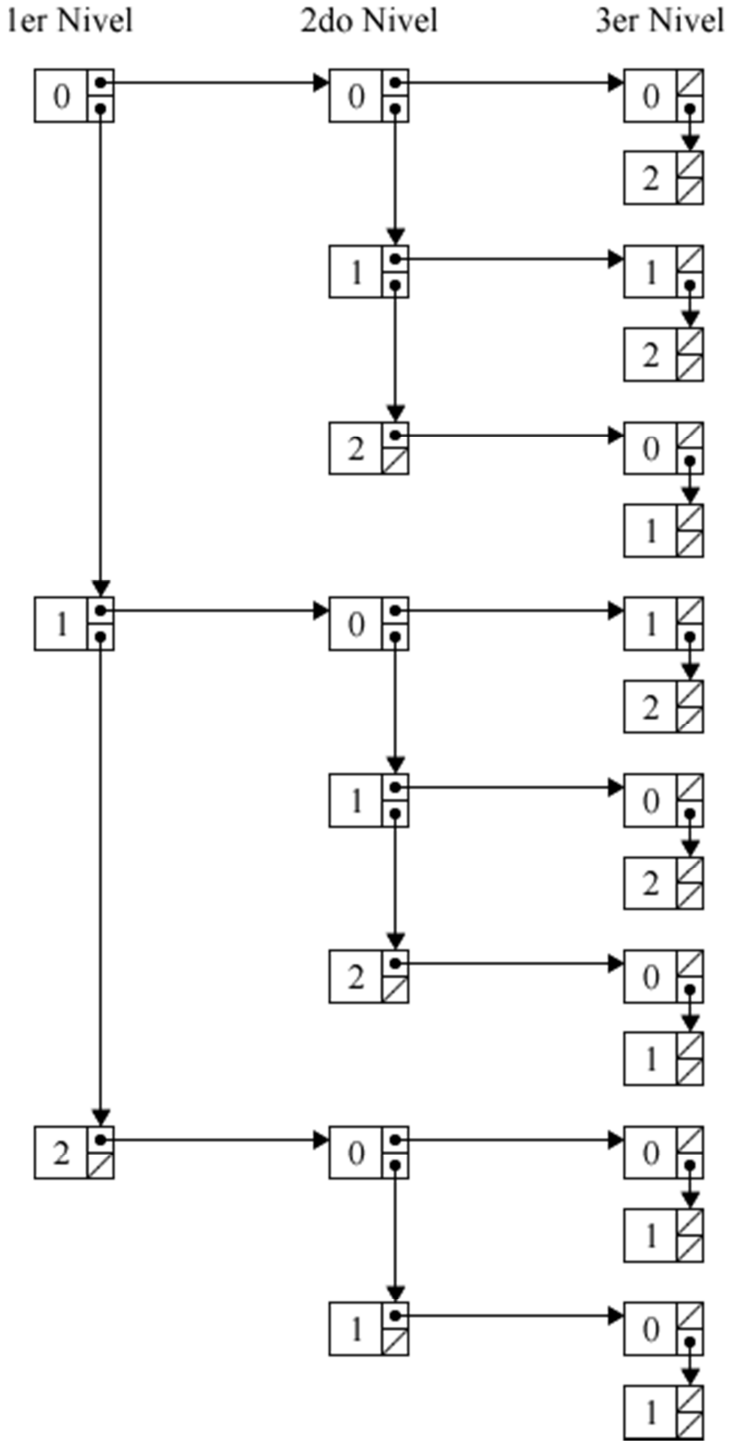
Fuente: Elaboración propia.

Trie-Tree

Pérez-Águila, R. (2006: 127), basado en los esquemas de compresión de datos propuestos por Aguilera, A. (1998: 7-3), propone la utilización de un *Trie-Tree* para almacenar un EVM.

En esta representación un vértice extremo puede ser visto como una clave de n dígitos, x_1, x_2, ..., x_n, donde cada dígito se encuentra en un nivel del árbol y representa una coordenada. Cada digito x_i está relacionado con un sub-árbol de $n - i$ niveles que representa todas aquellos vértices extremos que comparten x_i como coordenada en el nivel i.

Figura 2. Trie-Tree de un OPP.



Fuente: Elaboración propia.

La Figura 2 muestra el *Trie-Tree*$_3$(*P*) correspondiente al OPP de la Figura 1. Como se puede ver, en esta estructura cada nodo en el nivel 1, posee un subárbol que representa un plano de vértices perpendicular a A, y cada nodo en el nivel 2 posee un sub-árbol que representa un borde perpendicular a B.

La desventaja principal de esta representación es que, por su naturaleza de árbol, contempla únicamente un ordenamiento de *EVM*(*P*), por lo tanto no permite realizar la conversión de instancias de EVM a BRep (Rodríguez, J., 2004: 76), la cual hace uso intensivo de la reorganización, y tampoco se pueden obtener todos sus ordenamientos del modelo OUDB (*Ordered Union of Disjoint Boxes*).

Pruebas y Evaluación

En esta sección se presentan tres pruebas experimentales que permiten evaluar el desempeño del algoritmo de detección de colisiones entre EVMs: La primera presenta el comportamiento de la intersección como base para la detección de colisiones, la segunda presenta el comportamiento del algoritmo de detección de colisiones implementado en este trabajo, y la tercera permite evaluar si el algoritmo de detección de colisiones hace posible la utilización del EVM como un VB ajustado al objeto volumétrico.

A continuación, se presentan las características del computador utilizado para realizar las pruebas, cómo fueron generados los objetos volumétricos utilizados y las descripción detallada de cómo fue realizada cada prueba. Al final de cada prueba, se realiza un análisis de los resultados y se presenta una conclusión sobre el comportamiento de la operación.

Cada una de las pruebas se realizó para ambas implementaciones del tipo EVM: La implementación basada en la estructura de datos ABC-*Sorted* y la implementación basada en la estructura *Trie-Tree*.

Características del Computador

Las pruebas fueron realizadas en un computador con las siguientes características:

Hardware:

- Procesador Intel i5-2500k 3.3 Ghz 3 MB de caché.
- Tarjeta madre MSI P67A-GD53.
- Memoria RAM DDR3 de 4 GB (2 x 2 GB), 1600 MHz.
- Disco Duro de 1 TB@7200rpm.
- Tarjeta de video MSI GeForce GTS 450.

Software:

- Sistema operativo Ubuntu 12.04 (Precise Pangolín) Kernel 3.5.0-27-generic.
- Compilador g++ versión 4.6.3.

Generador de Casos de Prueba

Para la obtención de los objetos volumétricos necesarios para realizar las pruebas, se desarrolló un generador de casos de prueba. Este generador requiere como datos de entrada el tamaño del objeto volumétrico a producir y la probabilidad de que un voxel sea negro, y genera un objeto volumétrico de tipo vObject con dichas características. Con este generador se obtuvieron 2995 objetos volumétricos que una vez convertidos a EVM, poseen de 12 a 6000 vértices extremos.

Análisis Experimental de la Complejidad en Tiempo de la Operación de Intersección entre EVMs

Esta prueba se basa en las pruebas de desempeño de las operaciones booleanas definidas para el EVM, realizadas por Pérez-Águila, R. (2006: 136). En este trabajo las pruebas fueron enfocadas sobre la operación de intersección entre EVMs tridimensionales. Pese a que en el trabajo de Pérez-Águila, R. se realiza un análisis experimental de la complejidad en tiempo de la intersección y se presentan resultados sobre el desempeño de la misma, estos no son comparables con los resultados obtenidos en este trabajo ya que tanto el lenguaje de programación utilizado (Java), como las características de la arquitectura subyacente eran diferentes, por esta razón, fue necesario replicarlos sobre el lenguaje y la arquitectura que se utilizaron.

Se considera en esta prueba la intersección entre los EVMs P y Q, cuya suma de vértices extremos, $x = Card(EVM_3(P)) + Card(EVM_3(Q))$, varía entre 24 y 6000. Para cada x y estructura de datos, se obtuvieron dos tiempos promedio en microsegundos (μs) de la operación de intersección: (1) el tiempo promedio cuando no existe

intersección entre *P* y *Q*, y (2) el tiempo promedio cuando si existe intersección entre *P* y *Q*. Cada tiempo promedio es resultado de 1000 observaciones aleatorias para un determinado caso, en las cuales la ubicación de los objetos varía. En la Tabla 4 se muestran estos resultados para *x* con una frecuencia de 500.

Tabla 4. Comparación de los tiempos promedio de la operación de intersección 3D entre las estructuras ABC-*Sorted* y *Trie-Tree* en μs, cuando existe y cuando no existe solapamiento.

	Tiempo Promedio de Intersección en μs			
	ABC-*Sorted*		***Trie-Tree***	
x	**No Existe**	**Si Existe**	**No Existe**	**Si Existe**
500	73,052	259,685	360,990	1217,993
1000	118,511	442,062	609,738	1993,424
1500	166,487	1018,794	812,652	4395,904
2000	219,147	1395,977	1022,025	5948,518
2500	259,377	1607,284	1202,213	6610,269
3000	306,860	1730,999	1372,651	7052,986
3500	346,404	1939,742	1558,798	7648,771
4000	377,089	2121,719	1674,251	8133,469
4500	415,453	2254,310	1828,041	8573,740
5000	465,353	2371,832	2006,193	8931,911
5500	494,553	2399,574	2157,062	8932,042
6000	560,257	2630,598	2400,781	9640,061

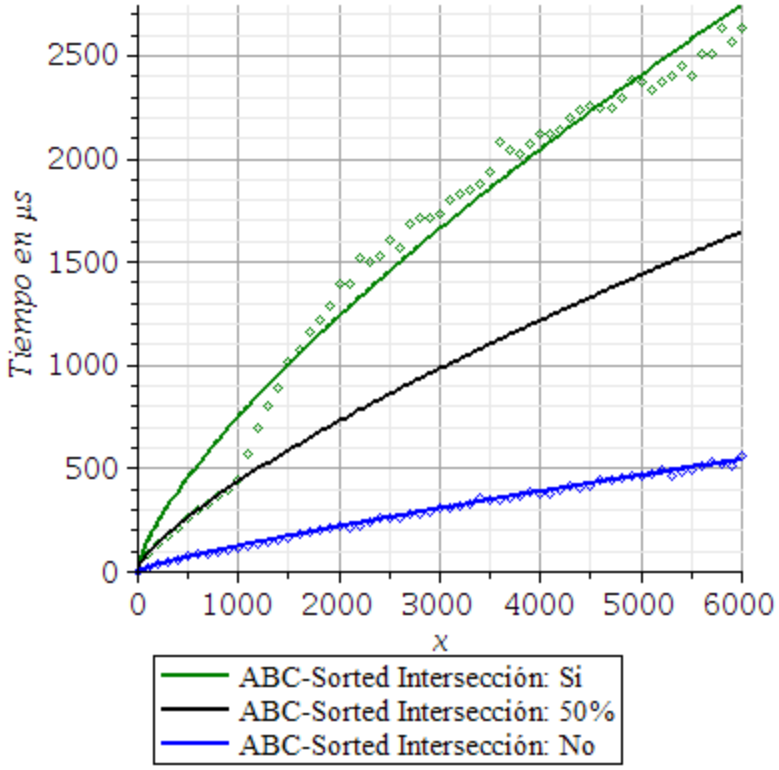
Fuente: Elaboración propia.

Como se puede observar en la Tabla 4, la operación de intersección 3D del tipo EVM implementado con la estructura ABC-*Sorted* se comporta mejor que la del tipo EVM implementado con la estructura *Trie-Tree* para todos los casos tanto si existe solapamiento entre *P* y *Q* como en caso contrario.

Este comportamiento se debe a que en la estructura *Trie-Tree* la inserción de vértices extremos es más costosa que en la estructura ABC-*Sorted*, ya que para

realizar esta inserción en un *Trie-Tree* se debe realizar la búsqueda de la posición correcta en el árbol para que los vértices se mantengan en orden, y construir los nodos necesarios en caso de que no existan, mientras que en la estructura ABC-*Sorted* esta inserción se realiza de manera directa al final.

Figura 3. Tiempos promedio en µs de la operación de intersección 3D sobre la estructura ABC-*Sorted*, cuando existe y cuando no existe intersección.



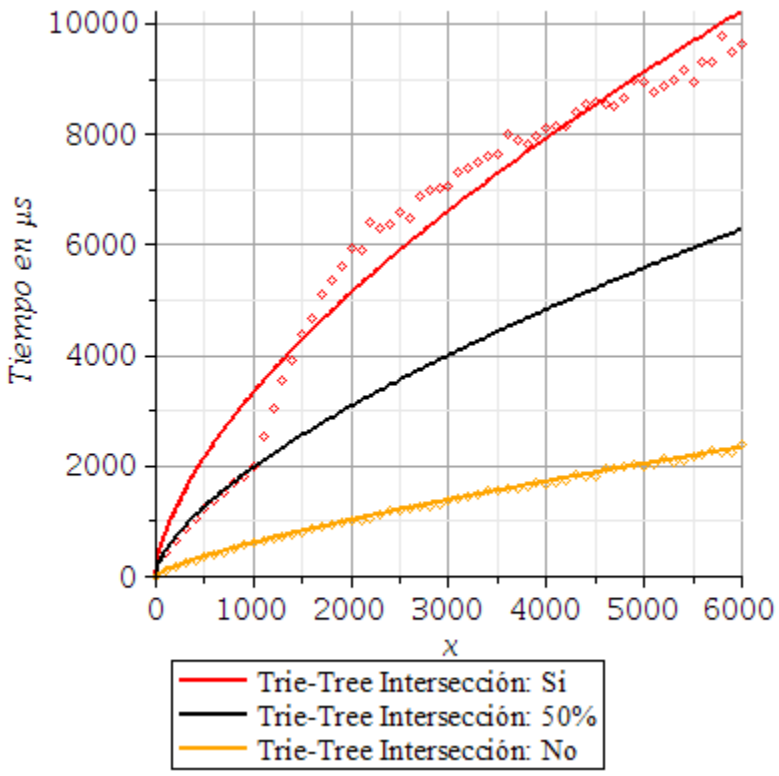
Fuente: Elaboración propia.

Igualmente, pese a que en esta estructura la búsqueda de planos es sencilla y además se eliminaron las copias de la estructura por medio de la utilización de apuntadores, la operación mergeXor, necesaria para realizar las operaciones

getSection y getPlv, realiza una construcción de un nuevo *Trie-Tree*, la cual requiere la construcción de nodos nuevos y la copia de planos o bordes.

La Figura 3 muestra el comportamiento de la operación de intersección entre dos EVM implementada con la estructura ABC-*Sorted*. Los puntos verdes muestran el tiempo promedio de la operación cuando existe una intersección entre *P* y *Q*, mientras que los puntos azules muestran el tiempo promedio de la operación cuando no existe dicha intersección. Estos puntos se muestran en la gráfica para *x* de 100 en 100.

Figura 4. Tiempos promedio en µs de la operación de intersección 3D sobre la estructura *Trie-Tree*, cuando existe y cuando no existe intersección.



Fuente: Elaboración propia.

Las líneas verde y azul son las líneas de tendencia, $f(x)$ y $g(x)$ correspondientes a los puntos verdes y azules respectivamente, las cuales se muestran en la Tabla 5. Si se realiza una suma ponderada de ambas líneas de la forma $h(x) = pf(x) + (1 - p)g(x)$, donde p es la probabilidad de que P y Q colisionen, se puede obtener la tendencia del tiempo promedio de la operación intersección. La línea negra muestra esta tendencia para $p = 0{,}5$. Las funciones $f(x)$, $g(x)$ y $h(x)$ pueden interpretarse como los resultados experimentales del peor caso, mejor caso y caso promedio respectivamente.

La Figura 4 es análoga a la Figura 3, con respecto a los tiempos promedio de la operación intersección para la estructura *Trie-Tree*.

En la Figura 5 se puede apreciar el comportamiento de la operación de intersección entre dos EVMs para las estructuras ABC-*Sorted* y *Trie-Tree*.

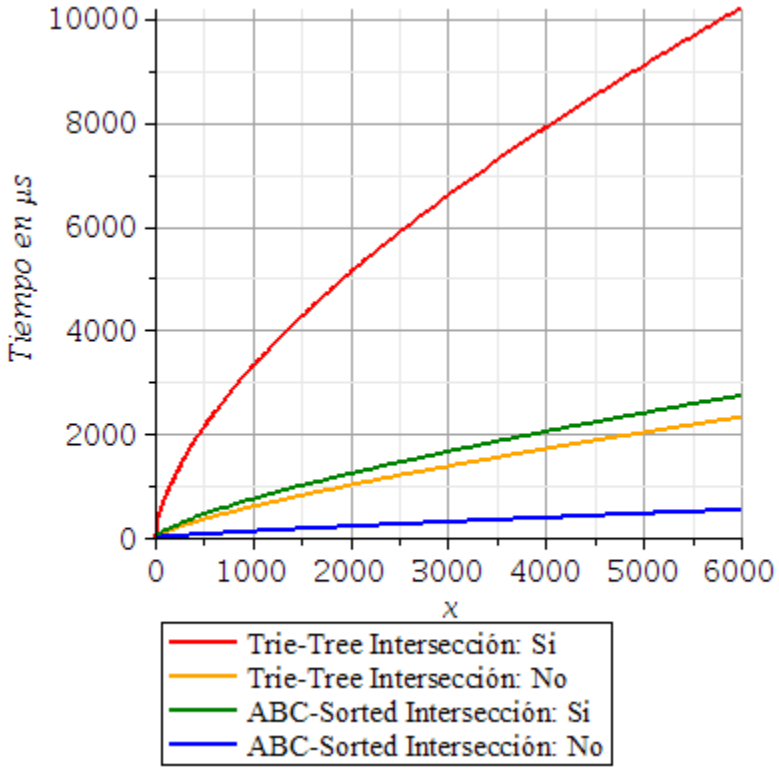
En general, se puede decir que la intersección implementada con ABC-*Sorted* presenta menor complejidad en tiempo que la implementada con *Trie-Trie*.

Tabla 5. Líneas de tendencia para los tiempos promedio en μs de las operaciones de intersección y colisión entre EVMs.

Intersección	**ABC-*Sorted***	**No**	$g(x) = 0{,}37266436343223475\ x^{0{,}8368259808295921}$
		Si	$f(x) = 4{,}899377409246352\ x^{0{,}7272849067147759}$
	Trie-Tree	**No**	$g(x) = 3{,}1851435925749723\ x^{0{,}7577425384702893}$
		Si	$f(x) = 42{,}71934460685103\ x^{0{,}6295199258377696}$
Colisión	**ABC-*Sorted***	**No**	$g(x) = 0{,}32225289305688865\ x^{0{,}8543446407614483}$
		Si	$f(x) = 0{,}8370219207896642\ x^{0{,}7242175171392802}$
	Trie-Tree	**No**	$g(x) = 3{,}4626144565901487\ x^{0{,}7485189896004869}$
		Si	$f(x) = 6{,}919927049287307\ x^{0{,}637837116441642}$

Fuente: Elaboración propia.

Figura 5. Comparación gráfica de los tiempos promedio de intersección 3D entre las estructuras ABC-*Sorted* y *Trie-Tree* en µs.



Fuente: Elaboración propia.

Análisis Experimental de la Complejidad en Tiempo de la Operación de Colisión entre EVMs

La finalidad de esta prueba es verificar si el tiempo promedio de colisión es mejor con respecto al tiempo promedio de intersección mediante la simplificación lograda en el algoritmo de detección de colisiones implementado en este trabajo.

Para esta prueba, se considera la colisión entre los EVMs *P* y *Q*, cuya suma de vértices extremos, $x = Card(EVM_3(P)) + Card(EVM_3(Q))$, varía entre 24 y 6000 al igual que en la prueba anterior. Para cada *x* y estructura de datos, se obtuvieron dos tiempos promedio, en microsegundos (μs), de la operación de colisión: (1) el tiempo promedio cuando no existe colisión entre *P* y *Q*, y (2) el tiempo promedio cuando si existe colisión entre *P* y *Q*. Cada tiempo promedio es resultado de 1000 observaciones aleatorias para un determinado caso, en las cuales la ubicación de los objetos varía. En la Tabla 6 se muestran estos resultados para *x* con una frecuencia de 500.

Como se puede notar en la Tabla 6, el tiempo promedio de colisión 3D del tipo EVM implementado con la estructura de datos ABC-*Sorted* es menor que el tiempo para el tipo EVM implementado con la estructura de datos *Trie-Tree* para todos los casos. Debido a que la operación de colisión se basa en la operación de intersección y sólo se simplifica para aquellos casos en los cuales *P* y *Q* se solapan, cuando no existe solapamiento entre *P* y *Q*, los tiempos de ambas operaciones son similares.

Un detalle interesante que puede notarse en la Tabla 6 es que los tiempos promedio de colisión cuando existe solapamiento son menores a los tiempos promedio de colisión cuando no existe solapamiento. Este comportamiento de la operación de colisión se debe a la salida adelantada que ocurre cuando se encuentra el primer borde solapado entre *P* y *Q*, lo cual evita que la operación continúe revisando el resto de los planos de vértices, situación que sucede cuando no existe solapamiento, en cuyo caso deben revisarse todos los planos del EVM que se encuentre a la izquierda.

Tabla 6. Comparación del tiempo promedio de colisión 3D entre las estructuras ABC-*Sorted* y *Trie-Tree* en µs.

	Tiempo Promedio de Colisión en µs			
	ABC-*Sorted*		***Trie-Tree***	
x	**No Existe**	**Si Existe**	**No Existe**	**Si Existe**
500	71,789	67,068	365,876	318,551
1000	116,365	104,506	597,704	489,115
1500	167,894	164,755	828,813	739,605
2000	212,272	209,813	1037,741	915,246
2500	256,921	248,983	1219,019	1049,218
3000	303,789	296,830	1394,399	1226,443
3500	344,819	307,420	1574,756	1249,163
4000	381,186	338,298	1689,252	1361,318
4500	417,726	373,299	1845,625	1476,998
5000	465,547	425,399	2021,242	1669,207
5500	499,960	432,141	2155,393	1728,775
6000	567,009	453,000	2397,118	1746,343

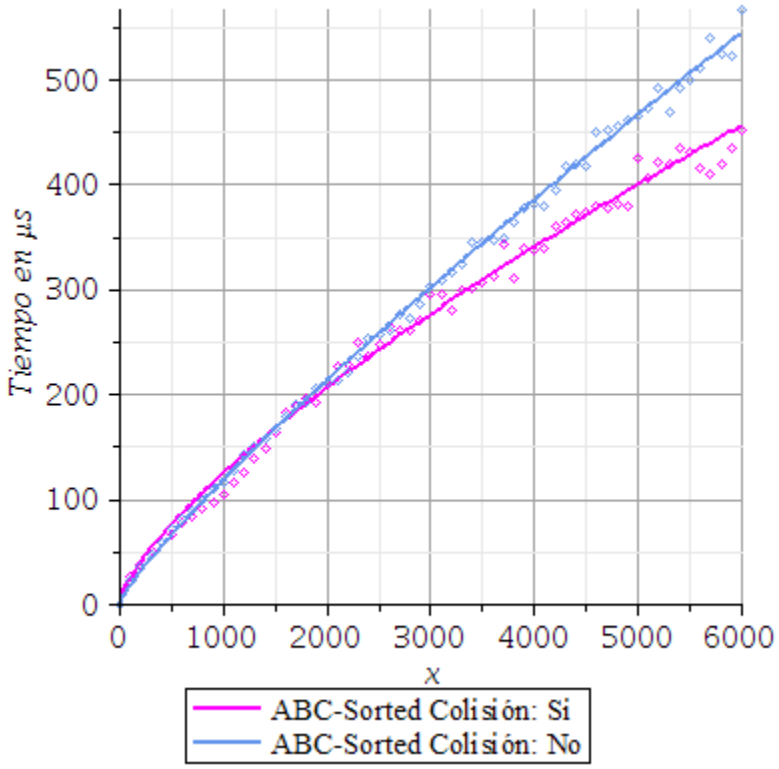
Fuente: Elaboración propia.

La Figura 6 muestra el comportamiento de la operación de colisión entre dos EVM implementada con la estructura ABC-*Sorted*.

Los puntos morados muestran el tiempo promedio de la operación cuando existe una colisión entre *P* y *Q*, mientras que los puntos azul claro muestran el tiempo promedio de la operación cuando no existe dicha colisión. Estos puntos se muestran en la gráfica para *x* de 100 en 100.

Las líneas en azul claro y morado son las líneas de tendencia, $f(x)$ y $g(x)$ correspondientes a los puntos azul claro y morados respectivamente, las cuales se muestran en la Tabla 5.

Figura 6. Tiempos promedio en µs de la operación de colisión 3D de la estructura ABC-*Sorted*, cuando existe y cuando no existe colisión.

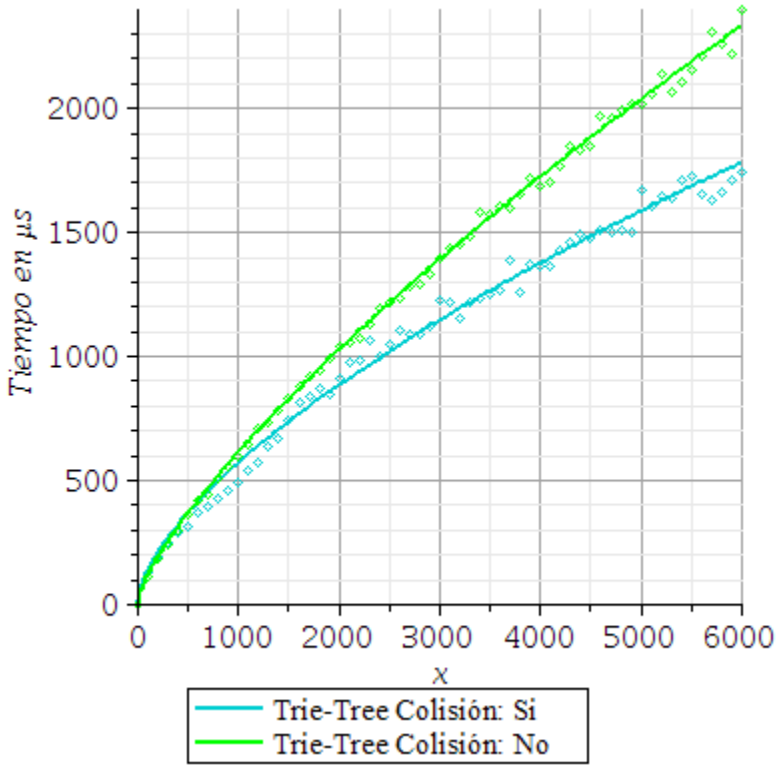


Fuente: Elaboración propia.

Nótese que en el caso de la operación de colisión, los tiempos promedio cuando existe o no solapamiento son similares hasta $x = 2000$, luego los tiempos promedio cuando existe solapamiento son menores. Esto quiere decir que el peor caso para la operación se da, en general, cuando no existe solapamiento entre los EVMs.

La Figura 7 es análoga a la Figura 6, con respecto a los tiempos promedio de la operación de colisión para la estructura *Trie-Tree*.

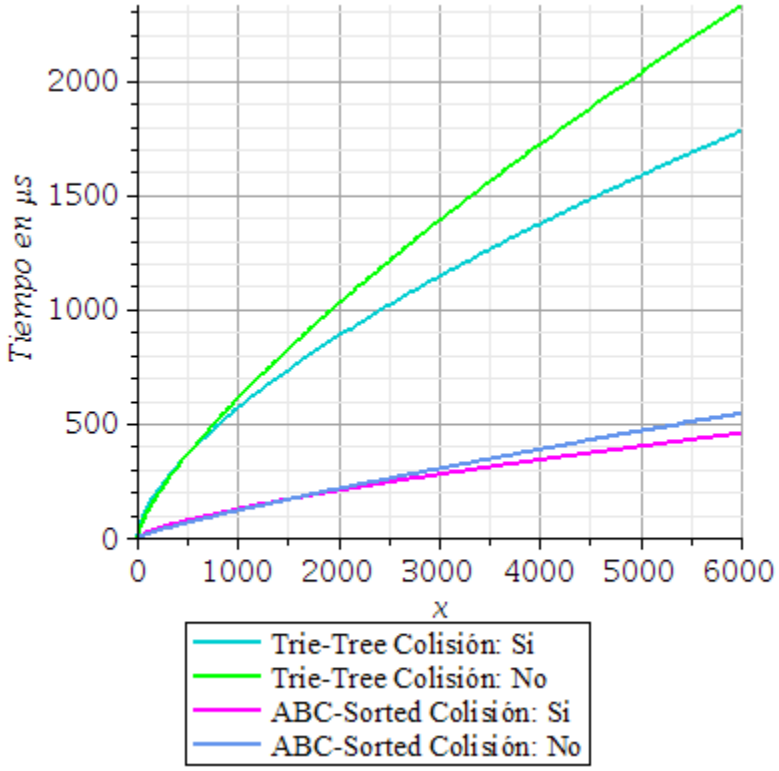
Figura 7. Tiempos promedio en µs de la operación de colisión 3D de la estructura *Trie-Tree*, cuando existe y cuando no existe colisión.



Fuente: Elaboración propia.

La Figura 8 muestra una comparación entre las líneas de tendencia de los tiempos promedio de colisión para el tipo EVM sobre las estructuras ABC-*Sorted* y *Trie-Tree*. En esta se puede apreciar como la operación de colisión implementada con la estructura ABC-*Sorted* presenta un mejor comportamiento que la operación implementada con *Trie-Tree*, es decir, queda confirmado que efectivamente el tiempo promedio de colisión para la estructura ABC-*Sorted* es menor que el tiempo de colisión para la estructura *Trie-Tree*, para todas las instancias.

Figura 8. Comparación gráfica de los tiempos promedio de colisión 3D entre las estructuras ABC-*Sorted* y *Trie-Tree* en µs.

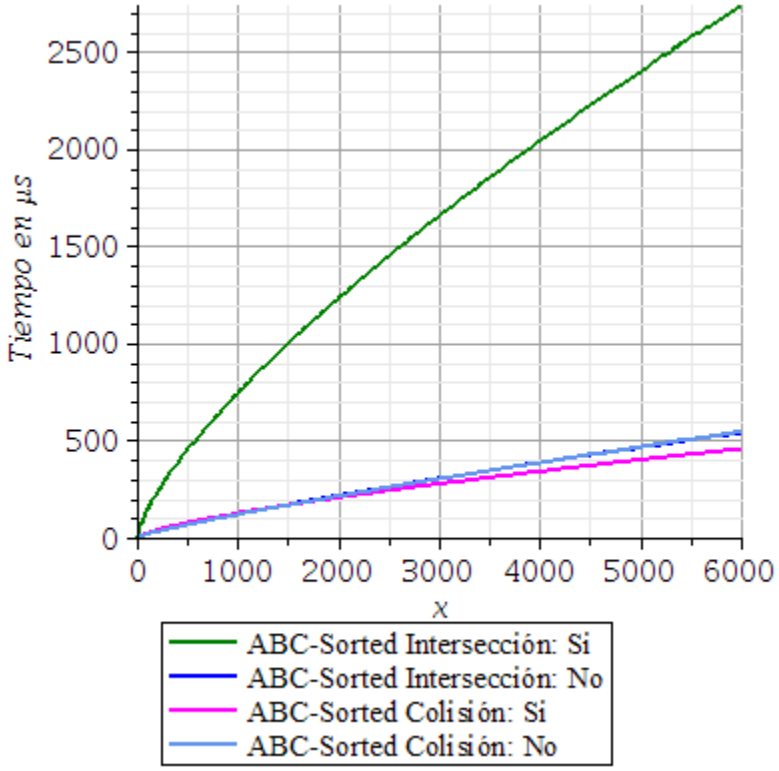


Fuente: Elaboración propia.

Las Figuras 9 y 10 muestran la comparación entre las operaciones de intersección y colisión para el tipo EVM implementado con ABC-*Sorted* y *Trie-Tree* respectivamente.

En la Figura 9, las líneas en azul y azul claro, que representan aquellos casos en los cuales *P* y *Q* no se solapan, son bastante similares y quedan una sobre otra. Para los casos donde existe solapamiento se puede notar que los tiempos promedio de la operación de intersección, en verde, son mucho mayores que los tiempos promedio de la operación de colisión, en morado.

Figura 9. Comparación gráfica de los tiempos promedio en µs de las operaciones de intersección y colisión 3D para la estructura ABC-*Sorted*.



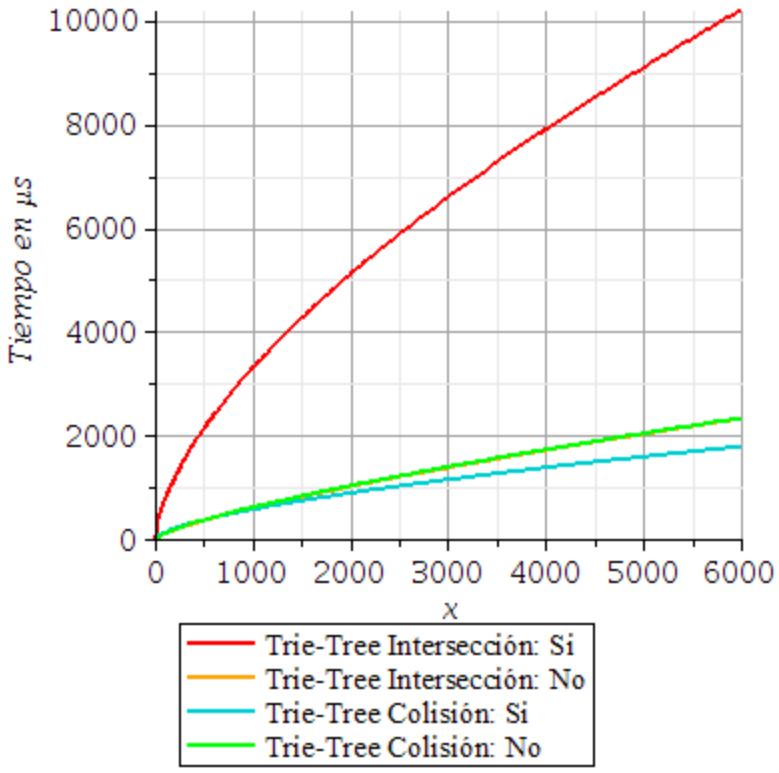
Fuente: Elaboración propia.

En la Figura 10, las líneas en amarillo y verde claro representan aquellos en los cuales *P* y *Q* no se solapan, por lo cual quedan una sobre otra. Para los casos donde existe solapamiento se puede notar que los tiempos promedio de la operación de intersección, en rojo, son mucho mayores que los tiempos promedio de la operación de colisión, en turquesa.

De estos resultados se puede concluir que el tiempo promedio de detección de colisiones con el algoritmo presentado en este trabajo cuando existe solapamiento entre los EVMs es mejor en comparación al tiempo promedio de la intersección, la

cual puede ser utilizada para detectar colisiones, debido a la salida adelantada que se realiza cuando se encuentra el primer borde solapado entre *P* y *Q*. Esto se debe a que, al detener la operación una vez que se consigue una colisión, no se deben recorrer todos los planos de la región crítica ni realizar las inserciones de planos en el EVM resultante, *R*, lo cual constituía una sobrecarga en la operación de intersección. En aquellos casos donde no hay solapamiento en los EVMs, ambas operaciones tienen el mismo comportamiento.

Figura 10. Comparación gráfica de los tiempos promedio en µs de las operaciones de intersección y colisión 3D para la estructura *Trie-Tree*.



Fuente: Elaboración propia.

La Tabla 7 muestra una comparación entre el tiempo promedio, cuando existe solapamiento entre los EVMs, de las operaciones de intersección y colisión para cada estructura de datos, así como el porcentaje de mejora presentado por la operación para el cálculo de colisiones.

Tabla 7. Comparación entre los tiempos promedio, cuando existe solapamiento entre los EVMs, de las operaciones de intersección y colisión 3D para las estructuras ABC-*Sorted* y *Trie-Tree*.

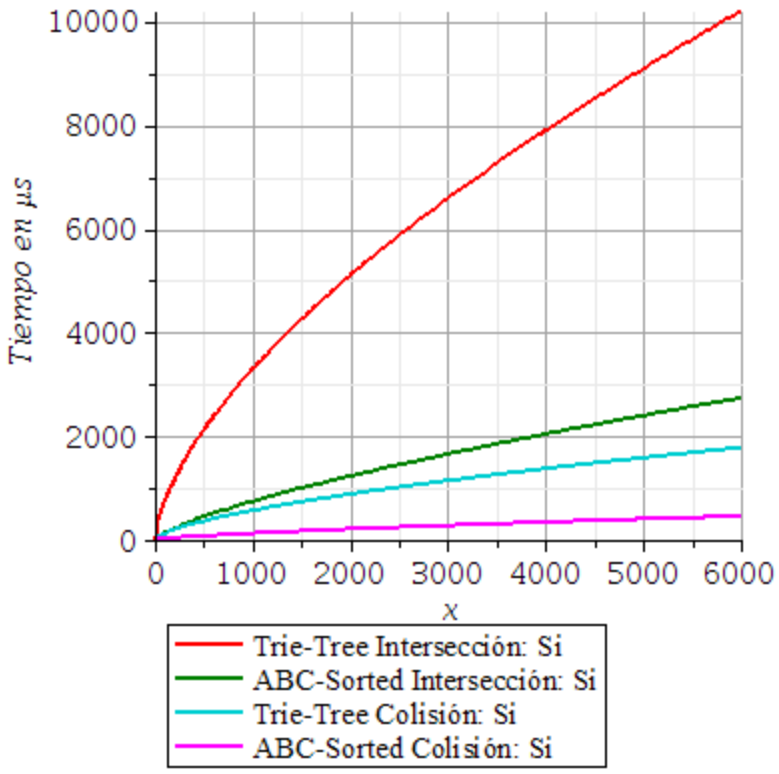
	ABC-*Sorted*			***Trie-Tree***		
x	**Intersec.**	**Colisión**	**Mejora**	**Intersec.**	**Colisión**	**Mejora**
500	259,685	67,068	74,173%	1217,993	318,551	73,846%
1000	442,062	104,506	76,359%	1993,424	489,115	75,464%
1500	1018,794	164,755	83,828%	4395,904	739,605	83,175%
2000	1395,977	209,813	84,970%	5948,518	915,246	84,614%
2500	1607,284	248,983	84,509%	6610,269	1049,218	84,127%
3000	1730,999	296,830	82,852%	7052,986	1226,443	82,611%
3500	1939,742	307,420	84,152%	7648,771	1249,163	83,668%
4000	2121,719	338,298	84,055%	8133,469	1361,318	83,263%
4500	2254,310	373,299	83,441%	8573,740	1476,998	82,773%
5000	2371,832	425,399	82,065%	8931,911	1669,207	81,312%
5500	2399,574	432,141	81,991%	8932,042	1728,775	80,645%
6000	2630,598	453,000	82,780%	9640,061	1746,343	81,885%

Fuente: Elaboración propia.

Con la estructura ABC-*Sorted*, la colisión presenta un tiempo menor que la intersección desde un 74,173% (para $x = 500$) hasta un 82,78% (para $x = 6000$). Por otro lado, con la estructura *Trie-Tree*, la colisión también presenta igualmente un tiempo menor que la intersección desde un 73,846% (para $x = 500$) hasta un 81,885% (para $x = 6000$). De estos resultados se puede intuir que la detección de colisiones realizada mediante el algoritmo presentado en este trabajo es considerablemente más rápida que la basada en la operación de intersección.

La Figura 11 muestra gráficamente el comportamiento de las operaciones de intersección y colisión para ambas estructuras cuando existe solapamiento entre *P* y *Q*. Se puede ver claramente como en ambos casos la operación de colisión presenta una tendencia de tiempo promedio menor que a la tendencia de la operación de intersección.

Figura 11. Comparación gráfica entre los tiempos promedio, cuando existe solapamiento, entre los EVMs, de las operaciones de intersección y colisión 3D para las estructuras ABC-*Sorted* y *Trie-Tree*.



Fuente: Elaboración propia.

Detección de Colisiones Mediante la Utilización del EVM como BV

Probar directamente si dos objetos poligonales colisionan a partir de su geometría es generalmente un proceso que requiere gran cantidad de cómputo, especialmente cuando los mismos están formados por cientos o miles de polígonos (Ericson, C., 2005: 75), por esta razón, es común el uso de *Bounding Volumes* (BVs) o *Bounding Volume Hierarchies* (BVHs).

Como se explicó en la sección Bases Teóricas, el modelo de los vértices extremos puede ser utilizado como un modelo intermedio de visualización y manipulación tanto de objetos volumétricos como de objetos geométricos que admitan una transformación a objetos volumétricos, entre ellos, aquellos objetos poligonales cuya primitiva de construcción es el triángulo. Dado que se puede construir un OPP de un objeto poligonal con diferentes niveles de detalle, o voxeles por unidad (VPU), y a partir de éste su EVM, es posible utilizar el EVM como un BV ajustado al objeto, el cual puede ser usado para disminuir la cantidad de casos en los que se debe probar la colisión basada directamente en la geometría del objeto.

La finalidad de esta prueba es evaluar el comportamiento del EVM como el BV de un objeto y establecer una comparación con el AABB obtenido del mismo objeto.

En esta prueba se utilizaron dos esferas unitarias, formadas por 362 vértices y 760 triángulos, dispuestas de manera tal no exista colisión entre ellas, tomando en cuenta uno de los peores casos para el algoritmo de detección de colisiones sobre XYZ_EVMs, es decir, una esfera sobre otra. Se eligió una esfera para esta prueba ya

que por poseer una superficie curva es más complicada de representar mediante OPPs, y en consecuencia genera una mayor cantidad de vértices extremos.

De cada esfera se obtuvo un AABB. El tiempo de obtención del AABB es considerado un tiempo de preprocesamiento para la detección de colisiones. Para 1000 observaciones, el tiempo promedio de preprocesamiento para el AABB correspondiente a una de las esferas fue de 30,813 μs, mientras que el tiempo promedio de detección de colisiones entre los AABBs de las esferas fue de 0,261 μs.

De manera referencial se obtuvo para esta prueba el resultado de aplicar la aproximación *naive* para la detección de colisiones entre dos objetos, utilizando la prueba de intersección triángulo-triángulo presentada por Möller, T. (1997: 25). Para éstas dos esferas, si no existe colisión, se realizan 760^2 pruebas triángulo-triángulo, lo cual lleva un tiempo promedio de 486.790,46 μs.

De cada esfera se obtuvieron 20 objetos volumétricos con niveles de detalle (de 1 a 20 VPU), mediante los algoritmos de voxelización presentados por Kaufman, A. y Shimony, E. (1985: 45), y sobre cada uno de ellos se generó un EVM mediante el algoritmo presentado en Rodríguez, J. (2004: 71).

Tanto el tiempo necesario para voxelizar el objeto como el tiempo para obtener su EVM son considerados tiempos de preprocesamiento. El tiempo de conversión de EVM a B-Rep es considerado un tiempo de postprocesamiento, y se debe tomar en cuenta en caso de realizarse la visualización del EVM.

Es importante destacar que todos los tiempos de preprocesamiento y postprocesamiento deben realizarse una sola vez, a menos que se realice una rotación sobre el objeto. Una vez que el BV del objeto ha sido construido, se puede realizar la prueba de colisión sobre el mismo tantas veces como sea necesario.

Tabla 8. Tiempos promedio en μs de preprocesamiento, colisión y postprocesamiento al usar EVM basado en ABC-*Sort* como BV sobre esferas de 760 triángulos.

		Tiempos Promedio en μs					
		Preproc.			**Postproc.**		
VPU	**# EV**	**Gen. del OPP**	**Gen. del EVM**	**Colisión**	**Gen. del B-Rep**	**# V**	**# F**
1	48	2.517,13	45,60	71,56	2.364,01	56	96
2	48	2.811,20	137,59	99,92	2.729,89	56	96
3	176	3.021,99	302,92	176,29	10.726,64	192	368
4	208	3.308,88	558,49	261,63	14.570,87	248	456
5	348	3.634,32	921,72	411,10	33.381,71	491	938
6	412	3.971,71	1.415,20	421,01	41.602,22	577	1088
7	526	4.385,68	2.035,52	582,25	65.226,91	798	1582
8	820	4.832,80	2.859,03	838,13	125.759,08	1200	2346
9	972	5.254,76	3.859,70	977,81	165.055,87	1437	2820
10	880	5.772,30	5.001,57	1.000,22	151.912,17	1404	2820
11	1350	6.355,37	6.479,75	1.311,14	285.489,75	1958	3930
12	1554	6.904,51	8.219,97	1.558,33	387.675,67	2485	4774
13	1670	7.524,13	10.150,25	1.621,81	450.245,18	2726	5268
14	2142	8.083,94	12.356,04	2.029,10	660.840,47	3303	6368
15	2356	8.908,74	15.057,35	2.259,32	831.098,12	3822	7616
16	2584	9.645,93	17.821,20	2.431,84	939.678,53	3962	7890
17	3436	10.396,51	21.212,82	3.044,94	1.530.740,15	4986	10010
18	3784	11.230,05	24.878,50	3.507,42	1.883.589,13	5776	11360
19	3806	12.010,77	28.685,18	3.467,65	1.869.969,21	5453	11580
20	4646	12.965,48	33.320,03	4.100,29	2.700.413,30	6853	13534

Fuente: Elaboración propia.

Las Tablas 7 y 8 muestran los resultados de esta prueba para el tipo EVM implementado con las estructuras ABC-*Sorted* y *Trie-Tree* respectivamente. En cada tabla una fila representa un nivel de detalle distinto medido en VPU.

Tabla 9. Tiempos promedio en µs de preprocesamiento y colisión al usar EVM basado en *Trie-Tree* como BV sobre esferas de 760 triángulos.

		Tiempos Promedio en µs		
		Preprocesamiento		
VPU	**# EV**	**Gen. del OPP**	**Gen. del EVM**	**Colisión**
1	48	2.517,13	63,691	338,873
2	48	2.811,20	163,383	486,675
3	176	3.021,99	365,152	717,652
4	208	3.308,88	664,470	1.158,829
5	348	3.634,32	1.078,549	1.679,595
6	412	3.971,71	1.645,868	1.773,262
7	526	4.385,68	2.287,901	2.492,021
8	820	4.832,80	3.225,494	3.442,772
9	972	5.254,76	4.305,534	4.048,409
10	880	5.772,30	5.500,680	4.228,657
11	1350	6.355,37	7.114,202	5.097,247
12	1554	6.904,51	8.997,419	6.468,538
13	1670	7.524,13	10.922,185	6.820,675
14	2142	8.083,94	13.602,024	7.969,057
15	2356	8.908,74	16.181,898	9.175,770
16	2584	9.645,93	19.235,802	9.696,170
17	3436	10.396,51	22.971,170	12.244,904
18	3784	11.230,05	26.905,584	14.402,984
19	3806	12.010,77	31.034,367	13.839,814
20	4646	12.965,48	36.008,932	16.533,475

Fuente: Elaboración propia.

Para cada nivel de detalle se tiene la cantidad de vértices extremos del EVM que representa el objeto, el tiempo promedio de voxelización del objeto, el tiempo promedio de obtención del EVM, el tiempo promedio de colisión y, únicamente en el

caso de la estructura ABC-*Sorted*, el tiempo promedio de obtención del B-Rep con sus respectivas cantidades de vértices y caras (triángulos).

Ambas tablas presentan tiempos de preprocesamiento similares, y como se puede notar, estos tiempos son relativamente pequeños para todos los casos mostrados, lo cual indica que el EVM puede ser utilizado como un BV para la detección de colisiones sin que esto implique una gran sobrecarga en cuanto a tiempo de procesamiento.

El tiempo de detección de colisiones presentado en las Tablas 8 y 9 para todos los casos es considerablemente pequeño si se compara con el tiempo promedio de la detección de colisiones triángulo-triángulo.

Se puede ver claramente que en términos de tiempo, el preprocesamiento necesario para construir el AABB del objeto es menor que el necesario para construir su EVM en todos los casos. Igualmente, el tiempo de detección de colisiones entre AABBs es mucho menor que el tiempo de detección de colisiones entre EVMs en todos los casos. Sin embargo, se debe tomar en cuenta que, aunque la utilización del EVM como BV del objeto para la detección de colisiones es más costosa a nivel de cómputo que la utilización del AABB, una ventaja que presenta el EVM con respecto al AABB es que al ajustarse a un objeto, incluso si este presenta una superficie irregular o huecos internos, reduce el espacio en el cual la detección de colisiones muestra falsos positivos en los cuales se debe realizar la detección de colisiones triángulo-triángulo.

Por ejemplo, si se toma en cuenta una esfera hueca de radio externo, R = 1.0, y radio interno, r = 0,955, su volumen es de aproximadamente 0,54 unidades cúbicas, mientras el volumen de su AABB es de 8 unidades cúbicas, por lo tanto un 93,24% del espacio representado por dicho AABB presentará falsos positivos. Si se obtiene el OPP de la misma esfera y se representa mediante su EVM puede reducirse el espacio que produce falsos positivos.

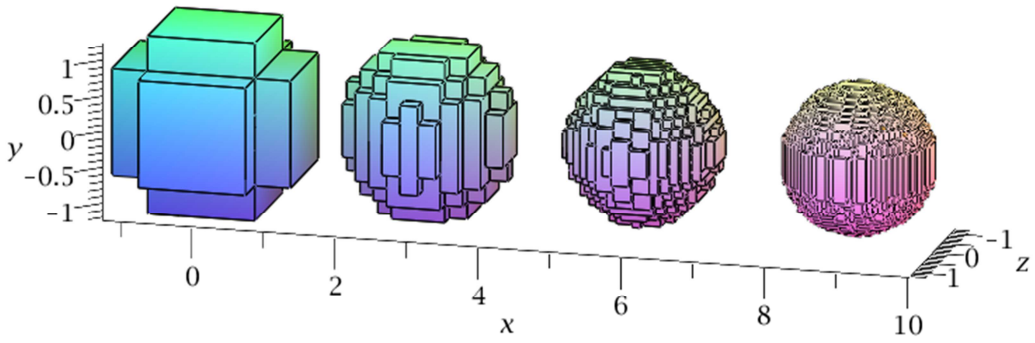
Tabla 10. Porcentaje del espacio ocupado por el EVM de una esfera de R = 1.0 y r = 0,995 que produce falsos positivos para diferentes niveles de detalle.

VPU	Volumen del EVM (u^3)	Espacio Desocupado (u^3)	% Espacio de Falsos Positivos
1	18,000	17,460	97,00%
2	8,750	8,210	93,82%
3	3,926	3,386	86,23%
4	3,219	2,678	83,21%
5	2,480	1,940	78,21%
6	1,968	1,427	72,53%
7	1,662	1,121	67,48%
8	1,383	0,842	60,92%
9	1,239	0,698	56,37%
10	1,095	0,555	50,65%
11	1,000	0,460	45,96%
12	0,925	0,385	41,60%
13	0,831	0,290	34,94%
14	0,789	0,249	31,50%
15	0,728	0,188	25,80%
16	0,691	0,151	21,84%
17	0,656	0,116	17,67%
18	0,612	0,071	11,67%
19	0,561	0,020	3,62%
20	0,559	0,018	3,30%

Fuente: Elaboración propia.

La Tabla 10 presenta, para veinte niveles de detalle distintos, el volumen ocupado por el EVM que representa la esfera del ejemplo, cuanto de ese volumen no está realmente siendo ocupado por la esfera real, y finalmente el porcentaje de espacio en el que se producen falsos positivos. Para 1 y 2 VPU el volumen ocupado por el EVM es mayor al volumen ocupado por el AABB de la esfera y por lo tanto el porcentaje del espacio que produce falsos positivos es mayor, sin embargo, a partir de 3 VPU el porcentaje del espacio en el que ocurren falsos positivos es menor que el presentado por el AABB, y va de 86,23% para 3 VPU a 3,3% para 20 VPU.

Figura 12. B-Rep obtenido del EVM de una esfera de radio 1.0, para 2, 4, 8 y 16 VPU.



Fuente: Elaboración propia.

En la Figura 12 se puede ver cómo, a medida en que se aumenta el nivel de detalle del EVM calculado, se va reduciendo el espacio en donde se producen falsos positivos.

Finalmente, con respecto al postprocesamiento, en la Tabla 8 se puede notar que el tiempo promedio necesario para la obtención del B-Rep del objeto a partir de su

EVM, es bastante costoso debido a que hace uso intensivo del reordenamiento del EVM. Igualmente se puede ver que a partir de 5 VPU, tanto el número de vértices y como el número de caras del B-Rep resultante son mayores que los números de vértices y caras del objeto original.

CONCLUSIONES Y RECOMENDACIONES

En esta investigación se desarrolló un algoritmo para la detección de colisiones entre dos objetos volumétricos con la finalidad de complementar el conjunto de operaciones del modelo de los vértices extremos. En vista de que la detección de colisiones podía realizarse tomando como base la operación booleana regularizada de intersección pero, al usarla se realizaban un conjunto de cálculos innecesarios los cuales implicaban un mayor tiempo para la detección de colisiones y en consecuencia limitaban la interactividad de las aplicaciones que hicieran uso del modelo de los vértices extremos, se diseñó un nuevo algoritmo, tomando como base el algoritmo existente para las operaciones booleanas regularizadas.

Una vez concluido el diseño del algoritmo, se inició la etapa de implementación, en la cual fue necesario el desarrollo de una serie de librerías de soporte para la clase EVM. Asimismo, para implementar la clase EVM, se tomaron en cuenta dos estructuras de datos: ABC-*Sorted* y *Trie-Tree*.

Al finalizar la implementación se realizaron las pruebas pertinentes para evaluar el desempeño del algoritmo. En todas las pruebas se tomaron en cuenta ambas estructuras de datos. En la primera prueba se evaluó el comportamiento de la operación de intersección como base para la detección de colisiones; en la segunda prueba se utilizó el algoritmo implementado para la detección de colisiones. En estas pruebas quedó demostrado que el algoritmo de detección de colisiones desarrollado en esta investigación presenta un tiempo de ejecución bastante menor que el mismo algoritmo basado en la operación booleana de intersección, con una mejora entre 74% para casos pequeños y 82% para casos grandes.

La tercera prueba presenta como el modelo de los vértices extremos con el nuevo algoritmo de detección de colisiones puede ser utilizado para modelar volúmenes envolventes de objetos basados en triángulos con la finalidad de disminuir la cantidad de veces que se necesita realizar la prueba de detección de colisiones directamente sobre los triángulos de los objetos.

Los resultados obtenidos en esta prueba revelan que, pese a que es más costoso a nivel de cómputo obtener el EVM de un objeto que obtener su AABB, la cantidad de falsos positivos (cuando se detecta una colisión y realmente esta no existe) se reduce considerablemente. Esta reducción se da debido a que el EVM presenta dos ventajas al ser usado como volumen envolvente: Primero, el EVM se puede generar tan ajustado al objeto como se necesite simplemente configurando el nivel de detalle, y segundo, el EVM puede representar objetos huecos, por lo que evita que sea detectado un falso positivo cuando un objeto se encuentra dentro de otro.

Por otro lado, de acuerdo a los resultados obtenidos en las pruebas, la implementación de la clase EVM basada en la estructura de datos ABC-*Sorted* resultó ser la más conveniente para implementar las operaciones de colisión e intersección para la clase EVM, por lo que se propone como trabajo futuro extender las pruebas a todas las operaciones booleanas regularizadas a fin de concluir una comparación entre ambas estructuras de datos. Asimismo, también se propone realizar la implementación de la clase EVM en un lenguaje de programación orientado a la Web, como por ejemplo JavaScript, y realizar las comparaciones pertinentes, dado el advenimiento de HTML5 y sus nuevas APIs, en particular WebGL, que permiten realizar visualización y manipulación de volúmenes en navegadores Web.

REFERENCIAS BIBLIOGRÁFICAS

Aguilera, Antonio. (1998). **Orthogonal Polyhedra: Study and Application**. Tesis doctoral, Universitat Politècnica de Catalunya, España.

Baciu, George y Wong, Wingo S. K. (2003). **Image-Based Techniques in a Hybrid Collision Detector**. IEEE Transactions on Visualization and Computer Graphics. 9(2), 254-271.

Ericson, Christer. (2005). **Real-Time Collision Detection**. San Francisco, CA: Elsevier.

Fan, Zhaowei, Wan, Huagen y Gao, Shuming. (2003). **IBCD: A fast Collision Detection Algorithm Based on Image Space Using OBB**. The Journal of Visualization and Computer Animation. 14(4), 169-181.

Faure, François, Barbier Sébastien, Allard, Jérémie y Falipou, Florent. (2008). **Image-based Collision Detection and Response between Arbitrary Volume Objects**. ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 155-162. Dublin, Irlanda.

Florez, Roberto. (2005). **Algoritmos, Estructuras de Datos y Programación Orientada a Objetos**. Bogotá: Ecoe Ediciones.

Fosdick, Lloyd D., Jessup, Elizabeth R., Schauble, Carolyn J. C. y Domik, Gitta. (1996). **An Introduction to High-Performance Scientific Computing**. Massachusetts: MIT Press.

Gómez, Marcelo. (2006). **Introducción a la Metodología de la Investigación Científica**. Córdoba: Brujas.

Govindaraju, Naga K., Redon, Stephane, Lin, Ming C. y Manocha, Dinesh. (2003). **CULLIDE: Interactive Collision Detection Between Complex Models in Large Enviroments Using Graphics**. Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. 25-32. San Diego, California.

Heidelberger, Bruno, Teschner, Matthias y Gross, Markus. (2004). **Detection of Collisions and Self-collisions Using Image-space Techniques**. Journal of WSCG. 145-152.

Hernández, Roberto, Fernández, Carlos y Baptista, Pilar. (2010). **Metodología de la Investigación**. México: McGraw-Hill.

Huang, Jian, Li, Yan, Crawfis, Roger, Lu, Shao-Chiung y Liou, Shuh-Yuan. (2001). **A Complete Distance Field Representation**. Proceedings of the conference on Visualization '01. 247-254. Washington D.C.

James, Doug y Pai, Dinesh. (2004). **BD-tree: Output-sensitive Collision Detection for Reduced Deformable Models**. ACM Transactions on Graphics. 23(3), 393-398.

Jang, Hanyoung y Han, JungHyung. (2008). **Fast Collision Detection using the A-buffer**. The Visual Computer: International Journal of Computer Graphics. 24(7), 659-667.

Jiménez, Juan y Segura, Rafael. (2008). **Collision Detection between Complex Polyhedra**. Computers and Graphics. 32(4), 402-411.

Kaufman, Arie y Shimony, Eyal. (1985). **3D Scan-Conversion Algoritms for Voxel-Based Graphics**. Proceedings of the 1986 Workshop on Interactive 3D Graphics. 45-75.

Kim, Ho Kyung, Guibas, Leonidas y Shin, Sung Yong. (2005). **Efficient Collision Detection among Moving Spheres with Unknown Trajectories**. Algoritmica. 43(3), 195-210.

Kim, Sujeong, Redon, Stephane y Kim, Young J. (2008, Marzo). **Continuous Collision Detection for Adaptive Simulation of Articulated Bodies**. The Visual Computer: International Journal of Computer Graphics. 24(4), 261-269.

Larsson, Thomas y Akenine-Möller, Tomas. (2006, Junio). **A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection**. Computers and Graphics. 30(3), 450-459.

Liu, Peiran, Shen, Xiaojun, Georganas, Nicolas y Roth, Gerhard. (2005). **Multi-Resolution Modeling and Locally Refined Collision Detection for Haptic Interaction**. Proceedings of the Fifth International Conference on 3D Digital Imaging and Modeling. 581-588.

Moller, T. (1997). **A Fast Triangle-Triangle Intersection Test**. Journal of Graphics Tools. 2(2), 25-30.

Ottogalli, Kiara, Martínez, Amadis y León Luis. (2011). **NASPOO: Una Notación Algorítmica Estándar para la Programación Orientada a Objetos**. Télématique. 10(1). 81-102.

Pérez-Águila, Ricardo. (2006). **Orthogonal Polytopes: Study and Application**. Puebla: Universidad de las Américas.

Rodríguez, Jorge. (2004). **Contribution to Surface/Volume Integration: A Model for Visualization and Manipulation**. Barcelona: Universitat Politècnica de Catalunya.

Rodríguez, Jorge y Ayala, Dolors. (2006). **Speed-Up of EVM-Based Boolean Operators**. Faraute de Ciencias y Tecnología. 1(1), 50-56.

Sabino, Carlos. (1992). **El Proceso de Investigación**. Caracas: Panapo.

Stevens, Roger T. (1993). **Quick Reference to Computer Graphics Terms**. Boston: Academic Press Professional.

Tamayo y Tamayo, Mario. (2003). **El Proceso de la Investigación Científica**. México, D. F.: LIMUSA.

Teschner, M., Kimmerle, S, Heidelberger, B., Zachmann, G., Raghupathi, L., Fuhrmann, A., Cani, M. P., Faure, F., Magnenat-Thalmann, N., Strasser, W. y Volino, P. (2005, Marzo). **Collision detection for Deformable Objects**. Computer Graphics Forum. 24(1), 61-81.

Zhang, Xinyu y Kim, Young J. (2007, Marzo). **Interactive Collision Detection for Deformable Models using Streaming AABBs**. IEEE Transactions on Visualization and Computer Graphics. 13(2). 318-329.

ANEXOS

Anexo A: Visualizador de Colisiones entre Objetos

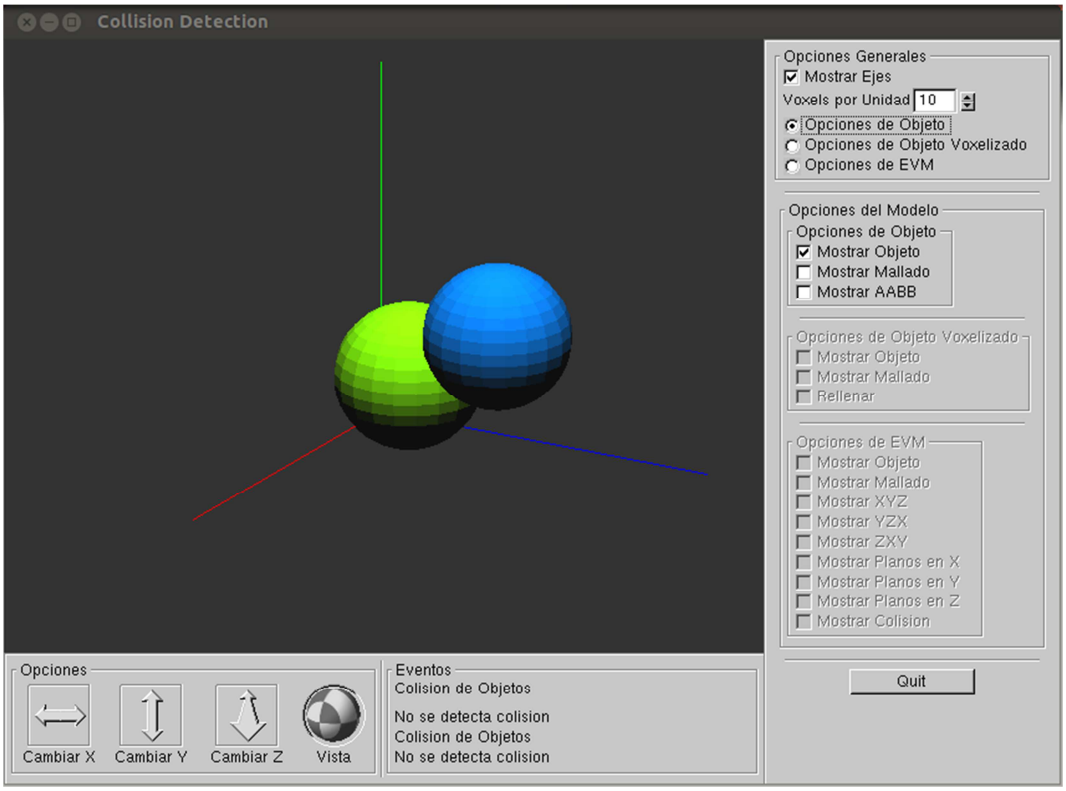
Durante el desarrollo de la investigación se creó un *software* visualizador de colisiones entre objetos, con la finalidad de facilitar la verificación del funcionamiento del algoritmo de detección de colisiones entre objetos volumétricos modelados mediante el Modelo de los Vértices Extremos.

Este *software* fue creado en C++ con OpenGL (Open Graphics Library) bajo el sistema operativo Linux en su distribución Ubuntu. Para la visualización de los objetos se utilizaron las librerías gl y glut, y para el desarrollo de la interfaz gráfica se utilizó glui.

La interfaz gráfica de este *software* se muestra en la Figura 13. Como se puede ver, la interfaz cuenta con un panel de opciones de la escena, un panel de opciones generales y un panel de eventos.

El panel de opciones de la escena permite mover uno de los objetos presentes en la escena, con la finalidad de generar colisiones en distintas posiciones, y así verificar el correcto comportamiento del algoritmo de detección de colisiones. Este panel consta de cuatro botones interactivos: El primero, Cambiar X, permite trasladar el objeto sobre el eje X; el segundo, Cambiar Y, permite trasladar el objeto sobre el eje Y; el tercero, Cambiar Z, permite trasladar el objeto sobre el eje Z; y el cuarto, Vista, permite rotar la escena para visualizar diferentes ángulos.

Figura 13. Interfaz gráfica del visualizador de colisiones.



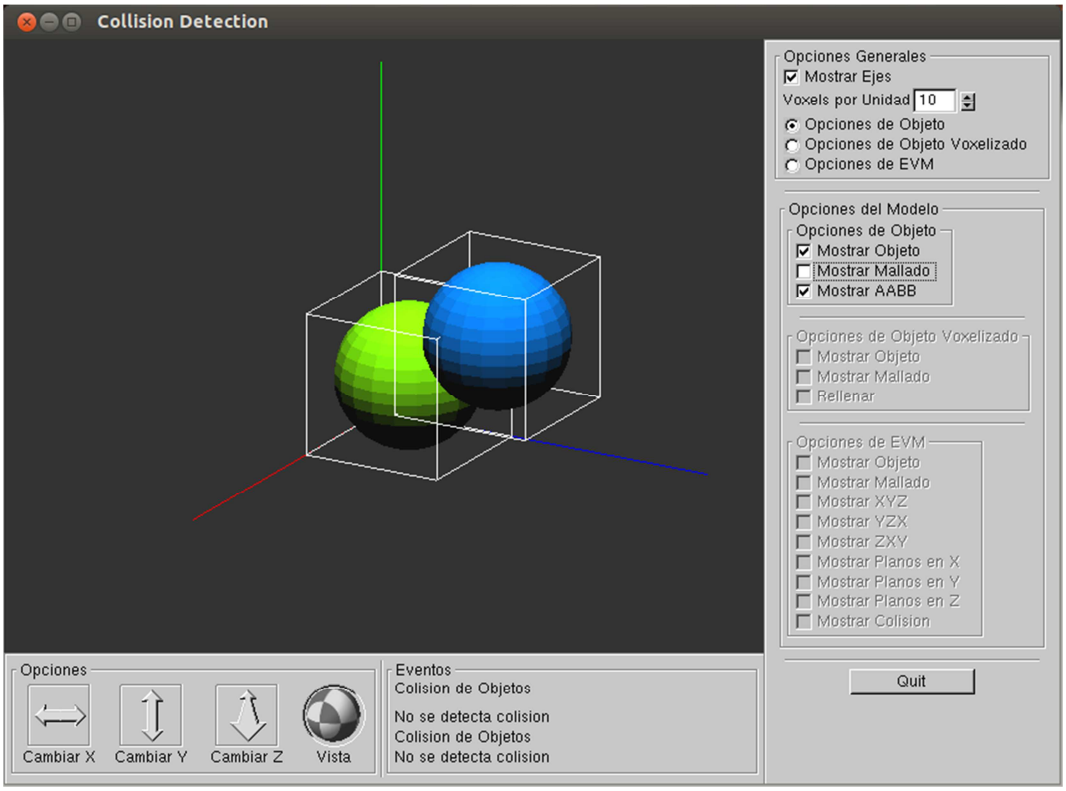
Fuente: Elaboración propia.

El panel de opciones generales permite la visualización de objetos modelados de tres formas diferentes: (1) Objetos tridimensionales basados en triángulos, (2) objetos voxelizados y (3) Representación de frontera (B-Rep) del EVM de un objeto voxelizado. Este panel está relacionado con el subpanel de opciones del modelo, ya que dependiendo del modelo que se esté visualizando, se presentan diferentes opciones.

Al seleccionar "opciones de objeto" en el panel de opciones generales, se activa el subpanel con el mismo nombre, el cual se encuentra dentro del subpanel de opciones del modelo. Este subpanel presenta tres opciones: (1) Mostrar objeto, que

permite mostrar u ocultar el objeto basado en triángulos, (2) Mostrar mallado, que permite mostrar u ocultar el mallado del objeto, y (3) mostrar AABB, que permite mostrar u ocultar el AABB del objeto (Figura 14).

Figura 14. Vista del AABB del objeto visualizado.

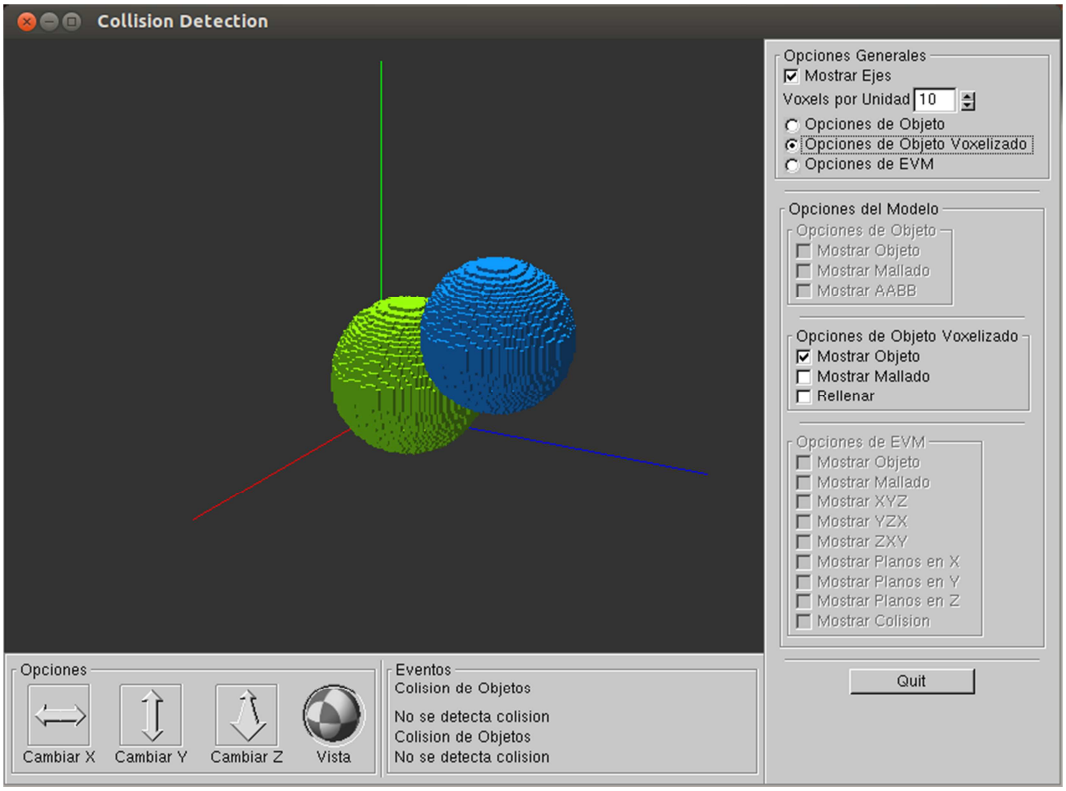


Fuente: Elaboración propia.

Al seleccionar “opciones de objeto voxelizado” en el panel de opciones generales, se activa el subpanel con el mismo nombre, el cual se encuentra dentro del subpanel de opciones del modelo. Este subpanel presenta tres opciones: (1) Mostrar objeto, que permite mostrar u ocultar el objeto voxelizado, en cuyo caso cada voxel es representado como un cubo, (2) Mostrar mallado, que permite mostrar u ocultar el

mallado del objeto voxelizado, y (3) Rellenar, que permite representar los voxeles internos del objeto como si fueran sólidos (Figura 15).

Figura 15. Vista de un objeto voxelizado.



Fuente: Elaboración propia.

Al seleccionar "opciones de EVM" en el panel de opciones generales, se activa el subpanel con el mismo nombre, el cual se encuentra dentro del subpanel de opciones del modelo. Este subpanel presenta nueve opciones: (1) Mostrar objeto, que permite mostrar u ocultar el EVM de un objeto, (2) Mostrar mallado, que permite mostrar u ocultar el mallado del EVM de un objeto, (3) Mostrar XYZ, permite visualizar los *brinks* del XYZ_EVM del objeto, (4) Mostrar YZX, permite visualizar los *brinks* del YZX_EVM del objeto, (5) Mostrar ZXY, permite visualizar los *brinks* del

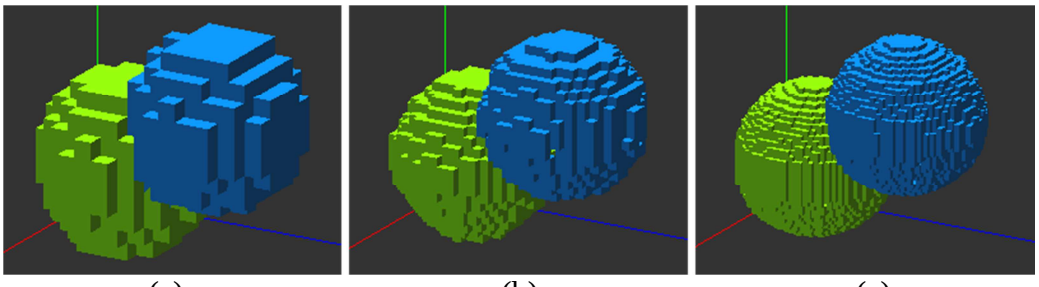
ZXY_EVM del objeto, (6) Mostrar planos en X, permite visualizar los planos del EVM perpendiculares a X, (7) Mostrar planos en Y, permite visualizar los planos del EVM perpendiculares a Y, (8) Mostrar planos en Z, permite visualizar los planos del EVM perpendiculares a Z, y (9) Mostrar colisión, que permite visualizar el área de colisión entre los objetos presentes en la escena.

Cualquier combinación de opciones en cualquier subpanel activo presente en Opciones del Modelo es permitida.

Finalmente se tiene un panel de Eventos, que genera avisos en tiempo real cada vez que ocurre una colisión entre EVMs, y si es así, verifica si esta colisión se da en los objetos basados en triángulos.

Vale la pena destacar que entre las opciones generales, cuando se activan las opciones de objeto voxelizado o de EVM, hay una opción que permite cambiar los voxeles por unidad, para obtener diferentes niveles de detalle del mismo objeto basado en triángulos (Figura 16).

Figura 16. Diferentes niveles de detalle: (a) 2 VPU, (b) 4 VPU y (c) 8 VPU.



(a) (b) (c)

Fuente: Elaboración propia.

Anexo B: Formato de un Archivo de EVM

Los objetos modelados con EVM se guardan en archivos de extensión .evm, cuyo formato se muestra en la Tabla 11. Estos archivos están distribuidos como sigue:

Tabla 11. Un archivo .evm

1	#EV: 12 Order: ZYX
2	
3	0.000000 0.000000 0.000000
4	2.000000 0.000000 0.000000
5	0.000000 1.000000 0.000000
6	1.000000 1.000000 0.000000
7	1.000000 2.000000 0.000000
8	2.000000 2.000000 0.000000
9	0.000000 0.000000 2.000000
10	2.000000 0.000000 2.000000
11	0.000000 1.000000 2.000000
12	1.000000 1.000000 2.000000
13	1.000000 2.000000 2.000000
14	2.000000 2.000000 2.000000
15	

Fuente: Elaboración Propia.

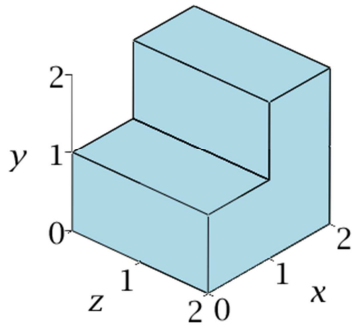
– La primera línea contiene la cadena de caracteres “#EV:” seguida de un espacio en blanco, el número, *n*, de vértices extremos que forman el objeto, un espacio en blanco, la cadena de caracteres “Order:” seguida de un espacio en blanco y el orden ABC del objeto.

– La segunda línea es una línea en blanco.

– Cada una de las siguientes *n* líneas representa las coordenadas *x*, *y* y *z* de uno de los vértices extremos del objeto, y está formada por tres números reales, separados por espacios en blanco.

– La última línea queda en blanco.

Figura 17. B-Rep del EVM de un objeto.



Fuente: Elaboración Propia.

La Figura 17 muestra el objeto voxelizado correspondiente a la Tabla 11.

Anexo C: Formato de un Archivo de Voxelización

Los objetos volumétricos se guardan en archivos de extensión .vox, cuyo formato se muestra en la Tabla 12. Estos archivos están distribuidos como sigue:

Tabla 12. Un archivo .vox

1	MIN 0 0 0
2	MAX 1 1 1
3	
4	0 1
5	1 1
6	
7	0 1
8	1 1
9	

Fuente: Elaboración Propia.

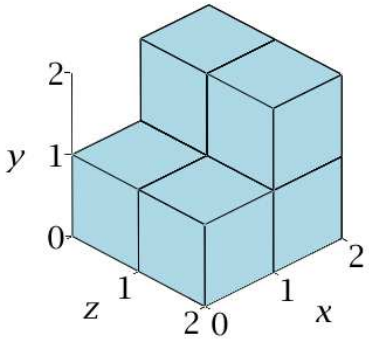
– La primera línea contiene la cadena de caracteres “MIN” y un espacio en blanco seguido de tres enteros x_1, y_1 y z_1 que representan el punto mínimo del objeto voxelizado.

– La segunda línea contiene la cadena de caracteres “MAX” y un espacio en blanco seguido de tres enteros x_2, y_2 y z_2 que representan el punto máximo del objeto voxelizado.

– La tercera línea es una línea en blanco.

– Las siguientes líneas están formadas por $z_2 - z_1 + 1$ bloques de $y_2 - y_1 + 1$ líneas seguidas de un espacio en blanco. Cada bloque representa un *slice* del objeto en el eje de coordenadas *Z*.

Figura 18. Objeto voxelizado.



Fuente: Elaboración Propia.

La Figura 18 muestra el objeto voxelizado correspondiente a la Tabla 12.