UNIVERSIDAD NACIONAL

AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

APLICACIÓN DE ALGORITMOS DE COLISIÓN PARA MODELOS TRIDIMENSIONALES

TESIS

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN

PRESENTA:

CAZAÑAZ BENÍTEZ JASSIEL

DIRECTOR DE TESIS

ING. RODRIGO GUILLERMO TINTOR PÉREZ

MÉXICO D.F. 2010

Índice

Índice de figuras

Introducción

Actualmente el uso de aplicaciones de computo que involucran modelos tridimensionales, ya sean estas aplicaciones de realidad virtual o videojuegos, tienen la necesidad de determinar si dos o más de estos modelos están colisionando dentro de un ambiente tridimensional, debido a esto surgen técnicas como el uso de volúmenes de colisión, ya sean envolventes o en base a la geometría.

Estos volúmenes envolventes de colisión, son calculados a partir del conjunto de vértices que componen a un modelo tridimensional, pero dichos modelos difícilmente se ajustan a la geometría del modelo tridimensional, o hacen un uso excesivo de recursos de cómputo ya sean de software o de hardware.

Debido a esta problemática el objetivo de este trabajo es el desarrollar algoritmos y herramientas, que optimicen los recursos de cómputo usados para hacer el cálculo de colisiones entre modelos tridimensionales, sin sacrificar la exactitud al hacerlo, y que puedan satisfacer todas las necesidades de las aplicaciones de realidad virtual o videojuegos.

Tomando en cuenta estas necesidades a lo largo de este trabajo se explicarán los conceptos relacionados con esta problemática y como en base a estándares y teorías se desarrollaron varios algoritmos y una metodología que es mostrada a través del desarrollo de una aplicación que es capaz de ofrecer una alternativa para resolver la problemática mencionada.

Esta aplicación fue desarrollada apegándose a los estándares de aplicaciones de diseño de modelos tridimensionales y haciendo uso de tecnologías estándar entre estos y las usadas para el desarrollo de aplicaciones tanto de videojuegos como de realidad virtual, buscando con esto que el uso de esta herramienta sea muy sencillo e intuitivo para los usuarios acostumbrados a usar este tipo de aplicaciones, además de que sea lo más portable posible y que no necesite de un amplio uso de recursos de computo.

Capítulo 1

La Computación Gráfica

1. Los gráficos por computadora

La computación gráfica es una rama de la computación que tiene como objetivo el representar, mediante recursos de cómputo, una serie de datos de manera visual permitiendo así una interpretación más natural para los usuarios de la información proporcionada por una computadora.

Ya que la computación gráfica usa elementos visuales para representar información, existen diversos campos donde es aplicada como el desarrollo de interfaces de usuarios, la representación grafica de datos, la cartografía, la medicina, el diseño asistido por computadora, la representación gráfica de datos científicos, el desarrollo de ambientes virtuales, el procesamiento digital de imágenes, entretenimiento, arte, educación, entre otros.

El dispositivo de salida más común que usan las computadoras para desplegar la información visual que se procesa es el monitor basado en un método de barrido (raster) [FOLEY96]. Este método consiste en que un cañón de electrones barre la pantalla, entonces la imagen se va formando a partir de una cuadricula de puntos, y a cada punto se le domina pixel (de picture element).

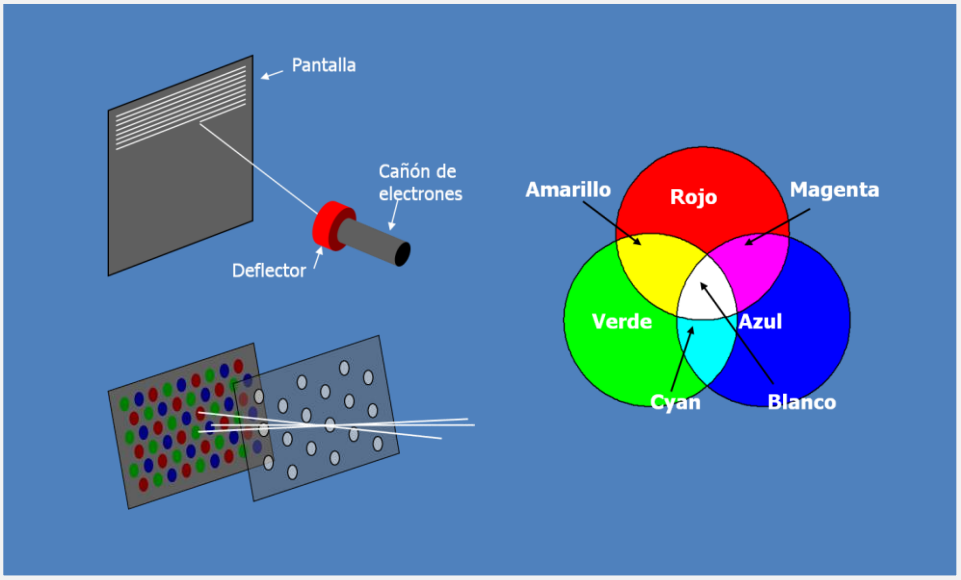


Figura 1: Método de raster para el modelo RGB

Para la representar los colores que conforman la imagen desplegada en un monitor se usan tres colores primarios: el rojo, el verde y el azul, que al combinarlos en diferentes intensidades logran representar los colores deseados. A este método se le denomina modelo RGB (Red, Green, Blue).

1.1 Gráficos bidimensionales

Los gráficos 2D están conformados por primitivas básicas como puntos, líneas, y polígonos que pueden ser directamente representados mediante el método de raster, y se les denomina mapa de bits los cuales contienen información sobre color y coordenadas de los puntos que conforman la imagen.

También es posible representar los gráficos 2D mediante vectores que almacenan datos geométricos precisos, topología y estilo, así como las coordenadas de los puntos, las uniones entre puntos para formar curvas y el color, el grosor y posible relleno de las formas. Pero a diferencia de los mapas de bits es necesario realizar un proceso previo para después realizar el raster.

1.2 Gráficos tridimensionales

Los gráficos tridimensionales están conformados por primitivas 2D básicas a las cuales se les añade un factor de profundidad. En base a este concepto se pueden generar diferentes tipos de primitivas tridimensionales básicas como líneas, polígonos, curvas y superficies. Para que en base a la información de los elementos que conforman a un objeto tridimensionales se pueda generar un mapa de bits, es necesario calcular los elementos que son visibles de acuerdo a su profundidad.

Los gráficos tridimensionales tienen como objetivo el representar la información visual de los objetos lo más aproximada a la realidad por lo que es necesario involucrar conceptos como sombreado, iluminación y texturizado que permiten obtener una imagen final capaz de representar objetos y ambientes reales.

1.2.1 Sombreado

En el caso de los objetos tridimensionales, construidos en base a polígonos, es posible aplicar métodos de sombreado en base a un modelo de iluminación; existen tres tipos de sombreado de uso común, el sombreado constante, de Gouraud, y Phong [FOLEY96].

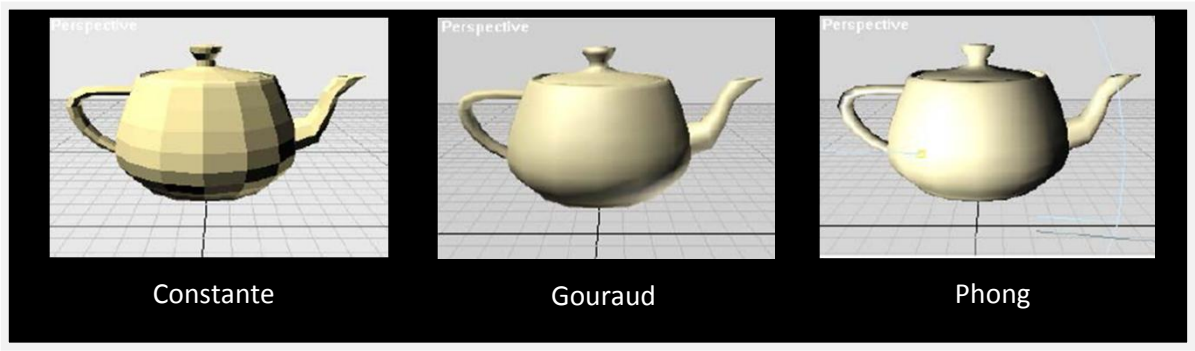


Figura 2: Tipos de sombreado

Sombreado constante: En este método en cada cara de la malla poligonal tiene mismo color y la misma intensidad de a cuerdo a la fuente de iluminación. Es muy usado ya que provee un cálculo muy rápido de la representación.

Sombreado de Gouraud: En este método se realiza una interpolación de color entre cada vértice que conforma la malla poligonal. Solo es usado en iluminación ambiental y difusa.

Sombreado de Phong: En este método se interpola el vector normal de vértice de la malla poligonal. Es el método más usado ya que simula superficies más realistas para todo tipo de iluminación.

1.2.2 Iluminación

Al igual que en un ambiente real, en un ambiente tridimensional la iluminación juega un papel muy importante ya que de esta afecta el color y apariencia de un objeto. Otro factor que afecta un ambiente es la manera en que lo objetos reflejan o transmiten la luz. También depende del tipo de fuente de la luz, que puede ser puntual, distribuida, omnidireccional, o direccional.

Un modelo de iluminación puede definirse como el método usado para definir la cantidad de luz que se refleja o se transmite, en cada punto de un objeto desde un punto de vista especifico. Existen tres modelos de iluminación, el ambiental, el difuso y el especular.

- **Iluminación ambiental:** En un modelo de iluminación ambiental los objetos se ven igualmente iluminados desde cualquier punto de vista, la fuente de luz no tiene una dirección definida, y se ve afectada por la reflexión y transmisión debida a los objetos que componen la escena.

- **Reflexión difusa:** En este método de iluminación, la luz se refleja en todas direcciones, en este caso los objetos parecen plásticos.
- **Reflexión especular:** Para el caso de la iluminación especular, la luz que incide en un objeto es totalmente reflejada lo que da la apariencia a los objetos de ser metálicos.

Para calcular un modelo de iluminación global es necesario tomar en cuenta todos los modelos de iluminación y su comportamiento de acuerdo a todos los elementos que componen la escena.

1.2.3 Materiales y texturas

Para lograr que la imagen rasterizada obtenida a partir de un modelo tridimensional sea lo mas cercana a la realidad, a los objetos se les asigna materiales y texturas. Estos le indican al software encargado de generar una representación basada en imagen, como se debe comportar la luz al incidir en los objetos, simulando como sucede en la realidad en el caso de materiales, o el mapeo de texturas, es decir asignación de mapas de bits a determinadas caras de un polígono u superficie del modelo tridimensional.

El uso de la simulación de materiales y el mapeo de texturas permite obtener una representación muy cercana, a como se ven diversos materiales en el mundo real, lo que ha generado el concepto conocido como fotorealismo, que indica que una imagen generada por computadora a partir de una escena tridimensional es lo más aproximada a una imagen real.

1.2.4 Representación basada en imagen

Existen diferentes algoritmos para realizar este proceso dependiendo de los recursos de hardware y software, y si se necesita que la imagen rasterizada se genere en tiempo real o que se necesite una representación muy realista. Como se ha visto están involucrados muchos conceptos como la iluminación, el sombreado, la simulación de materiales y el texturizado; todos estos deben de ser tomados en cuenta para poder generar una representación basada en imagen.

Shader: Este método consiste en un algoritmo que calcula la apariencia en un objeto tridimensional en tiempo real, siendo capaz de obtener efectos de iluminación muy realistas. Este método es muy usado en videojuegos y aplicaciones de diseño de modelos tridimensionales. Entre los efectos que es capaz de calcular están:

- **Sombreado constante, de Gouraud y Phong.**
- **Texturas envolventes:** Un solo mapa de bits que envuelve a todo el modelo tridimensional.
- **Bump Mapping:** Se modifica el color de cada pixel en base a un mapa de normales.
- **Normal Mapping:** Se sustituye el color asignado a cada pixel por uno basado en varios mapas de normales mas complejos que el del modelo tridimensional.

  .

Trazado de rayos: Este método es utilizado para generar imágenes fotorealistas. Se basa en diversos principios de la óptica, para lograr simular el comportamiento de la luz en diversos materiales. Para lograr esto se lanza un rayo desde la cámara hacia cada pixel, se verifica la iluminación, reflexión y absorción de la luz de los materiales de los objetos en la escena y la visibilidad de los mismos de manera recurrente hasta lograr la calidad deseada. Es un proceso muy complejo que hace uso de muchos recursos de cómputo.

Radiosidad: Este método realiza el mismo proceso que el trazado de rayos con la diferencia que en este interviene una iluminación global y se toman en cuenta todos los objetos del ambiente tridimensional, obteniendo iluminación indirecta que no es tomada en cuenta en el método de trazado de rayos, logrando una imagen más realista pero que necesita mayor tiempo para su generación.

1.3 Animación

La animación por computadora es un proceso en el que se modifican las propiedades de forma y posición de los modelos que componen una escena dentro de una línea de tiempo dividida en cuadros para simular movimiento, este proceso puede ser aplicado a gráficos 2D o tridimensionales. Existen varios métodos para lograr este proceso en él que se busca interpolar el cambio de forma y/o posición entre un estado inicial y final [FOLEY96].

Animación por cuadros clave: Este método consiste en definir dos cuadros clave, uno inicial y uno final, e interpolar de manera lineal la posición y/o forma de cada elemento de la escena, en cada uno de los cuadros intermedios, para pasar de la poción y/o forma inicial a la final.

Animación por cinemática inversa: Este método consiste en determinar la posición y/o forma inicial y final de un modelo en base al cambio de posición y/o forma de otro modelo de la escena.

Animación por fuerzas físicas: Para calcular la animación de un modelo en base a la simulación de fuerzas físicas como la gravedad, es necesario definir la dirección e intensidad de la fuerza y como el modelo va a cambiar en su posición y/o forma al ser afectado por la fuerza.

Capítulo 2

Interfaz gráfica de usuario

2. Diseño de interfaces gráficas de usuario

Una interfaz gráfica de usuario es un software que a través de imágenes y objetos gráficos que conforman un entorno visual, permite conocer de manera física, perceptiva o conceptual la información y las acciones disponibles, permitiendo así la comunicación entre el usuario y el sistema de manera sencilla e intuitiva.

El objetivo principal de una interfaz gráfica es el permitir la interacción entre un usuario y un sistema de manera sencilla, buscando hacer más eficiente el uso de este al centrarse en satisfacer las necesidades de información tanto de el usuario como de el sistema, por lo cual deben buscar cumplir las siguientes parámetros al diseñar una interfaz: el reducir el tiempo de aprendizaje, diseñar buscando aumentar la productividad, reducir errores de comunicación y lograr el equilibrio entre el atractivo visual y la funcionalidad.

Los elementos de una interfaz gráfica de usuario que representan interacción deben de mostrar las cualidades físicas de dicha acción, siguiendo el principio de *affordance* que se basa en dos conceptos: la visibilidad, que nos dice que un elemento de la interfaz que represente una acción debe ser visible, y la evidencia, que permite que su comprensión sea intuitiva al reflejar de manera evidente la funcionalidad a realizar y como realizarla [LORÉS01].

La interacción entre el usuario y una interfaz gráfica se puede dar de diversas formas y depende de los dispositivos hardware y del software con el cual el usuario interactúa, como la posición del puntero, la selección de elementos, intercambio de texto, cuantificación de elementos, la interacción 3D, etc. como ejemplo de tareas básicas, dando lugar con el uso combinado a taras compuestas como diálogos, construcción y manipulación de elementos, etc.

Actualmente una interfaz gráfica consta de elementos bien definidos a los cuales los usuarios de una computadora están familiarizados a usar como: ventanas, botones, menús, barras de tareas, barras de herramientas, barras de desplazamiento, cajas de texto, casillas de selección y verificación, imágenes, listas, etc.

Para lograr cumplir todos los requerimientos del sistema y del el usuario antes mencionados hay que apoyarse en diversas disciplinas como la psicología cognitiva y social, el diseño gráfico para la iconografía y el uso de color, la sociología, la ergonomía o factores humanos, la ingeniería de software, la inteligencia artificial y la programación.

2.1 Usabilidad

Como ya se mencionó el objetivo de una interfaz gráfica es que sea fácil de usar, útil, eficiente y debe satisfacer los requerimientos tanto del usuario como del sistema para lograr entonces la máxima usabilidad de la interfaz, para que se consiga esto se deben cumplir los siguientes principios:

Facilidad de aprendizaje: el tiempo necesario para aprender a usar la interfaz debe ser mínimo. Para conseguir esto es necesario que la interfaz sea *sintetizable,* es decir, que el usuario tiene que ser capaz de evaluar el efecto de operaciones anteriores en el estado actual, y que sea *familiar* para los usuarios con experiencia previa en el uso de interfaces similares.

Consistencia: todas las metáforas, objetos y comportamientos que se utilizan dentro de una interfaz deben ser usados siempre de la misma manera, siempre que se utilicen y sea cual sea el momento en que se haga.

Flexibilidad: Se debe de proveer a los usuarios de herramientas suficientes para un ágil intercambio de información entre el usuario, la interfaz y el sistema.

Para lograr esto se debe dar control al usuario para facilitar la interacción, para que la interfaz los apoye en las tareas, para que el algoritmo en que se realizan sea obvio, para que el usuario sea capaz de controlar y deshacer todas las operaciones; también se debe permitir la migración de tares, es decir, que tanto el usuario como el sistema o en conjunto pueden realizar tareas, además la interfaz debe de estar preparada para evaluar la información que el usuario y el sistema tienen que recibir. Otro aspecto importante es que la interfaz pueda adaptarse a las necesidades del usuario, su experiencia y el tipo de tareas que realiza con mayor frecuencia.

Robustez: La interfaz debe de ser lo suficientemente completa para poder satisfacer las características de la interacción entre el usuario y el sistema.

Recuperabilidad: La interfaz debe permitir de manera sencilla el volver a realizar una acción en caso de haber cometido un error.

Tiempo de respuesta: Es la capacidad de la interfaz de reflejar el estado actual del sistema en base a las acciones del usuario.

Adecuación de las tareas: Es grado de soporte que la interfaz brinda para todas las operaciones que un usuario necesita realizar y que el sistema permite.

Disminución de la carga cognitiva: los usuarios deben de confiar más en su intuición que en la memorización, por lo que no se deben de memorizar demasiadas cosas para poder usar la interfaz.

2.2 Metodología para el diseño de una interfaz gráfica

Para lograr que una interfaz gráfica consiga cumplir con todos los principios de usabilidad y accesibilidad, como ya se a mencionado, su desarrollo se debe apoyar en diversas disciplinas, en particular la ingeniería de software, que provee de diversas metodologías para desarrollar de manera eficaz cada una de las etapas que comprenden al diseño de una interfaz gráfica, evaluando constantemente el desempeño del la interfaz ante el usuario y el sistema para el cual fue diseñada, esto mediante el análisis de requerimientos, una evaluación de las necesidades y una constante serie de pruebas con el usuario. A continuación se muestra un diagrama que explica el modelo de proceso a seguir.

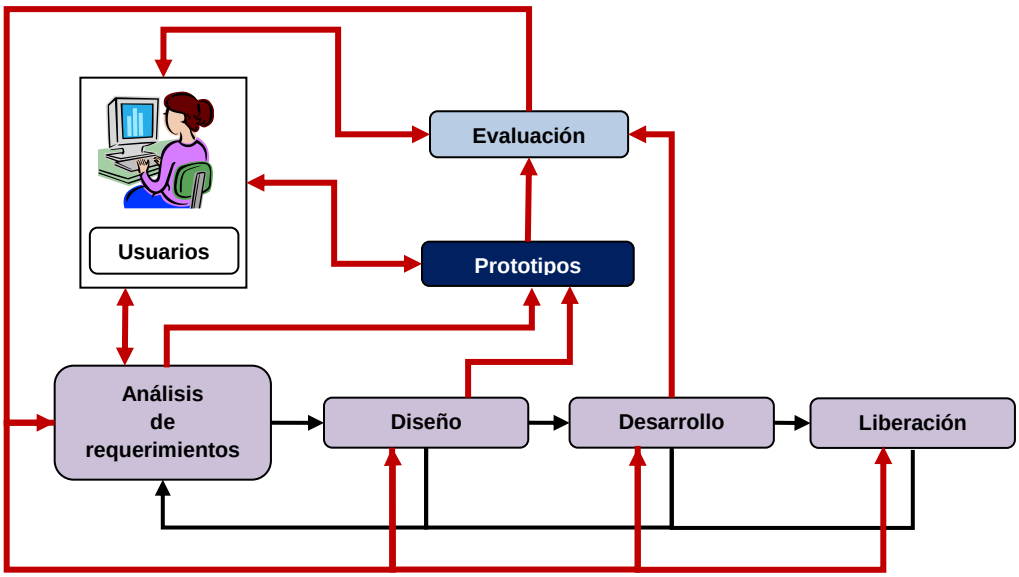


Figura 3: Metodología de la ingeniería de software aplicado al diseño de interfaces gráficas de usuario

Este diagrama explica la metodología de un proyecto de software desde el punto de vista de la ingeniería de software, en este caso aplicado al desarrollo de una interfaz gráfica de usuario, como primer paso el análisis de requerimientos, seguido del diseño, el desarrollo y la liberación, en base al análisis de requerimientos y el diseño se pueden desarrollar prototipos. Como se muestra por el flujo de las flechas negras este proceso puede ser cíclico de acuerdo a la evaluación de cada uno de los resultados obtenidos en cada etapa, a partir de las necesidades del usuario y el sistema, las cuales se evalúan constantemente en cada módulo del proceso diagramado con las flechas rojas.

Análisis de requerimientos: El análisis de requerimientos tiene como meta principal el recabar toda la información necesaria para el diseño de la interfaz gráfica en base a las necesidades y restricciones existentes tanto por parte del usuario como del sistema; además de las características de hardware y software el particular el sistema operativo bajo el cual se va a operar, para que el diseño se desarrolle buscando la mayor usabilidad y accesibilidad posible.

Para llevar a cabo este proceso se deben de seguir las siguientes actividades:

- análisis socio-cultural, para conocer el lenguaje de usuario y su concepto de organización
- Conocer el perfil de los usuarios a los cuales está dirigido
- Analizar los patrones con los que se llevan a cabo los procesos del sistema que ejecuta la interfaz gráfica
- Identificar los tipos de usuario
- Definir que tareas especificas realiza cada tipo de usuario
- Identificar la estructura organizacional entre tipos de usuario y las tareas que realizan
- Definir la manera en que se manejará la información relativa del sistema hacia el usuario y que elementos gráficos son convenientes de usar
- Definir las ventajas y desventajas de la tecnología a utilizar(software y hardware)
- Estructurar claramente los objetivos de la interfaz tanto ante el sistema como hacia el usuario.

Diseño: El que un sistema tenga una adecuada comunicación con el usuario y que satisfaga todas las necesidades de información de ambas partes es necesario que el diseño visual de la interfaz sea lo suficientemente cuidado para lograr un equilibrio entre la estética, la organización y la funcionalidad, ya que de esto depende que el sistema opere adecuadamente y que se brinde un correcto soporte para las operaciones que el usuario realiza a través de este.

Para conseguir de manera eficaz el que el diseño cumpla con las expectativas mencionadas, el uso de la siguiente metodología es de gran utilidad:

- Estructurar la información obtenida en el análisis de requerimientos y de las tareas que realizara el usuario en el sistema
- Elaborar un diseño conceptual de la interfaz, donde se identificaran las secciones que compondrán a la interfaz
- Definir el estilo que tendrá toda la interfaz, en cuanto a estándares generales, las metáforas a usar, los colores y los estándares diseñados para el usuario, en base a esto ya es posible obtener datos suficientes para el diseño de prototipos de interfaz.

Prototipos: La elaboración de prototipos de interfaces gráficas resultante de la fase de diseño y la evaluación de los requerimientos del usuario y del sistema, que se realiza antes de incorporar la funcionalidad en la etapa de desarrollo, resulta

muy útil ya que permite que el usuario pueda evaluar la ergonomía, estética y accesibilidad de los prototipos de interfaz y seleccionar el mas adecuado antes de pasar a la etapa de desarrollo. Ya que el diseño de prototipos solo obedece a cuestiones de diseño gráfico es posible realizarse sin necesidad de pasar por la etapa de análisis de requerimientos si existe experiencia previa en interfaces similares.

Dependiendo de la información con la que se cuente para el desarrollo de los prototipos, y de los recursos de hardware y software bajo los cuales se desarrollara la interfaz y el sistema, se puede hacer uso de diversas herramientas de diseño desde bocetos en papel, plantillas realizadas con algún software de diseño, hasta programar la interfaz sin incorporar funcionalidad; la única condición es que se ejemplifique como será la interfaz para cada uno de los escenarios posibles que puedan existir al momento de que se incorpore al sistema y tenga funcionalidad con datos reales.

Evaluación: La evaluación de todos los procesos en el ciclo de vida del diseño de una interfaz de usuario es trascendental, ya que permite que el desarrollo de la interfaz se centre en cada etapa, en satisfacer las necesidades tanto del sistema como del usuario, además de que da lugar a que los desarrolladores y diseñadores apliquen los estándares respectivos y pueda darse una retroalimentación en cada etapa en base a la información obtenida de la realización de las pruebas pertinentes en cada fase del diseño de una interfaz gráfica de usuario.

Toda esta serie de pruebas se hace buscando que la interfaz cumpla con todos los principios de usabilidad. Para poder cumplir con este objetivo dependiendo de la etapa es necesario que esta evaluación se lleve a cabo por varios usuarios y que antes de la etapa de liberación se realice una evaluación del desempeño de la interfaz tanto ante el usuario como al sistema. Este proceso debe hacerse a través de pruebas con datos reales y pruebas de estrés, es decir, pruebas que determinen si la interfaz responderá adecuadamente ante escenarios críticos en los que se especula que podría no funcionar correctamente.

Desarrollo: En la etapa de desarrollo, en función de toda la información recopilada, ya se hace una elección definitiva de la o las tecnologías que se van a emplear para el desarrollo de la interfaz en base a la tecnología en la cual esta construido el sistema y de los recursos de software y hardware bajo los cuales operara la interfaz en conjunto con el sistema.

En esta etapa se desarrolla la interfaz gráfica definitiva por lo cual no se debe descuidar todo el proceso previo realizado que se centro en buscar la mayor usabilidad y accesibilidad posible centrándose en las necesidades del usuario y de el sistema para el cual se esta desarrollando la interfaz.

Liberación: En esta fase del diseño de la interfaz gráfica de usuario es donde culmina toda la metodología descrita para obtener un producto final y que se

espera que cumpla con los requerimientos y con los principios de usabilidad, ya que como se enfatizó a lo largo de todas las fases, la evaluación constante por parte del usuario es trascendental debido a que de esta manera todo el proceso se basó en las necesidades y gustos del el usuario. Este proceso debe hacerse sin descuidar que deben cubrir los requerimientos del sistema para la cual fue diseñada, por lo que al mostrar al usuario el resultado final, debe satisfacerlo, resultarle familiar y muy intuitivo al empezar a usarlo ya que este fue participe directa e indirectamente en todas las fases el desarrollo.

Esta retroalimentación por parte del usuario no debe de terminar con la liberación de la interfaz en conjunto con el sistema para el cual fue desarrollada, ya que esto permitirá que los diseñadores y desarrolladores puedan mejorar la interfaz en un futuro dependiendo de su desempeño con el sistema al trabajar en un escenario real, además de que las necesidades pueden cambiar, o sea necesario implementar nuevas funcionalidades o bien que esta interfaz sirva como base para posteriores versiones del sistema.

Capítulo 3

Realidad virtual

3. Entornos Virtuales

El concepto de realidad virtual se comenzó a formar a partir del estudio de la cibernética, que puede definirse como el análisis de cómo organismos biológicos, digitales o mecánicos procesan y reaccionan ante la información generando procesos de comunicación y control que logran una mejor asimilación y procesamiento de la información [LEIVA06].

En base a esto la realidad virtual puede definirse como un ambiente tridimensional e interactivo generado a partir de recursos de computo (hardware y software) que tiene como finalidad estimular todos los sentidos de un usuario, logrando simular toda la información sensorial que constituye un ambiente real, es decir, es un ambiente que aparenta ser real pero no lo es.

La relación existente entre un ambiente real y un ambiente virtual puede medirse en base a los elementos que constituyen a cada uno de estos. Un ambiente real con algunos elementos virtuales es considerado como una realidad aumentada, mientras que un ambiente virtual que contiene algunos elementos reales es considerada como una virtualidad aumentada, siendo lo real y lo virtual los dos extremos de esta relación.

Entre las características que se le pueden atribuir a un ambiente virtual está el que es interactivo, es decir, existe un intercambio de información entre el usuario y el ambiente; se desarrolla en tiempo real, la física en el ambiente puede ser controlada, puede existir una lógica o perspectiva diferente del ambiente y todos los aspectos del ambiente pueden ser modificados.

Un sistema de realidad virtual está conformado por cinco elementos, los dispositivos de interacción con la interfaz, los sistemas de posicionamiento, los sistemas de visualización, los recursos físicos de computo (hardware) y el software que determina el comportamiento de cada uno de los elementos mencionados.

Entre las aplicaciones de los ambientes virtuales están las científicas que simulan procesos reales; y las de entretenimiento como los videojuegos, las de telepresencia como medio de comunicación, y las de teleoperación para la realización de procesos reales a distancia.

Un ambiente virtual puede clasificarse como inmersivo y no inmersivo. Los ambientes inmersivos se caracterizan por que los elementos que los conforman y su comportamiento, es decir su física es muy precisa al simular la realidad, lo que

engaña al usuario y logra sumergir mas a este en el ambiente, además de que hace uso de diversos sensores para detectar las respuestas del usuario y responder ante ellas, mientras que un ambiente no inmercivo se muestra sólo en un sistema de proyección y hace uso de recursos multimedia.

3.1 Modelos cognitivos

Para poder lograr una mejor interacción de usuario con el sistema es importante estudiar los modelos cognitivos de este, es decir, comprender como el usuario procesa la información sensorial que percibe del mundo virtual que se le despliega, como almacena y recupera la información y como responde ante esta para determinar si el sistema es intuitivo o no.

Un modelo cognitivo está fuertemente relacionado por cómo es afectado el sistema sensorial del usuario, ya que de esto depende que grado de la información que el sistema envía al usuario es percibida, si le es fácil memorizarla a corto o largo plazo y usarla en una tarea específica donde se tengan que tomar decisiones. Básicamente la interacción está compuesta de los siguientes pasos: establecer un objetivo, definir una intención, elaborar un algoritmo, ejecutar dicho algoritmo, determinar el estado del sistema, analizar dicho estado y evaluar los cambios percibidos respecto a las acciones que se llevaron a cabo.

Algunos factores del sistema que se deben prever para mejorar la interacción son la conducta del usuario, el conocimiento previo de la interfaz, la representación de la información y el modelo de aprendizaje.

3.2 Sistema sensorial

La interacción es posible cuando existe un intercambio de información entre el usuario y el sistema, para que este proceso se lleve a cabo de manera satisfactoria es esencial poder transmitir y percibir de manera correcta la información que se intercambia. Para logar dicho fin se debe estimular de manera adecuada el sistema sensorial del usuario el cual está compuesto del sistema visual, el auditivo, el táctil, el cenestésico, el vesicular y el olfativo.

El sistema visual es uno de los más investigados y usados y para el cual se desarrollan un mayor número de dispositivos. Este sistema es muy importante ya que a través de la información que los ojos perciben el usuario conoce los elementos del entorno que lo rodea. Entre los factores que se manipulan para afectar este sistema están el color y la iluminación.

El sistema auditivo también es estimulado frecuentemente, mediante este se envía gran parte de la información ya sean sonidos que representan sucesos u órdenes

orales al usuario, que se pueden combinar con los elementos visuales. Para manipular este sentido se controlan los aspectos físicos de las ondas sonoras.

El sistema táctil también es importante ya que por este medio el usuario es capaz de conocer las propiedades físicas de los elementos que componen un entorno virtual, entre los elementos que se busca estimular de este sistema es la capacidad de reconocer texturas y variaciones en la temperatura.

El sistema cenestésico probé al sistema de realidad virtual, información sobre los movimientos del usuario, mientras que al manipular el sistema vestibular podemos proveer al usuario información sobre el movimiento, orientación y aceleración.

3.3 Dispositivos de interacción

Los dispositivos de interacción de un entorno virtual pretenden estimular el sistema sensorial del usuario y obtener información de éste. Entre la información que el sistema necesita conocer del el usuario esta la posición de este y los movimientos que realiza, o los sonidos que produce en el caso de la comunicación oral; y entre la información que el sistema puede proporcionar al usuario están la sensación de movimiento, el despliegue de imágenes, la reproducción de sonidos entre ellos palabras y emisión de olores.

Para identificar la posición y movimiento de un usuario, y poderle además transmitir la sensación de movimiento, los dispositivos de interacción y posicionamiento toman en cuenta tres referencias conocidas como “Pitch”, “Yaw” y “Roll” que corresponden al desplazamiento angular sobre los ejes de un sistema cartesiano de tres dimensiones, es decir el eje X, Y y Z respectivamente; además del desplazamiento lineal sobre cada uno de estos.

Para que el sistema provea sonidos de manera realista se necesita que el sistema de audio del entorno virtual sea capaz de transmitir un sonido envolvente, que es como se transmiten los sonidos en un ambiente real. Con esto el usuario puede interpretar el origen de este, y además el sistema debe ser capaz de reconocer de manera correcta los sonidos emitidos por el usuario para así poder interpretarlos y reaccionar ante ellos.

Para transmitir al usuario los elementos visuales que conforman el entorno virtual es necesario que estos dispositivos sean capaces de reproducir imágenes de alta calidad y más importante aún es que lo puedan hacer en tiempo real.

Capítulo 4

Colisiones en ambientes tridimensionales

4. Volúmenes de colisión

Para poder comprobar si existe una colisión entre dos o más modelos tridimensionales y que este proceso se lo más sencillo posible para evitar el uso excesivo de recursos de computo, se usan volúmenes de colisión. Los volúmenes de colisión, son modelos tridimensionales más simples como cajas, cilindros, esferas, conos, etc. que envuelven al modelo tridimensional para que la colisión se compruebe en base a estos modelos, y no en base a los modelos tridimensionales, ya que estos generalmente están compuestos por un gran número de polígonos comúnmente triángulos y si el cálculo de la colisión se realiza en base a los polígonos que conforman al modelo tridimensional se consumen demasiados recursos de computo.

El que un volumen de colisión sea efectivo depende de que tan cercano esté a la forma del modelo tridimensional, además de que el poder obtener el volumen de colisión y determinar si existe una colisión entre dos o más volúmenes de colisión, consuma el menor número de recursos de cómputo. También debe ser tomando en cuenta, que el modelo tridimensional puede cambiar tanto de posición como de orientación con respecto a un sistema de ejes coordenados XYZ.

Entre los volúmenes de colisión más usados están la caja envolvente alineada a los ejes, la esfera y el cilindro; a partir de estos se han diseñado otros volúmenes de colisión que pretenden ajustarse mejor al modelo tridimensional y tener un mayor grado de realismo como la cápsula, la caja envolvente orientada al modelo, el elipsoide o el cono pero el obtener dichos volúmenes de colisión y el determinar la colisión entre estos resulta muy complejo ocasionando un uso excesivo de recursos de cómputo [JIMÉNEZ06].

4.1 Caja envolvente orientada a los ejes

Este volumen de colisión es una caja rectangular en donde las caras de esta son paralelas con los ejes coordenados, tanto la obtención del volumen de colisión, como la comprobación de colisión entre dos cajas resulta muy eficiente ya que consume muy pocos recursos de computo y en general se ajusta a la geometría de los modelos tridimensionales. La desventaja de este volumen de colisión es que si el modelo tridimensional debe rotar, el cálculo de las colisiones resulta en un uso excesivo de recursos de cómputo.

Para calcular este volumen de colisión en base al conjunto de vértices de un modelo tridimensional se usa el siguiente algoritmo:

```
mObj Mcol(mObj obj){
     int i;
     obj.xm = obj.v_obj[1].x;
     obj.ym = obj.v_obj[1].y;
     obj.zm = obj.v_obj[1].z;
     obj.xM = obj.v_obj[1].x;
     obj.yM = obj.v_obj[1].y;
     obj.zM = obj.v_obj[1].z;
     for(i=1;i<obj.nVer;i++){
          if(obj.xm>obj.v_obj[i].x)
               obj.xm = obj.v_obj[i].x;
          else{
               if(obj.xM<obj.v_obj[i].x)
                    obj.xM = obj.v_obj[i].x;
          }
          if(obj.ym>obj.v_obj[i].y)
               obj.ym = obj.v_obj[i].y;
          else{
               if(obj.yM<obj.v_obj[i].y)
                    obj.yM = obj.v_obj[i].y;
          }
          if(obj.zm>obj.v_obj[i].z)
               obj.zm = obj.v_obj[i].z;
          else{
               if(obj.zM<obj.v_obj[i].z)
                    obj.zM = obj.v_obj[i].z;
          }
     }

     return obj;
}
```

De este algoritmo obtenemos los valores de los puntos máximos y mínimos sobre cada eje coordenado, es decir, sobre e eje X, Y y Z. Estos son los datos necesarios para poder determinar los ocho vértices de la caja envolvente orientada a los ejes.

v1(xm,ym,zm)

v2(xm,ym,zM)

v3(xm,yM,zm)

v4(xm,yM,zM)

v5(xM,ym,zm)

v6(xM,ym,zM)

v7(xM,yM,zm)

v8(xM,yM,yM)

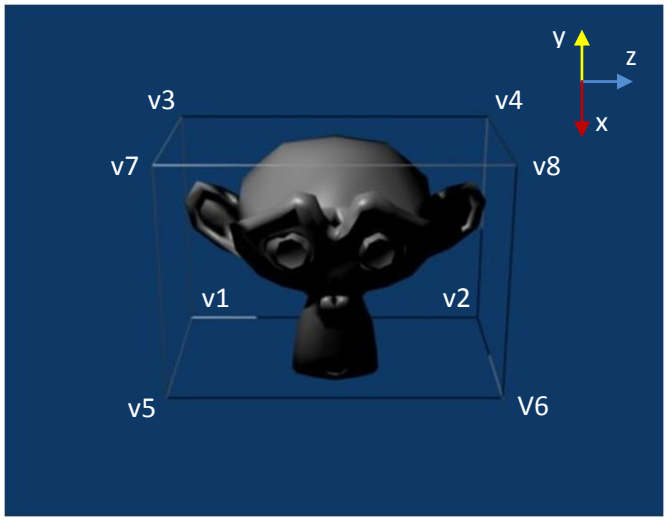


Figura 4: Caja envolvente orientada a los ejes

4.2 Esfera envolvente

La esfera envolvente, es un volumen de colisión que es invariante ante los cambios de posición y orientación al realizar un test de colisión, además de que este test es bastante sencillo y usa pocos recursos de cómputo. Una desventaja de este volumen de colisión es que difícilmente se adecua a la geometría de un modelo tridimensional además de que el cálculo de los datos de la esfera, el centro y el radio resulta un algoritmo complejo más aun si el modelo tridimensional no se encuentra en el origen de coordenadas como suele ser el caso.

Para calcular este volumen de colisión en base al conjunto de vértices de un modelo tridimensional se usa el siguiente algoritmo:

```
mObj Mcol(mObj obj){
     int i;
     float maxX,maxY,maxZ;
     maxX = 0;
     maxY = 0;
     maxZ = 0;
     obj.xm = obj.v_obj[1].x;
     obj.ym = obj.v_obj[1].y;
     obj.zm = obj.v_obj[1].z;
     obj.xM = obj.v_obj[1].x;
     obj.yM = obj.v_obj[1].y;
     obj.zM = obj.v_obj[1].z;
     for(i=1;i<obj.nVer;i++){
          if(obj.xm>obj.v_obj[i].x)
               obj.xm = obj.v_obj[i].x;
          else{
               if(obj.xM<obj.v_obj[i].x)
                    obj.xM = obj.v_obj[i].x;
          }
          if(obj.ym>obj.v_obj[i].y)
               obj.ym = obj.v_obj[i].y;
          else{
               if(obj.yM<obj.v_obj[i].y)
                    obj.yM = obj.v_obj[i].y;
          }
          if(obj.zm>obj.v_obj[i].z)
               obj.zm = obj.v_obj[i].z;
          else{
               if(obj.zM<obj.v_obj[i].z)
                    obj.zM = obj.v_obj[i].z;
          }
     }

     obj.cx=(obj.xM+obj.xm)/2;
     obj.cy=(obj.yM+obj.ym)/2;
     obj.cz=(obj.zM+obj.zm)/2;

     maxX=(abs(obj.xM)+abs(obj.xm))/2.0;
     maxY=(abs(obj.yM)+abs(obj.ym))/2.0;
     maxZ=(abs(obj.zM)+abs(obj.zm))/2.0;
     obj.rdio = sqrt(maxX*maxX + maxY*maxY + maxZ*maxZ);

     return obj;
}
```

De este algoritmo obtenemos las coordenadas del centro y la longitud del radio de la esfera envolvente:

c(cx,cy,cz)

r = rdio

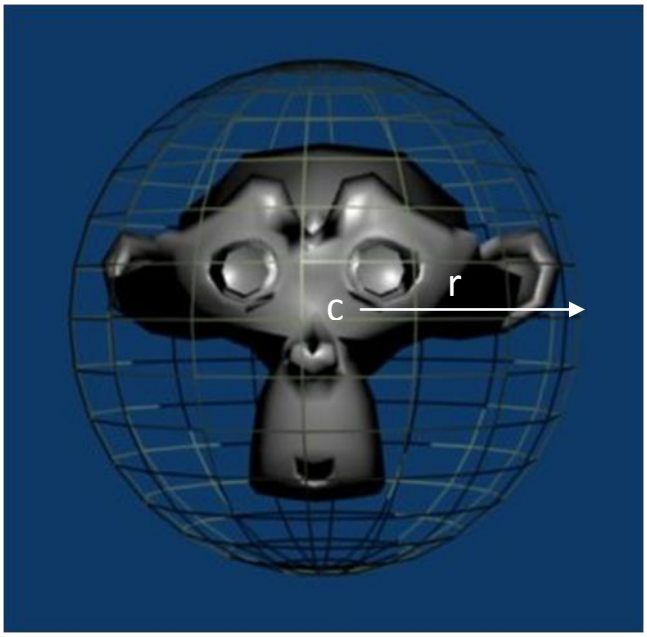


Figura 5: Esfera envolvente

4.3 Cilindro envolvente

En el caso del cilindro envolvente es un volumen de colisión que tiene la ventaja de que envuelve con mayor precisión a los modelos tridimensionales, además de que el detectar la colisión es sencillo mientras el modelo sólo se desplace sobre los ejes o gire sobre el eje Y. El cálculo del cilindro resulta más complejo que la caja o la esfera ya que para determinar sus dimensiones es necesario conocer el centro, el radio y la altura.

Para calcular este volumen de colisión en base al conjunto de vértices de un modelo tridimensional se usa el siguiente algoritmo:

```
mObj Mcol(mObj obj){
     int i;
     float maxX,maxY,maxZ;
     maxX = 0;
     maxY = 0;
     maxZ = 0;
     obj.xm = obj.v_obj[1].x;
     obj.ym = obj.v_obj[1].y;
     obj.zm = obj.v_obj[1].z;
     obj.xM = obj.v_obj[1].x;
     obj.yM = obj.v_obj[1].y;
     obj.zM = obj.v_obj[1].z;
     for(i=1;i<obj.nVer;i++){
          if(obj.xm>obj.v_obj[i].x)
               obj.xm = obj.v_obj[i].x;
          else{
               if(obj.xM<obj.v_obj[i].x)
                    obj.xM = obj.v_obj[i].x;
          }
          if(obj.ym>obj.v_obj[i].y)
               obj.ym = obj.v_obj[i].y;
          else{
               if(obj.yM<obj.v_obj[i].y)
                    obj.yM = obj.v_obj[i].y;
          }
          if(obj.zm>obj.v_obj[i].z)
               obj.zm = obj.v_obj[i].z;
          else{
               if(obj.zM<obj.v_obj[i].z)
                    obj.zM = obj.v_obj[i].z;
          }
     }

     obj.cx=(obj.xM+obj.xm)/2;
     obj.cy=(obj.yM+obj.ym)/2;
     obj.cz=(obj.zM+obj.zm)/2;

     maxX=(abs(obj.xM)+abs(obj.xm))/2.0;
     maxY=(abs(obj.yM)+abs(obj.ym))/2.0;
     maxZ=(abs(obj.zM)+abs(obj.zm))/2.0;
     obj.rdio = sqrt(maxX*maxX + maxZ*maxZ);
     obj.altra = maxY;

     return obj;
}
```

De este algoritmo obtenemos las coordenadas del centro, y la longitud del radio y altura del cilindro envolvente:

c(cx,cy,cz)

r = rdio

h = altra

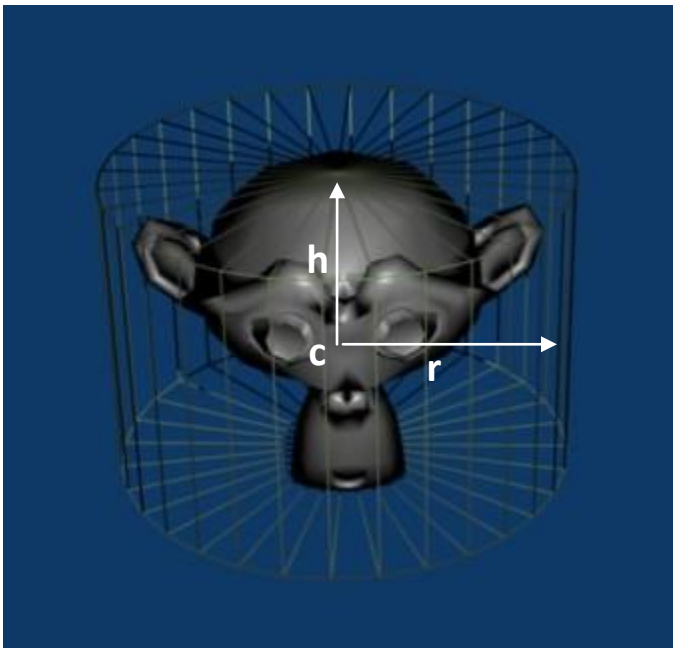


Figura 6: Cilindro envolvente

4.4 Algoritmos de colisión

Una colisión entre modelos tridimensionales puede definirse como un evento en un entorno tridimensional en donde dos o más modelos ocupan el mismo espacio en el mismo momento. Una colisión sólo puede presentarse entre un modelo que puede cambiar sus propiedades de posición ya sea rotación o traslación, y el entorno tridimensional u otros modelos con esta característica.

Ya que el despliegue de un entorno interactivo se hace frame a frame, es necesario que la detección de la colisión se realice también frame a frame, por esta razón es necesario que el algoritmo usado para detectar la colisión entre modelos tridimensionales consuma pocos recursos de computo, ya que de no ser así afectará el tiempo en que los frames que despliegan, lo que gráficamente representa que los movimientos que ocurran en la escena tridimensional sean lentos. En general para que un movimiento sea fluido es necesario que se desplieguen más de 16 frames por segundo (fps).

Una vez que una colisión ha ocurrido un entorno virtual debe ser capaz de responder a dicho evento. En general cuando este evento ocurre el o los modelos tridimensionales son cambiados de posición y orientación hasta que ya no se presente la colisión. Dependiendo del algoritmo usado para resolver la colisión puede llegar a ser necesario el determinar el punto o sección donde se a presentado.

Como se menciono anteriormente para reducir el uso de recursos de cómputo al calcular una colisión se usan volúmenes de colisión, sobre los cuales se aplican los algoritmos para calcular la colisión entre modelos tridimensionales. Por esta razón los algoritmos solo hacen uso de la información proporcionada por los volúmenes de colisión, es decir sus propiedades de ubicación, y dimensiones.

4.4.1 Detección de colisión entre cajas envolventes alineadas a los ejes

De este volumen de colisión se conocen los ocho vértices que lo conforman; en base a estos se puede determinar si dos cajas envolventes alineadas a los ejes comparten el mismo espacio a través del teorema de los ejes separados, que simplifica el determinar si existe una colisión mediante la proyección de los vértices de la caja sobre cada eje, si en la proyección sobre los tres ejes existe una colisión entonces existe la colisión en el entorno tridimensional. Esta proyección corresponde a la posición máxima (M) y mínima (m) sobre cada eje cartesiano.

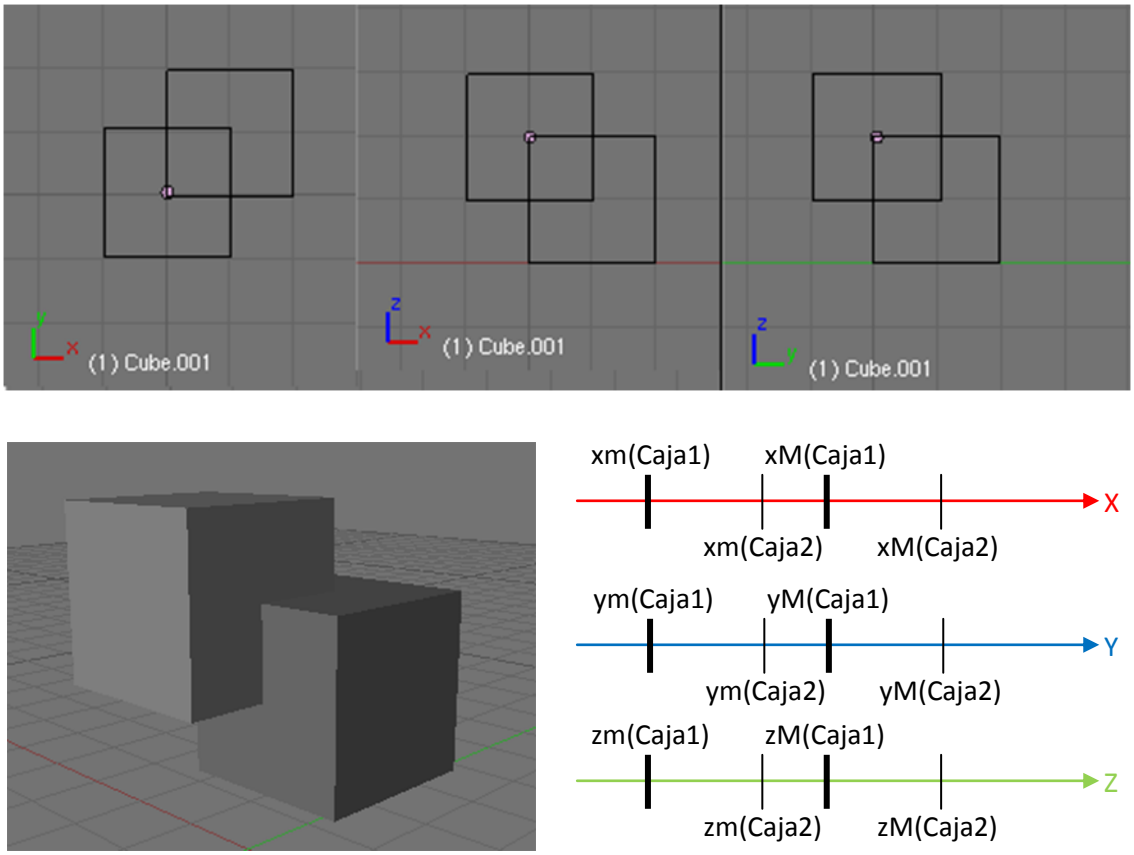
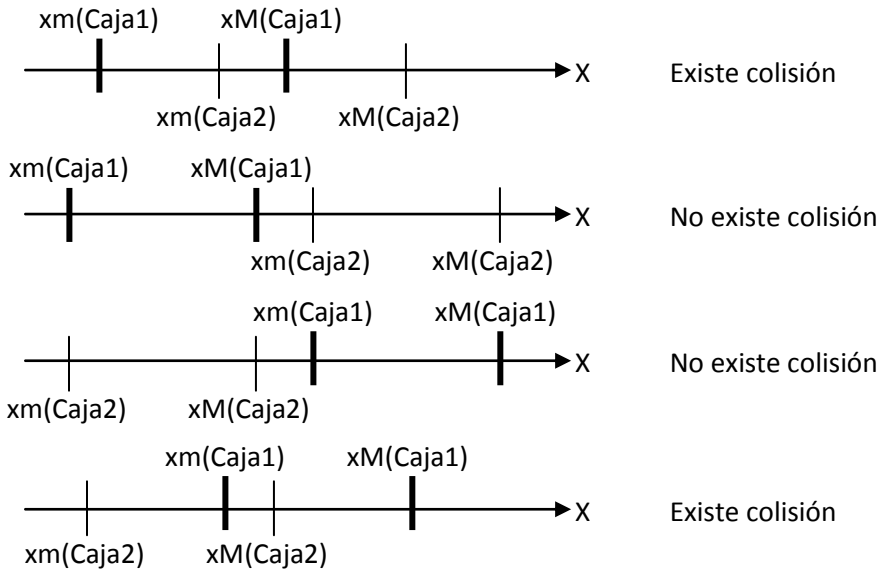


Figura 7: Análisis para la detección de colisión entre cajas envolventes alineadas a los ejes

A partir de este análisis sobre cada eje se pueden presentar cuatro casos diferentes:



En base al análisis el algoritmo para detectar la colisión entre dos cajas envolventes alineadas a los ejes es el siguiente:

```
bool TestColAABB(vC a,vC b){
     if(((a.xM>=b.xm)&&(b.xM>=a.xM))||((a.xm<=b.xM)&&(b.xM<=a
.xM))){

     if(((a.yM>=b.ym)&&(b.yM>=a.yM))||((a.ym<=b.yM)&&(b.yM<=a
.yM))){

     if(((a.zM>=b.zm)&&(b.zM>=a.zM))||((a.zm<=b.zM)&&(b.zM<=a
.zM))){
                    return true;
               }
               else
                    return false;
          }
          else
               return false;
     }
     else
          return false;
}
```

4.4.2 Detección de colisión entre esferas envolventes

Los datos conocidos de una esfera envolvente son el centro y el radio; a través de estos datos es posible determinar si existe una colisión entre dos esferas envolventes. Para determinar esta colisión se determina si la distancia entre los centros de cada esfera es menor a la suma de los radios de cada esfera. Si esto ocurre la colisión existe. Para hacer este proceso más simple las dimensiones mencionadas se manejan al cuadrado ya que el método para determinar la distancia entre los centros da este resultado al cuadrado y para poder determinar la longitud real es necesario obtenerla raíz cuadrada de este dato lo que es un cálculo complejo que usa muchos recursos de cómputo.

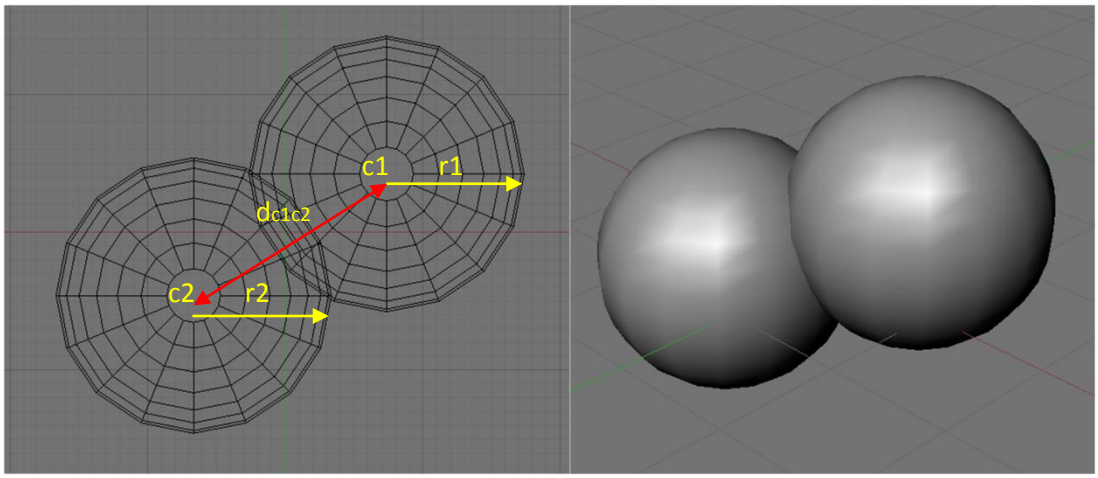


Figura 8: Análisis para detectar la colisión entre esferas envolventes

El algoritmo para llevar a cabo el proceso mencionado y determinar si existe una colisión entre dos esferas envolventes es el siguiente:

```
bool TestColBS(vC a, vC b){
     vC aux;
     float aux2,aux3;
     aux.cx=a.cx-b.cx;
     aux.cy=a.cy-b.cy;
     aux.cz=a.cz-b.cz;
     aux2=(aux.cx*aux.cx)+(aux.cy*aux.cy)+(aux.cz*aux.cz);
     aux3=a.rdio+b.rdio;
     if(aux2<=(aux3*aux3)){
          return true;
     }else{
          return false;
     }
}
```

4.4.3 Detección de colisión entre cilindros envolventes

Para el caso del cilindro envolvente los datos que lo definen son el centro, el radio y su altura; en base a estos el determinar si dos cilindros comparten el mismo espacio es detectada cuando la distancia entre los centros proyectados sobre el eje XZ es menor a la suma de los radios de cada cilindro; si este evento ocurre entonces se verifica si la distancia entre los centros es menor que la suma de las alturas de ambos cilindros; si esto ocurre entonces se presenta la colisión. Al igual que en el caso de las esferas envolventes, estas magnitudes son manejadas al cuadrado para evitar el realizar raíces cuadradas para obtener las longitudes absolutas.

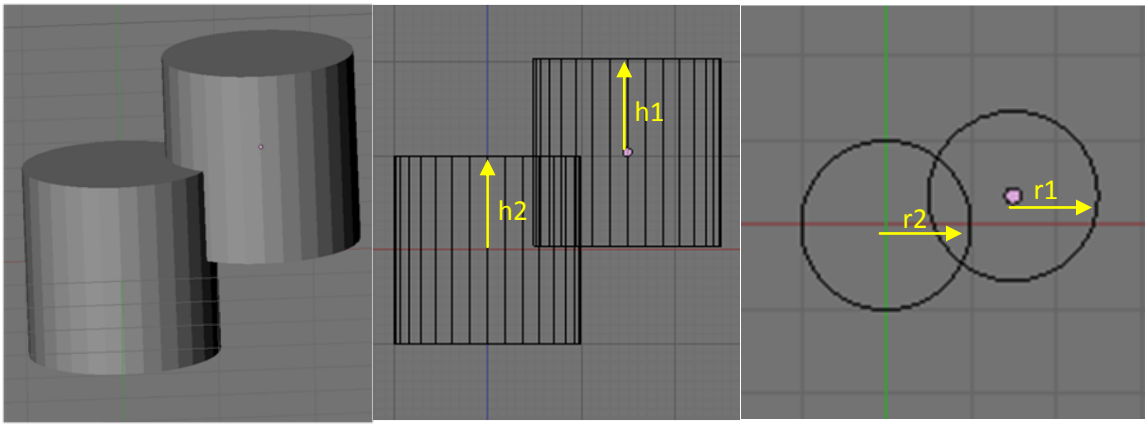


Figura 9: Análisis para detectar la colisión entre cilindros envolventes

El algoritmo para llevar a cabo el proceso mencionado y determinar si existe una colisión entre dos cilindros envolventes es el siguiente:

```
bool TestColBS(vC a, vC b){
     vC aux;
     float aux2,aux3;
     aux.cx=a.cx-b.cx;
     aux.cz=a.cz-b.cz;
     aux2=(aux.cx*aux.cx)+(aux.cz*aux.cz);
     aux3=a.rdio+b.rdio;
     if(aux2<=(aux3*aux3)){
          aux.cy=a.cy-b.cy;
          if(aux.cy<(a.hltra+b.hltra){
               return true;
          }
          else{
               return false;
          }
     }else{
          return false;
     }
}
```

4.5 Modelos tridimensionales

Un modelo tridimensional en un ambiente virtual es un objeto construido a partir de un conjunto de polígonos que están construidos a través de la unión de tres o más vértices; los vértices son una localización en el ambiente tridimensional, es decir, que son puntos en el espacio de coordenadas XYZ. Además de los vértices y la definición de los polígonos a través de estos, un modelo tridimensional tiene más propiedades que determinan características como el color, vértices normales de cada polígono, el material y la textura.

Un software de modelado tridimensional guarda toda esta información en uno o varios archivos que puede leer posteriormente para ser modificados. En general un archivo que guarda información sobre un modelo tridimensional tiene la siguiente estructura:

Identificador del archivo	
Conjunto de vértices (V1,…,Vn)	Cada vértice tiene su componente XYZ
Conjunto de polígonos (P1,…,Pn)	Cada polígono está definido por n vértices, y puede tener asociado uno o más vértices normales, y de textura. Este conjunto puede estar dividido en grupos a los cuales se les puede asignar un color, un material, una textura o un shader.
Vertices normales (VN1,…,VNn)	Un vértice normal está asociado a un vértice en especial
Coordenada de textura (VT1,…VTn)	Una coordenada de textura tiene componente UV sobre una imagen.
Materiales	Un material contiene datos sobre el modelo de iluminación, el color, shaders y la textura.

Las características de un archivo que guarda información sobre un modelo tridimensional depende del software con el que fue creado, el propósito con el cual va a ser usado, el medio donde va a ser almacenado y hasta el tipo de usuario que lo va a usar, ya que puede ser un archivo binario o de texto y puede contener información sobre sólo el modelo o características adicionales para el software con el que se está creando o para la aplicación en la que va a ser usado, o puede contener una animación de el modelo etc.

Capítulo 5

Sistema para aplicar volúmenes de colisión a modelos tridimensionales

5. Problema a resolver

Para poder calcular colisiones entre modelos tridimensionales de tal manera que no se requieran muchos recursos de cómputo, como el calcular la colisión entre los polígonos que conforman al modelo tridimensional, se usan volúmenes de colisión. El principal problema con este método es que difícilmente existe un volumen de colisión que se ajuste de manera adecuada al modelo tridimensional y más aun si se requiere que la existencia de una colisión se pueda calcular a un nivel de detalle mayor o en un área en particular del modelo tridimensional. Debido a esta problemática se han diseñado volúmenes de colisión que buscan ser más próximos a la geometría del modelo tridimensional, pero en general estos modelos son difíciles de calcular al igual que el determinar si existe una colisión entre uno o más modelos, proceso que puede llegar a requerir muchos recursos de cómputo, más aún si se involucran diferentes tipos de volúmenes de colisión en el entorno tridimensional.

Por ésta razón la solución propuesta para este problema es el diseñar una aplicación que permita añadir al usuario volúmenes de colisión a áreas específicas del modelo tridimensional, dependiendo de las necesidades de la aplicación que va a hacer uso de este. Con esto el nivel de aproximación a la geometría del modelo tridimensional dependerá de el número de volúmenes de colisión que se añadan, logrando aproximarse a la respuesta que obtendríamos con el cálculo de colisión entre los polígonos que componen al modelo, pero con un menor uso de recursos de cómputo, o si se necesita que sólo en un área en particular del modelo tridimensional se pueda determinar la existencia de colisión con mayor apego a la geometría del modelo tridimensional.

Partiendo de esta solución, los volúmenes de colisión que se pueden añadir al modelo tridimensional pueden ser modelos sencillos de calcular al igual que el calcular la existencia de colisión entre estos, como es en caso de la esfera envolvente, la caja envolvente alineada a los ejes o el cilindro envolvente.

5.1 Diseño de la interfaz gráfica de usuario

En base a la teoría de diseño de interfaces gráficas de usuario mencionada en capítulos anteriores, se partió del análisis de los requerimientos del sistema, que para este caso debe guiarse en los estándares tanto de diseño como de usabilidad que son utilizados por las aplicaciones de diseño de modelos tridimensionales.

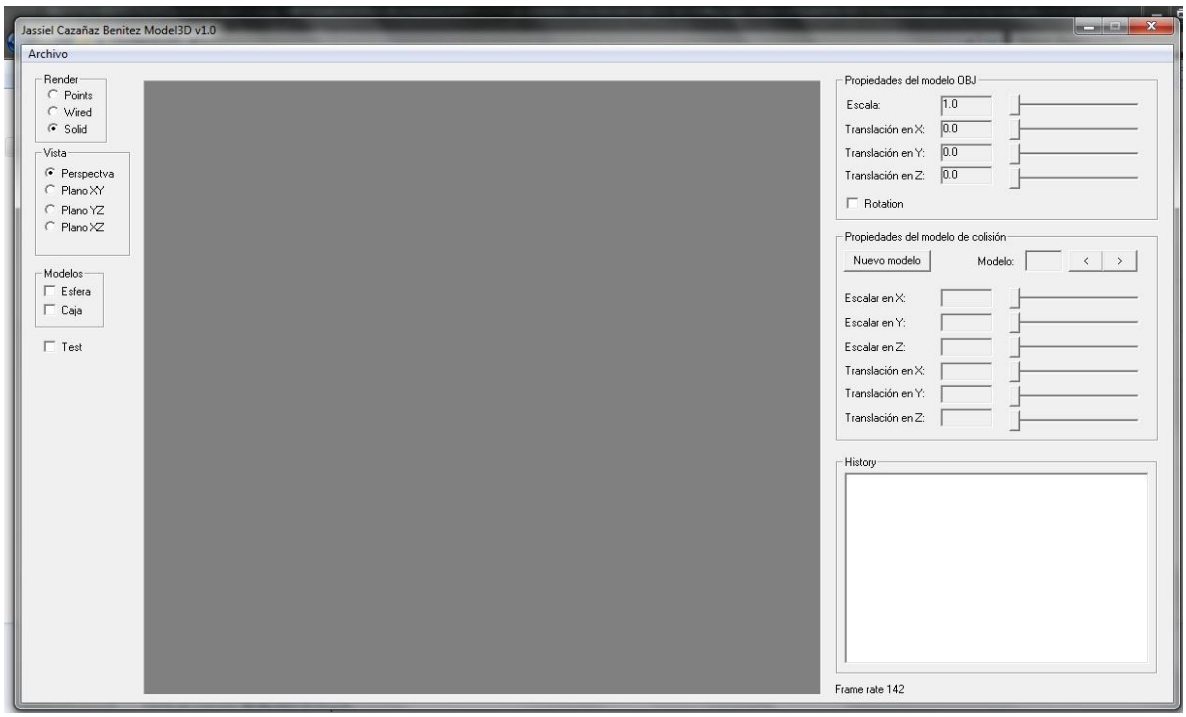


Figura 10: Interfaz grafica de la aplicación

Ya que se va a trabajar con modelos tridimensionales, uno de los primeros requerimientos es que la interfaz muestre, además de la vista en perspectiva del modelo, sus proyecciones sobre cada plano del entorno tridimensional, es decir, la proyección sobre los planos XY, YZ y ZX denominados vista frontal, lateral y superior respectivamente siendo este un estándar entre las aplicaciones de diseño de modelos tridimensionales; por lo tanto la interfaz debe permitir al usuario cambiar entre las vistas para facilitar el trabajo con el modelo tridimensional.

Otro de los requerimientos para la interfaz surge de la necesidad de manipular al modelo tridimensional, es decir, poder desplazar a este a lo largo de los ejes coordenados X, Y y Z o realizar un acercamiento a alguna sección en particular de el modelo tridimensional. Esto es necesario ya que facilita el agregar volúmenes de colisión a zonas especificas del modelo tridimensional. Esta manipulación requiere de un alto grado de precisión, por dicha razón debe de hacerse a través de controles gráficos, ya que estos proveen un mayor grado de control, a diferencia de manipulación a través del mouse, como se realiza en la mayoría de aplicaciones de diseño de modelos tridimensionales.

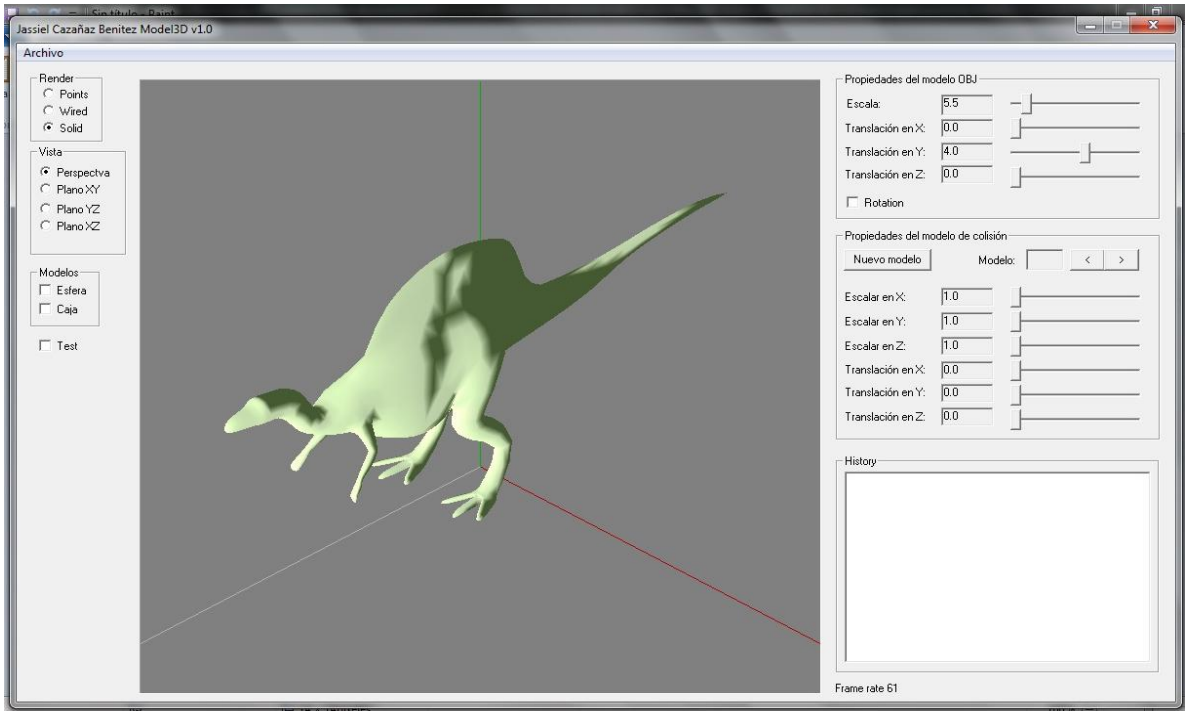


Figura 11: Visualización de la perspectiva de un modelo tridimensional

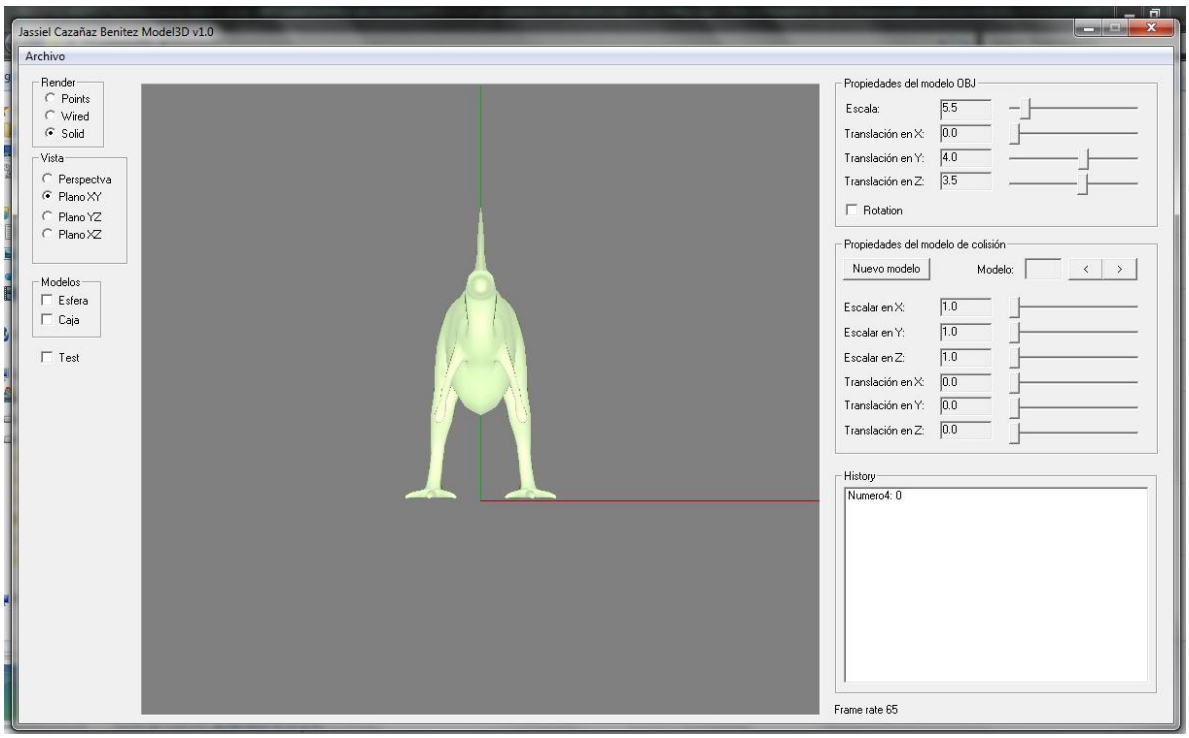


Figura 12: Visualización de frontal de un modelo tridimensional

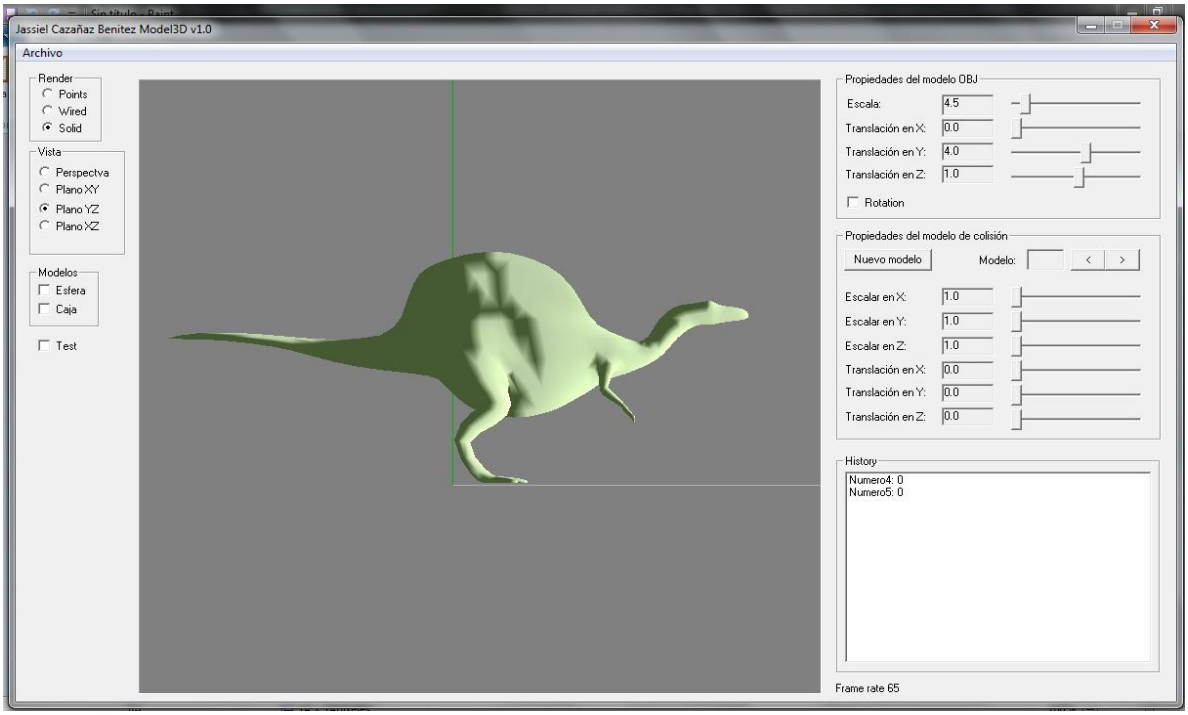


Figura 13: Visualización lateral de un modelo tridimensional

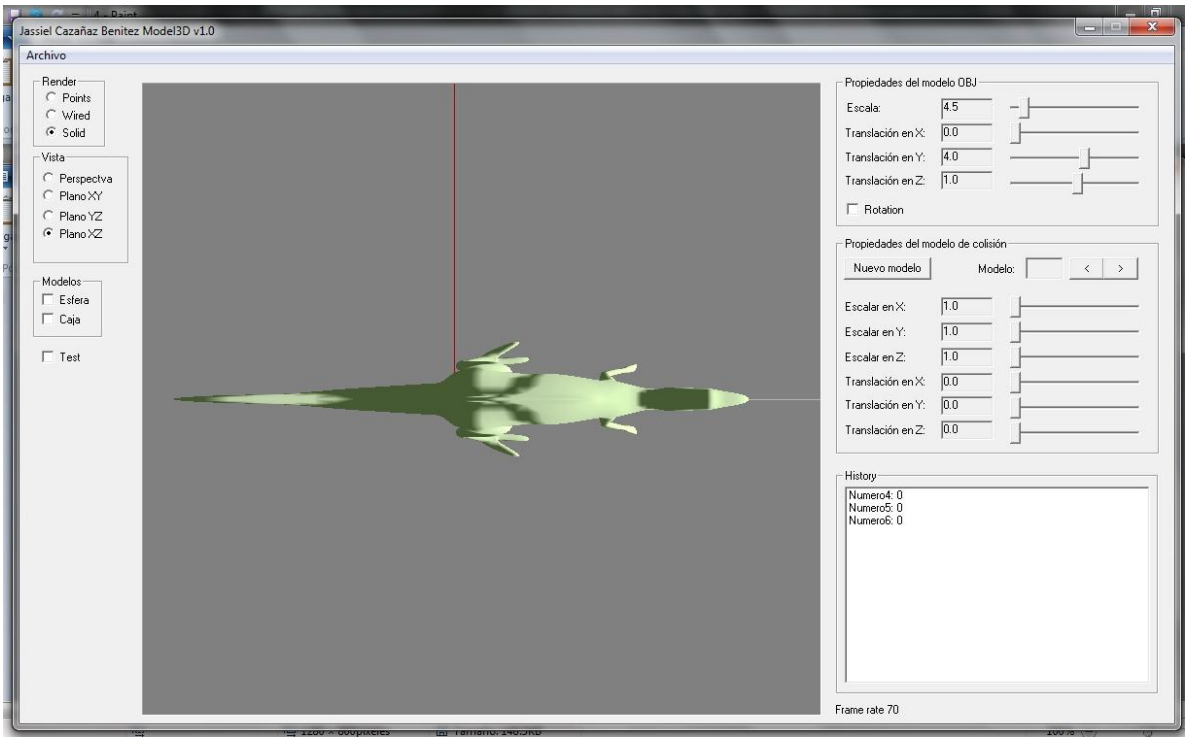


Figura 14: Visualización de superior de un modelo tridimensional

Como siguiente requerimiento, es necesario que la interfaz permita al usuario observar el modelo tridimensional en los diferentes modos de dibujo estándar como lo son el modo solido, el modo de alambre y el modo de vértices, ya que esto permite al usuario visualizar de mejor manera al modelo tridimensional y como está construido en base a polígonos. El uso de estos diferentes modos de dibujo, facilita el trabajo cuando se muestra el modelo tridimensional en conjunto con los volúmenes de colisión, ya sean los que la aplicación calculara en base al modelo tridimensional o los que el usuario decida añadir según sus necesidades.

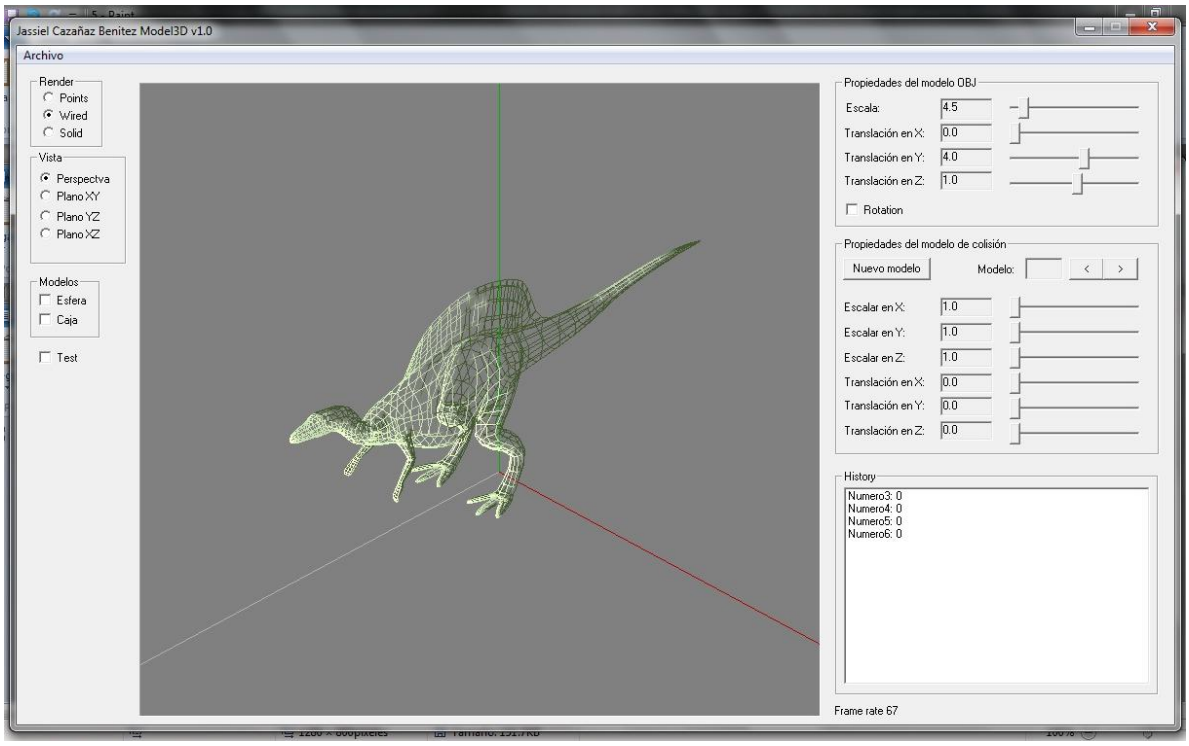


Figura 15: Visualización en modo alambre de un modelo tridimensional

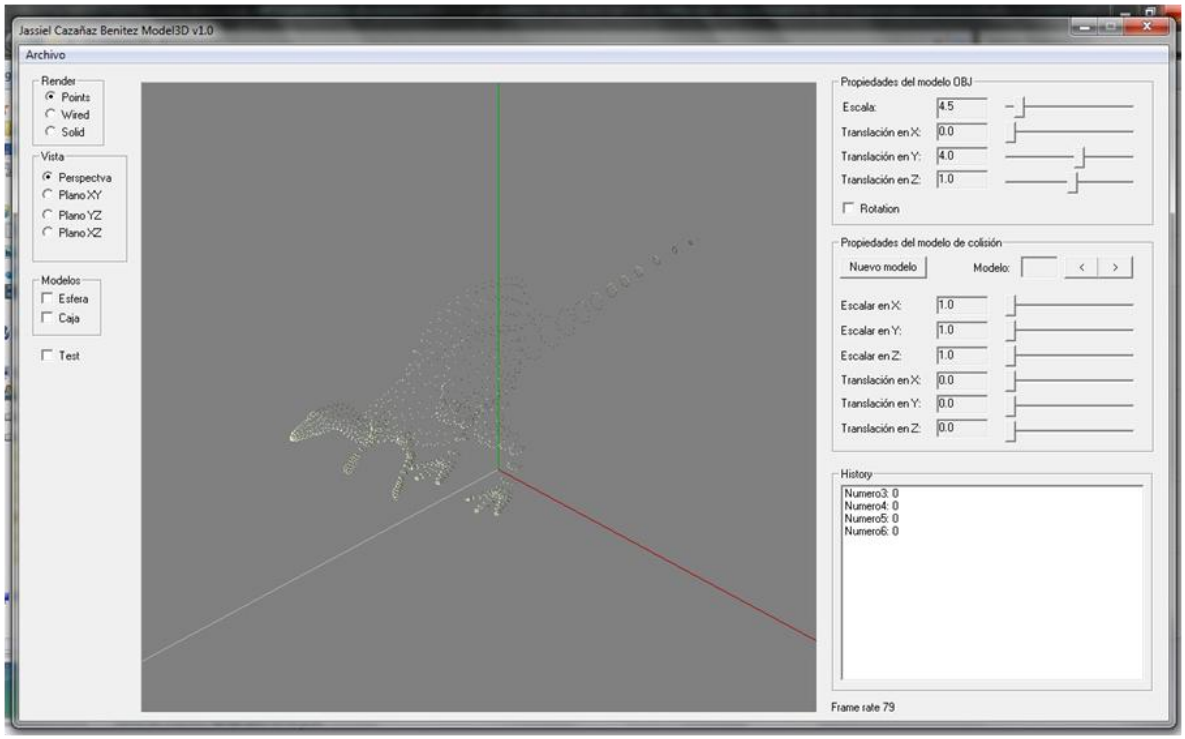


Figura 16: Visualización de los vértices de un modelo tridimensional

Ya que la aplicación calcula de manera automática dos tipos de volúmenes de colisión en base a los vértices que conforman al modelo tridimensional, que son la caja envolvente alineada a los ejes y la esfera envolvente, surge la necesidad de que la interfaz permita al usuario el mostrar u ocultar cada uno de estos modelos, en todas las vistas anteriormente mencionadas. Además, también debe permitir observar estos dos volúmenes de colisión en conjunto con los modelos que se desee añadir posteriormente, igualmente en el modo de test que se explicara más adelante, esto para ejemplificar que en la mayoría de los casos la caja envolvente alineada a los ejes se ajusta mejor al modelo que la esfera envolvente.

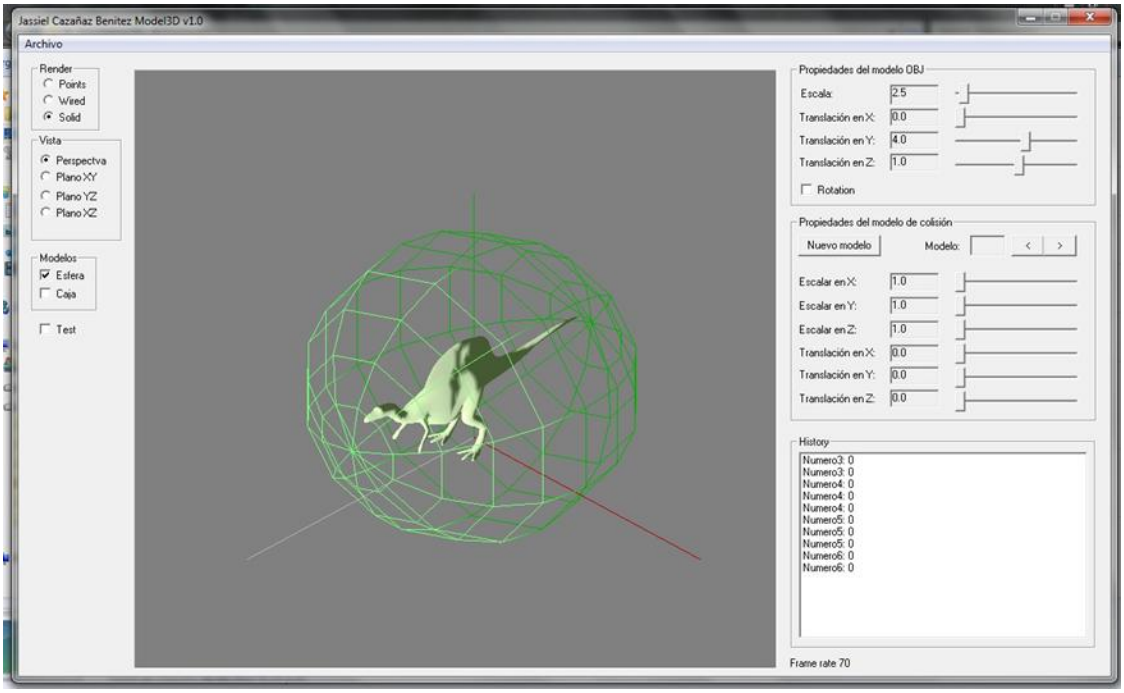


Figura 17: Visualización de la esfera envolvente calculada en base al modelo tridimensional

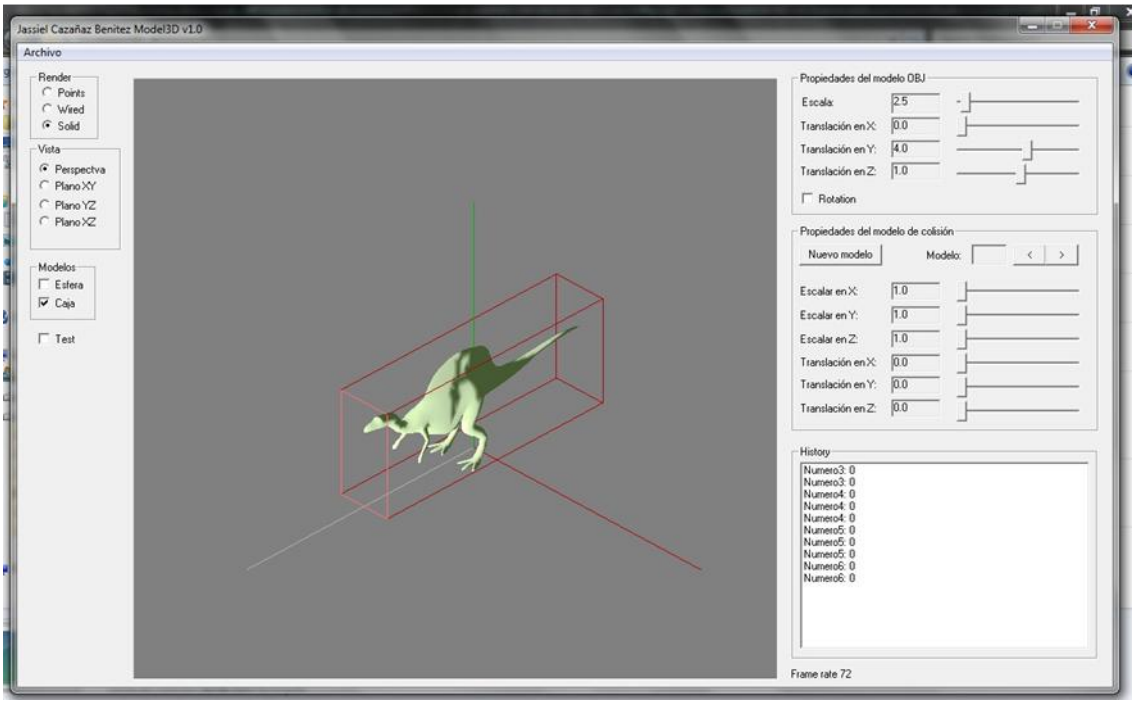


Figura 18: Visualización de la caja envolvente orientada a los ejes, calculada en base al modelo tridimensional

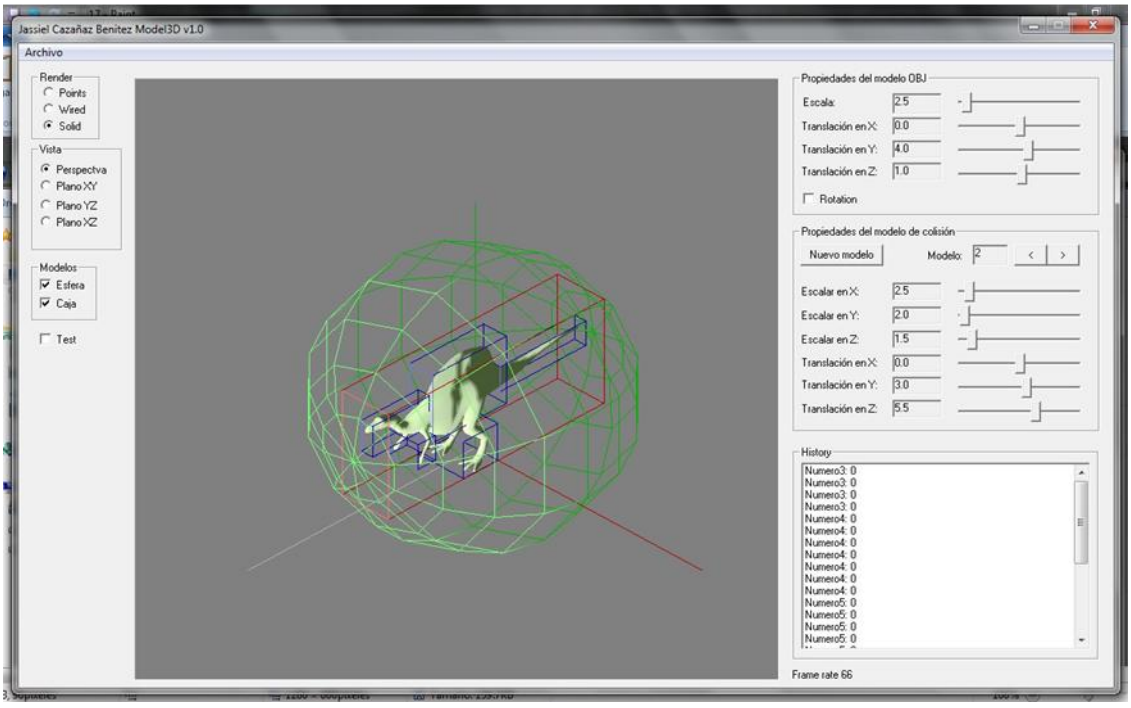


Figura 19: Visualización de varios volúmenes de colisión para un modelo tridimensional

La interfaz gráfica de usuario debe de permitir al usuario el abrir diferentes tipos de archivos, ya sean archivos de modelos tridimensionales, o archivos que contienen información sobre un archivo de modelo tridimensional y sus volúmenes de colisión asociados, que son los archivos con los que esta aplicación funciona. También debe de permitirle guardar la información asociada a los volúmenes de colisión que se le agregaron a un modelo tridimensional, dando la posibilidad de realizar este proceso tantas veces como sea necesario.

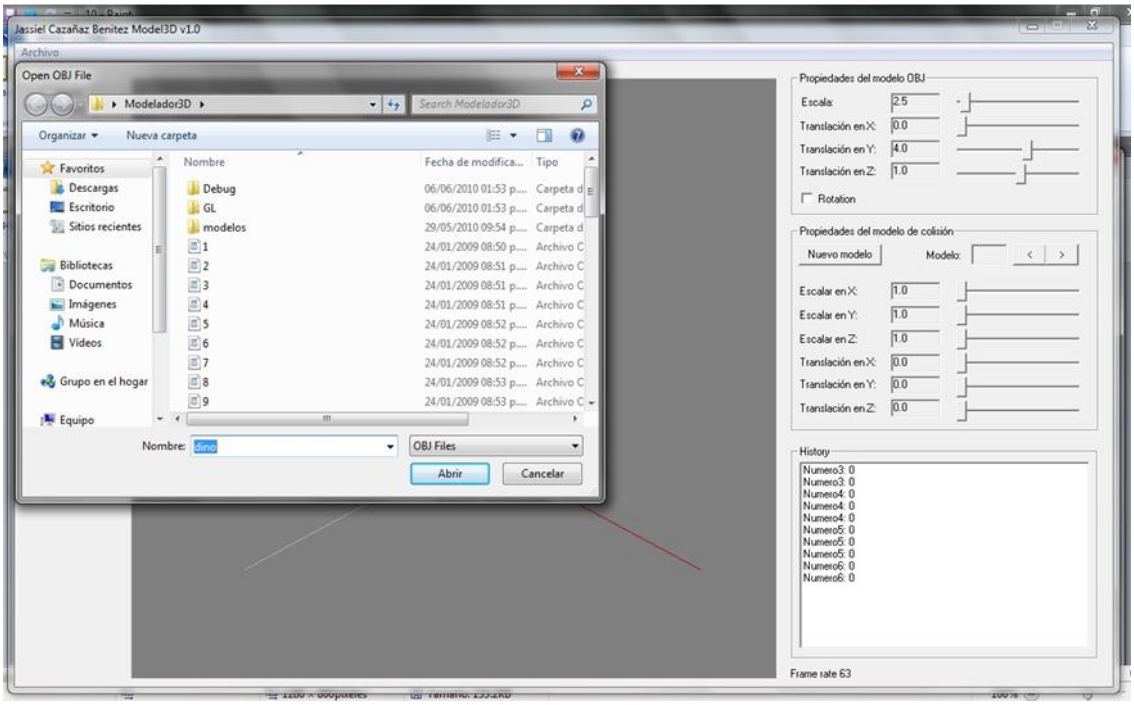


Figura 20: Manejo de archivos a través de la interfaz gráfica de usuario

También la interfaz debe permitir de manera sencilla agregar volúmenes de colisión adicionales al modelo tridimensional y manipularlos, es decir, trasladarlos y modificar sus dimensiones a largo de los ejes X, Y y Z y poder seleccionar cada uno en todo momento para volverlo a manipularlo de ser necesario, ya que este proceso también requiere de un alto grado de precisión y se debe llevar a cabo a través de controles gráficos.

Ya que la aplicación permite al usuario realizar un test para verificar la correcta colocación de volúmenes de colisión, es un requerimiento para la interfaz el permitir al usuario pasar fácilmente del modo de edición, en donde se pueden añadir volúmenes de colisión, y el modo de test. En este último, el poder ver el rendimiento de la computadora en donde se está ejecutando la aplicación resulta muy importante, esto al observar a través de la interfaz gráfica el número de cuadros por segundo con el que se está dibujando el entorno gráfico de nuestra aplicación. También resulta muy ilustrativo el que la interfaz provea una lista de cada acción que se ha realizado a través de la interfaz.

En base a estos requerimientos y a los estándares manejados entre las aplicaciones que trabajan con modelos tridimensionales, es conveniente el uso de el sistema de ventanas que usan los sistemas operativos de mayor uso, sobre los cuales funcionan este tipo de aplicaciones, y que proveen además interfaces

predefinidas para el manejo de archivos, es decir, la apertura y escritura, que es uno de los requerimientos de la interfaz gráfica de usuario de esta aplicación, además de su compatibilidad con los lenguajes y API's con los que está desarrollada esta aplicación para Microsoft Windows, que son: Visual C++, Win32 API, y OpenGL.

De acuerdo con las necesidades de la aplicación el diseño de la interfaz gráfica de usuario consta de las siguientes secciones: un menú que permite abrir un archivo de modelo tridimensional, en este caso el formato usado de modelos tridimensionales es el *.obj, el guardar la información concerniente a los volúmenes de colisión añadidos al modelo tridimensional, el abrir este tipo de archivos de extensión *.fmc y el salir de la aplicación. Otra sección de la interfaz es la que contiene los controles que permiten al usuario elegir la vista del modelo tridimensional, el modo de dibujo, el despliegue de los volúmenes de colisión calculados en base a los vértices del modelo tridimensional y el cambiar entre el modo de edición y el modo de test. También consta de una área de dibujo de los modelos tridimensionales y sus volúmenes de colisión y el número de cuadros por segundo que se dibujan. Otra sección es la que contiene a los controles que permiten la manipulación del modelo, el añadir volúmenes de colisión y manipularlos, y el área donde se muestran las acciones realizadas a través de la interfaz.

Ya que el diseño de la interfaz se basó en estándares utilizados por otras aplicaciones que trabajan con modelos tridimensionales, y ya que esta aplicación está destinada a ser utilizada por usuarios que manejan este tipo de aplicaciones en sistemas operativos de uso común, resulta sencillo el aprender a utilizar esta aplicación mediante la interfaz gráfica descrita.

5.2 Lectura de archivos de modelos tridimensionales

Como se mencionó anteriormente esta aplicación funciona con archivos de modelos tridimensionales con extensión *.obj, que es uno de los formatos de archivo estándar entre las aplicaciones de diseño de modelos tridimensionales, por lo cual se decidió usarla para esta aplicación, además de que su lectura es muy clara para cualquier usuario con conocimiento de cómo está conformado un modelo tridimensional ya que es un archivo de texto que puede ser abierto en cualquier sistema operativo [WAVEFRONT10].

Este tipo de archivo contiene la información concerniente a como está conformado un modelo tridimensional con la siguiente estructura:

- Sentencia general
- Datos de vértices
- Elementos
- Agrupamiento
- Atributos de visualización/render
- Comentarios

Datos de vértices:

v Vértices del modelo (x y z) float

vt Vértices de textura (u v) float

vn Vectores normales(x y z) float

Elementos:

f Polígono(cara) $v_1/vt_1/vn_1$ $v_2/vt_2/vn_2$ … $v_n/vt_n/vn_n$

Agrupamiento:

g Nombre del grupo

o Nombre del objeto

Atributos de visualización/render:

usemtl Nombre del material

mtllib archivo externo de materiales (extensión *.mtl)

Sentencias generales/Comentarios

Antecedidos por el símbolo '**#**', son información sobre el nombre del programa de creación, nombre del archivo, etc.

El archivo de extensión *.mtl, [MTL10] que contiene la información concerniente a la definición de materiales y texturas usados por el archivo de definición del modelo tridimensional tiene la siguiente estructura:

- Definición de materiales
- Componente ambiental del material
- Componente difusa del material
- Componente especular y coeficiente especular
- Mapas de texturas

Definición de materiales

newmtl Nombre del material

Componente ambiental del material

Ka r g b

Componente difusa del material

Kd r g b

Componente especular y coeficiente especular del material

Ks r g b

Ns rango (1-100)

Mapas de texturas para el material

map_Ka Nombre del archivo de textura

map_Kd Nombre del archivo de textura

map_Ks Nombre del archivo de textura

Los algoritmos para leer y almacenar esta información son los siguientes:

Lectura del archivo *.obj

```
mObj myLoad(char* name){
     mObj obj;
     FILE* archivo;
     char leer='#';
     char m[256];
     int a1,b1,c1,d1,e1;
     a1=0; b1=0; c1=0; d1=0; e1=0;

     archivo = fopen(name, "r");
     if ( !archivo ){
          printf("No se ha podido abrir el archivo: %s\n",
name);
     }
     else{
          printf("\nCargado: %s \n\n", name);
```

Esta sección del algoritmo determina el número de vértices, vértices de textura, vectores normales, y caras:

```
          while(leer!=EOF){

               leer=getc(archivo);
               if(leer=='f'){
                    leer=getc(archivo);
                    if(leer==' '){
                         b1++;
                    }
               }
               if(leer=='v'){
                    leer=getc(archivo);
                    if(leer==' '){
                         a1++;
                    }
                    if(leer=='t'){
                         leer=getc(archivo);
                         if(leer==' '){
                              d1++;
                         }
                    }
                    if(leer=='n'){
                         leer=getc(archivo);
                         if(leer==' '){
```

```
                    c1++;
                }
            }
        }
    }

    leer = '#';
    printf("v %d\n",a);
    rewind(archivo);
```

Esta sección del algoritmo reserva la memoria necesaria para el total de vértices, vértices de textura, vectores normales y caras:

```
    obj.g =1;
    obj.nFace=0;
    obj.nVer = 1;
    obj.nNor = 1;
    obj.nVtx = 1;
    obj.v_obj = new vrtx[a1+1];
    obj.n_obj= new nrml[c1+1];
    obj.t_obj = new text[d1+1];
    obj.f_obj = new fces[b1+1];

    for(int i=0;i<b1+1;i++){
        obj.f_obj[i].v = new int[3];
        obj.f_obj[i].vt = new int[3];
        obj.f_obj[i].vn = new int[3];
    }
```

Esta sección del algoritmo almacena la información sobre vértices, vértices de textura, vectores normales, y caras:

```
    while(leer!=EOF){
        leer=getc(archivo);
        if(leer=='v'){
            leer=getc(archivo);
            if(leer==' '){

fscanf(archivo,"%f",&obj.v_obj[obj.nVer].x);

fscanf(archivo,"%f",&obj.v_obj[obj.nVer].y);

fscanf(archivo,"%f",&obj.v_obj[obj.nVer].z);
                    obj.nVer++;
                }
                if(leer=='t'){
                    leer=getc(archivo);
```

```
                    if(leer==' '){

fscanf(archivo,"%f",&obj.t_obj[obj.nVtx].x);

fscanf(archivo,"%f",&obj.t_obj[obj.nVtx].y);

                         obj.nVtx++;
                    }
               }
               if(leer=='n'){
                    leer=getc(archivo);
                    if(leer==' '){

fscanf(archivo,"%f",&obj.n_obj[obj.nNor].x);

fscanf(archivo,"%f",&obj.n_obj[obj.nNor].y);

fscanf(archivo,"%f",&obj.n_obj[obj.nNor].z);
                         obj.nNor++;
                    }
               }
          }
          if(leer=='f'){
               leer=getc(archivo);
               if(leer==' '){
                    for(int j=0;;j++){


fscanf(archivo,"%d",&obj.f_obj[obj.nFace].v[j]);

                         leer=getc(archivo);

fscanf(archivo,"%d",&obj.f_obj[obj.nFace].vt[j]);
                         leer=getc(archivo);

fscanf(archivo,"%d",&obj.f_obj[obj.nFace].vn[j]);
                         leer=getc(archivo);

                         if(leer == '\n'){

                              obj.f_obj[obj.nFace].nE =
j+1;

                              break;
                         }
                    }
                    obj.nFace++;
```

```
                    }
               }
```

Esta sección del algoritmo almacena la información soble los materiales y texturas usados para un grupo de caras:

```
               if(leer == 'm'){
                    fscanf(archivo,"%s",&m);
                    if(strcmp(m,"tllib")==0){
                         fscanf(archivo,"%s",&m);
                         myLoadMTL(m);
                    }
               }
               if(leer == 'u'){
                    fscanf(archivo,"%s",&m);
                    if(strcmp(m,"semtl")==0){

     fscanf(archivo,"%s",&obj.f_obj[obj.nFace].n_mtl);
                    }
               }
          }
     fclose(archivo);
     }
     return obj;
}
```

Lectura del archivo *.mtl

```
void myLoadMTL(char* name){
     FILE* archivo;
     char m[10];

     char leer = '#';

     archivo = fopen(name, "r");
     if ( !archivo ){
          printf("No se ha podido abrir el archivo: %s\n",
name);
     }
     else{
          printf("\nCargado: %s \n\n", name);
          while(leer!=EOF){
               leer=getc(archivo);
```

Esta sección del algoritmo se almacena la información sobre materiales:

```
               if(leer=='K'){
                    leer=getc(archivo);
                    if(leer=='d'){
                         leer=getc(archivo);
                         if(leer==' '){

     fscanf(archivo,"%f",&m_obj[gm].kd[0]);

     fscanf(archivo,"%f",&m_obj[gm].kd[1]);

     fscanf(archivo,"%f",&m_obj[gm].kd[2]);

                         }
                    }
                    if(leer=='a'){
                         leer=getc(archivo);
                         if(leer==' '){

     fscanf(archivo,"%f",&m_obj[gm].ka[0]);

     fscanf(archivo,"%f",&m_obj[gm].ka[1]);

     fscanf(archivo,"%f",&m_obj[gm].ka[2]);
                         }
                    }
                    if(leer=='s'){
                         leer=getc(archivo);
```

```
                    if(leer==' '){

     fscanf(archivo,"%f",&m_obj[gm].ks[0]);

     fscanf(archivo,"%f",&m_obj[gm].ks[1]);

     fscanf(archivo,"%f",&m_obj[gm].ks[2]);
                    }
               }
          }
          if(leer == 'n'){
               fscanf(archivo,"%s",&m);
                    if(strcmp(m,"ewmtl")==0){
                         gm++;

     fscanf(archivo,"%s",&m_obj[gm].nMtl);

                    }

          }
```

Esta sección del algoritmo se almacena la información sobre texturas:

```
          if(leer == 'm'){
               fscanf(archivo,"%s",&m);
                    if(strcmp(m,"ap_Kd")==0){

     fscanf(archivo,"%s",&m_obj[gm].nTxt);
                         m_obj[gm].n_map_Kd =
LoadGLTexture(m_obj[gm].nTxt);
                    }
          }
     }
     fclose(archivo);
}
}
```

Cada grupo u objeto definido en el archivo *.obj está compuesto por un conjunto de caras (faces), a cada uno de estos grupos u objetos es posible asociarles un material definido en el archivo de materiales de extensión *.mtl.

5.3 Calculo de volúmenes de colisión en base al modelo tridimensional

Como fue mencionado anteriormente, al abrir el archivo de un modelo tridimensional son calculados dos de los volumen de colisión descritos en capítulos anteriores, los cuales son la caja envolvente alineada a los ejes, y la esfera envolvente, esto en base al conjunto de vértices que es leído y almacenado mediante el algoritmo para la lectura de archivos de extensión *.obj. La obtención de los datos que definen tanto la posición como las dimensiones de estos dos volúmenes de colisión es a partir de los algoritmos descritos en el capitulo referente a volúmenes de colisión.

Para ejemplificar las ventajas de permitirle al usuario añadir tantos volúmenes de colisión como la aplicación final lo requiera, además de los calculados en base al conjunto de vértices que conforman al modelo tridimensional, esta aplicación para añadir volúmenes de colisión a un modelo tridimensional trabaja con cajas envolventes alineadas a los ejes. La esfera envolvente solo es calculada, para mostrar que en la mayoría de los casos la caja se adapta mejor al modelo que la esfera, pero la elección del volumen de colisión a usar, dependerá de las características de la aplicación final, es decir, del grado de libertad de movimiento que tendrá el modelo tridimensional y de los demás elementos que conformaran el ambiente virtual, además de los recursos de cómputo donde esta es ejecutada.

Para facilitar el trabajo de manipulación de los volúmenes de colisión en conjunto con el modelo tridimensional, como se mencionó anteriormente, la interfaz gráfica de usuario permite mostrar u ocultar los volúmenes de colisión calculados en base al conjunto de vértices del modelo tridimensional. Además de esto para hacer más fácil esta tarea, es manejando un sistema de colores diferente para cada volumen de colisión, la esfera envolvente es de color verde, la caja envolvente alineada a los ejes es roja y las cajas, es decir los volúmenes de colisión que añade el usuario son de color azul.

Para el caso de modo test que es usado para verificar la manera en que interactúan los volumen de colisión en un ambiente virtual en el que el modelo tridimensional se desplaza a lo largo de los ejes coordenados XYZ, la caja envolvente orientada a los ejes cambia a color negro cuando se presenta una colisión y regresa a su color original, es decir el rojo cuando la colisión se ha resuelto.

Ya que la aplicación permite volver a abrir otro modelo tridimensional definido en un archivo *.obj en cualquier momento, los dos volúmenes de colisión son calculados nuevamente en base a los vértices que conforman al modelo tridimensional que se acaba de leer.

5.4 Añadiendo volúmenes de colisión al modelo tridimensional

Como se explicó en el capítulo dedicado al cálculo de volúmenes de colisión en base a los vértices que conforman al modelo tridimensional, estos volúmenes de colisión generalmente no se ajustan a la geometría del modelo tridimensional, y si no es posible realizar un cálculo de la colisión entre los triángulos que conforman a los modelos tridimensionales de un entorno virtual, o el uso de algoritmos que calculan varios volúmenes de colisión en base a los vértices, ya que tanto el cálculo de colisión entre triángulos y estos, requieren de muchos recursos de cómputo. Entonces la mejor opción es el permitir al usuario añadir tantos volúmenes de colisión como la aplicación final lo requiera, es decir con que grado de exactitud con respecto a la geometría del modelo tridimensional es necesario el determinar la existencia de una colisión, ya que entre mayor sea el número de volúmenes de colisión añadidos por el usuario mayor será la exactitud al determinar la existencia de una colisión.

Para que las cajas alineadas a los ejes, es decir, los volúmenes de colisión añadidos por el usuario se ajusten mejor a la geometría del modelo tridimensional en comparación con el uso de un solo volumen de colisión que es la caja envolvente alineada a los ejes, la aplicación permite modificar el tamaño y la posición de estas cajas alineadas a los ejes, a través de controles que permiten un mayor grado de precisión, y para facilitar aun más este proceso, es usado el código de colores mencionado y las diferentes vistas del modelo tridimensional. Además la aplicación permite modificar en cualquier momento las propiedades de estas cajas añadidas por el usuario.

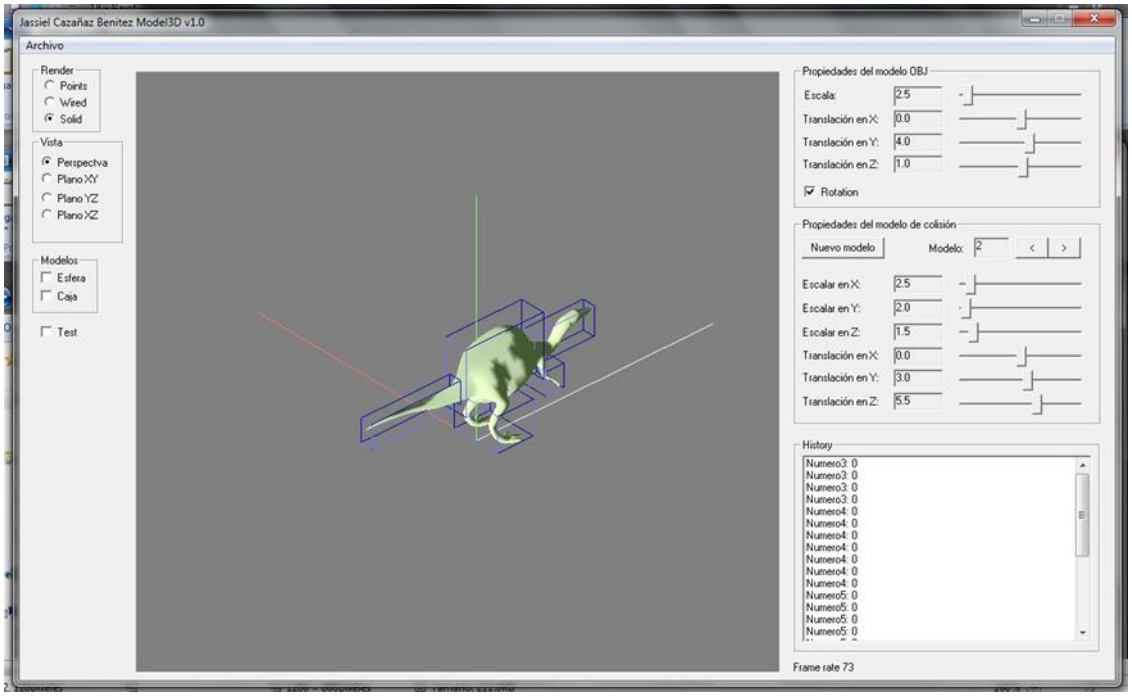


Figura 21: Modelo tridimensional con varios volúmenes de colisión añadidos por el usuario

5.5 Cálculo de colisiones de manera jerárquica

Ya que los algoritmos para determinar la existencia de colisión entre dos volúmenes de colisión resulta un proceso que requiere muchos recursos de cómputo, ya que debe realizarse en tiempo real, una práctica recomendable es el dividir el entorno tridimensional en varias secciones, dependiendo de las dimensiones del ambiente virtual, y sólo calcular la existencia de colisión entre el modelo tridimensional controlado por el usuario y los modelos tridimensionales que se encuentren situados dentro de esa sección del ambiente virtual.

Partiendo del uso de esta técnica para optimizar el uso de recursos de cómputo, es posible aplicarla para este caso, en donde además del volumen de colisión que envuelve al modelo tridimensional, existen volúmenes de colisión añadidos por el usuario y tomando en cuenta que el modelo tridimensional ocupa un menor espacio dentro del ambiente virtual que el volumen de colisión que lo envuelve. Entonces sólo se calculará la existencia de colisión entre los volúmenes de colisión y los demás modelos tridimensionales del entorno virtual, sólo cuando se haya presentado anteriormente una colisión entre algún modelo del entorno virtual y el volumen de colisión envolvente del modelo tridimensional controlado por el usuario en la aplicación final. A partir de este análisis la jerarquía para la

aplicación del algoritmo para determinar si se ha presentado una colisión dentro de un ambiente virtual es la siguiente:

1. Si las dimensiones del ambiente virtual son demasiado grandes, dividir el ambiente en varias secciones a través de volúmenes de colisión de preferencias cajas alineadas a los ejes.
2. Determinar en que sección se encuentra el modelo tridimensional controlado por el usuario, esto a partir de determinar si existe una colisión entre el volumen de colisión envolvente del modelo tridimensional y el volumen de colisión que define a la sección.
3. Determinar si existe una colisión entre el volumen de colisión envolvente del modelo tridimensional y alguno del los modelos tridimensionales que se encuentran dentro de la sección del ambiente virtual.
4. Si existe esta colisión entonces determinar si se está presentando una colisión entre algún modelo tridimensional de la sección del ambiente virtual y alguno de los volúmenes de colisión añadidos por el usuario.
5. Resolver la colisión, es decir, aplicar algún método para que esta ya no se presente en ese momento.

Para ejemplificar este proceso, y mostrar la utilidad de esta aplicación que permite al usuario añadir mas volúmenes de colisión, en este caso cajas alineadas a los ejes, además de la caja envolvente alineada a los ejes calculada en base a los vértices que componen a un modelo tridimensional, esta aplicación cuenta con un modo de test. Este modo permite visualizar al usuario el modelo tridimensional con los volúmenes de colisión que a colocado, y desplazarlo a través de un pequeño ambiente virtual en el cual podrá observar en ejecución el cálculo de colisiones dentro de este ambiente virtual de manera jerárquica. Esto puede observarse cuando existe una colisión entre el modelo tridimensional y alguno de los elementos de este ambiente virtual, ya que la caja envolvente cambiara a color negro y sólo entonces comenzara a determinarse si existe una colisión entre este elemento y los volúmenes de colisión añadidos por el usuario, cuando esto suceda este modelo cambiara a color negro también. La colisión en este modo de test se resuelve al no permitir el desplazamiento del modelo tridimensional en la dirección en la que se ha presentado la colisión, esta condición se mantiene hasta que la colisión deja de presentarse.

Para observar que esta metodología optimiza el uso de recursos de cómputo y sólo hace uso de estos en el momento adecuado la aplicación despliega el número de frames por segundo que se están dibujando, ya que este depende directamente del uso de los recursos entre mayor uso de estos menor será el número de frames por segundo que se estén dibujando.

Ya que esta aplicación fue diseñada para ejemplificar las ventajas de permitirle al usuario añadir volúmenes de colisión, en este caso cajas alineadas a los ejes, el modo de test sólo permite el desplazamiento del modelo tridimensional a lo largo de los ejes coordenados. En el caso de que una aplicación final requiriera que el modelo tridimensional además rotara sobre el eje Y, sería recomendable el uso de cilindros, en lugar de cajas y en el caso de que la aplicación final requiriera que el modelo tridimensional pudiera tener rotación en cualquiera de los ejes, sería recomendable el uso de esferas.

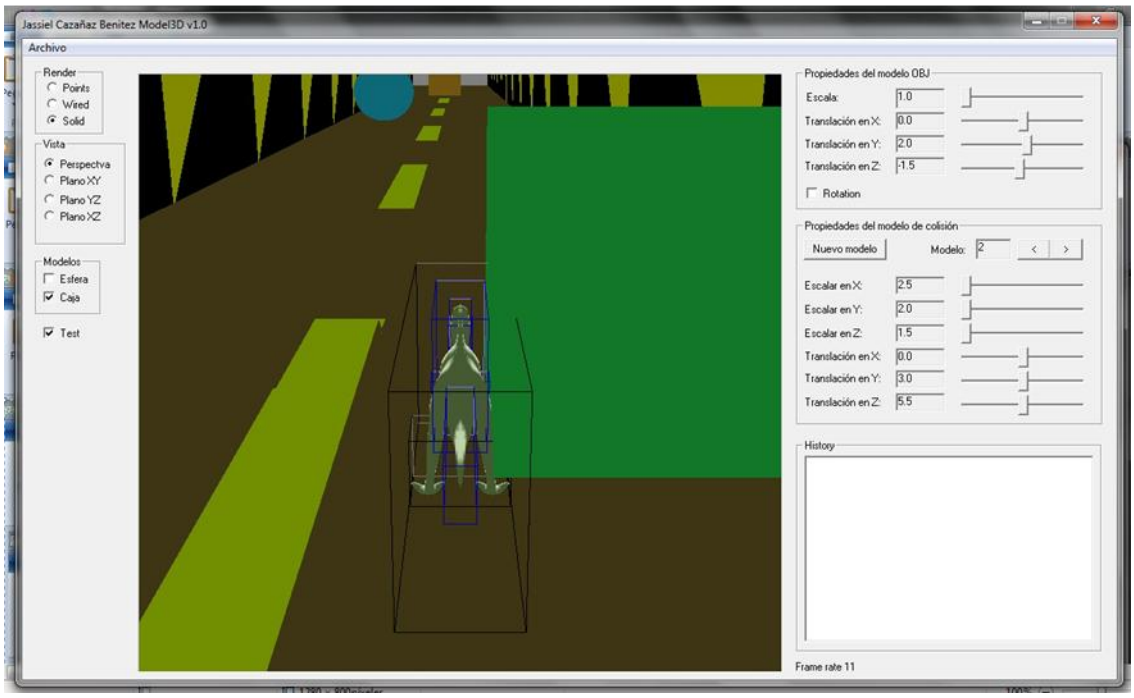


Figura 22: Modo de Test

5.6 Archivo de volúmenes de colisión

Esta aplicación como fue descrito anteriormente permite guardar la información referente a las dimensiones y posición de los volúmenes de colisión añadidos por el usuario, para ser usada por una aplicación final o para modificarla posteriormente a través de esta aplicación. Esta información es guardada en un archivo de tipo texto de extensión *.fmc que tiene la siguiente estructura:

- Nombre del archivo del modelo tridimensional de extensión *.obj
- Numero de volúmenes de colisión añadidos por el usuario
- Definición cada volumen de colisión (posición y dimensiones)

Resultados

Para poder evaluar de manera práctica las ventajas de utilizar varios volúmenes de colisión añadidos por el usuario, en este caso cajas alineadas a los ejes, buscando con esto ajustase mejor a la geometría del modelo tridimensional, debido a la necesidad de poder determinar de manera más exacta la existencia de una colisión. Además de evaluar de manera practica las ventajas de aplicar la metodología de calcular de manera jerárquica la existencia de una colisión, para optimizar el uso de recursos de cómputo. Esto en comparación con solo usar la caja envolvente alineada a los ejes coordenados que es calculada en base al conjunto de vértices que define al modelo tridimensional, se ejecutó esta aplicación en dos computadoras con las siguientes características:

Computadora 1:

Laptop Acer Aspire 3680

Procesador: Intel(R) Celeron(R) M CPU 440 @ 1.86GHz

Memoria RAM: 512 MB

Tarjeta grafica: Familia Mobile Intel(R) 945 Express Chipset

(Microsoft Corporation - WDDM 1.0)

Memoria de gráficos: 64 MB

Sistema Operativo: Windows 7 Ultimate

Computadora 2:

PC DELL Dimension 9150

Procesador: Intel(R) Pentium(R) D CPU 2.80GHz

Memoria RAM: 1.0 GB

Tarjeta grafica: NVIDIA GeForce 7300 LE

Memoria de gráficos: 383 MB

Sistema Operativo: Windows 7 Ultimate

Para poder evaluar la aplicación se uso un modelo tridimensional con las siguientes características:

Modelo: dino.obj

Vértices: 1692

Polígonos: 1690

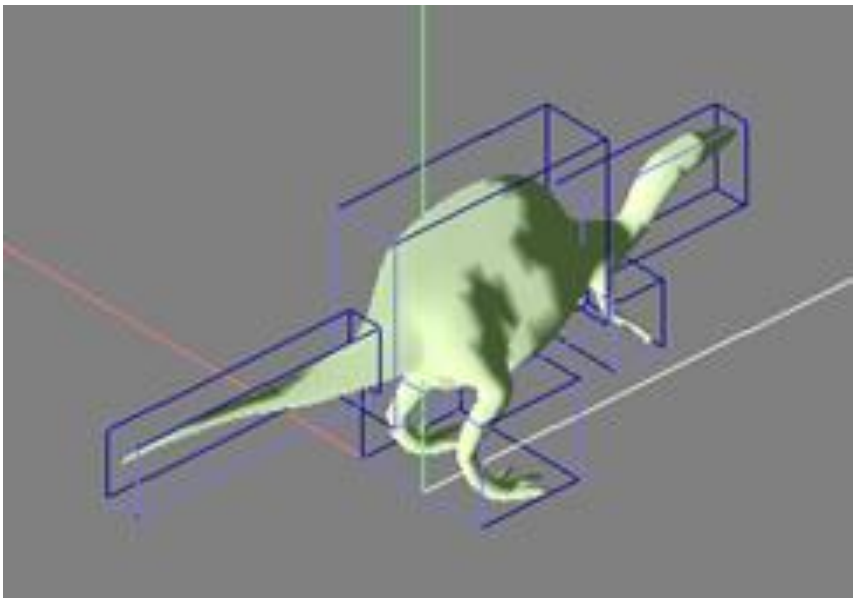


Figura 23: Modelo tridimensional usado en la evaluación de la aplicación.

A este modelo tridimensional se le añadieron cinco volúmenes de colisión, buscando con esto ajustarse mejor a su geometría.

Con estas características se realizo la evaluación de la aplicación en las tres etapas que pueden presentarse, que son:

- Modo de edición, es donde se le permite al usuario añadir varios volúmenes de colisión.
- Modo de test que permite al usuario observar el comportamiento del modelo tridimensional en conjunto con los volúmenes de colisión que añadió previamente, en esta etapa solo se busca la existencia de una colisión entre los volúmenes de colisión envolventes.
- Modo de test cuando se presenta una colisión, esta etapa se presenta cuando se ha encontrado una colisión entre el volumen de colisión envolvente del modelo tridimensional controlado por el usuario y alguno de los objetos existentes en el ambiente tridimensional. En esta etapa se busca entonces, la existencia de una colisión entre los modelos del ambiente tridimensional y los volúmenes de colisión añadidos por el usuario. Cuando este evento ocurre se resuelve la colision.

Usando como medida el número de frames por segundo dibujados en el área de visualización de modelos tridimensionales, al ser ejecutada cada computadora, en las tres etapas que se pueden presentar fue la siguiente:

Computadora 1:

Etapa	Número de frames por segundo
Adición y edición de volúmenes de colisión	66
Modo de test (sin presentarse una colisión)	13
Modo de test (presentándose una colisión)	11

Analizando los datos mostrados en la tabla superior podemos observar que al cambiar al modo de test sin que se presente una colisión, el número de frames por segundo que están siendo dibujados disminuye significativamente, esto debido a las características de la computadora que cuenta con muy pocos recursos tanto de procesamiento, memoria RAM y tarjeta de gráficos que son destinados al uso de aplicaciones gráficas como esta, que en el modo de test hace uso de algunos modelos tridimensionales con un gran número de caras. Pero aún en una computadora con tan pocos recursos, el uso de esta metodología para calcular la existencia de colisiones de manera jerárquica y el uso de volúmenes de colisión no afectó de manera importante el número de frames dibujados por segundo.

Computadora 2:

Etapa	Número de frames por segundo
Adición y edición de volúmenes de colisión	66
Modo de test (sin presentarse una colisión)	59
Modo de test (presentándose una colisión)	57

Analizando los datos mostrados en la tabla superior podemos observar que al cambiar al modo de test sin que se presente una colisión, el número de frames por segundo que están siendo dibujados no disminuye significativamente, a diferencia de la primera computadora, esto debido a las características de la segunda computadora, ya que cuenta con recursos adecuados, tanto de procesamiento, como de memoria RAM y tarjeta de gráficos, recursos que son destinados al manejo de gráficos. Observamos entonces, que el uso de esta metodología para calcular la existencia de colisiones de manera jerárquica y el uso de volúmenes de colisión presenta un muy buen rendimiento, en esta computadora, tomando como referencia el número de frames dibujados por segundo, aun cuando se presenta una colisión.

El uso de recursos de cómputo tanto de memoria como de procesamiento, al ser ejecutada esta aplicación, durante las tres etapas que se pueden presentar fue la siguiente:

Uso de memoria RAM: 26208 KB

Etapa	Uso de CPU	
	Computadora 1	**Computadora 2**
Adición y edición de volúmenes de colisión	10%	10%
Modo de test (sin presentarse una colisión)	82%	60%
Modo de test (presentándose una colisión)	86%	64%

El uso de memoria RAM es constante ya que todos los modelos usados en el modo de test, son almacenados en memoria, en el momento de leer el archivo del modelo tridimensional *.obj, y solo son dibujados al iniciar el modo test. Por otra parte el uso de CPU, es decir de procesamiento, no aumenta significativamente al presentarse una colisión entre el modelo tridimensional controlado por el usuario y algún otro modelo del ambiente virtual del modo de test, lo que confirma las ventajas de usar un calculo jerárquico de colisiones, para optimizar el uso de recursos de computo, en particular de procesamiento.

Conclusiones

A lo largo de este trabajo se desarrollo una aplicación que se baso tanto en los estándares manejados por las aplicaciones de diseño de modelos tridimensionales, como en la metodología de diseño de interfaces gráficas de usuario, que permiten que el aprendizaje y la utilización de esta herramienta sea muy ágil e intuitiva para los usuarios a los cuales está destinada. Esta herramienta fue desarrollada mediante el uso de tecnologías estándar, que no requieren un amplio uso de recursos de computo como son: Win32 API para el desarrollo de la interfaz gráfica de usuario, OpenGL como API para el dibujo de ambientes tridimensionales, y el uso de archivos de extensión *.obj, que es un formato ampliamente conocido y usado por su portabilidad entre las aplicaciones de diseño de modelos tridimensionales.

El objetivo principal de este trabajo fue el mostrar las ventajas que tiene el permitirle al usuario añadir varios volúmenes de colisión, además del envolvente que es calculado en base al conjunto de vértices que conforma al modelo tridimensional. Esto con la finalidad de que al usar un mayor número de volúmenes de colisión aplicados a zonas específicas del modelo tridimensional, sea posible ajustarse mejor a la geometría de este, logrando tener con esto un cálculo más exacto de la existencia de una colisión, esto dependiendo de las necesidades de aplicación final, ya sea esta un videojuego o un ambiente virtual. Optimizando con esto el uso de recursos de cómputo, al calcular la existencia de una colisión de manera jerárquica, partiendo desde verificar la existencia de colisión en los volúmenes que definen una zona del ambiente virtual, luego entre los volúmenes envolventes de los modelos tridimensionales de esa zona y finalmente entre los volúmenes añadidos por el usuario.

Al evaluar de manera práctica esta metodología, se pudo observar que el permitir al usuario añadir volúmenes de colisión para ajustarse mejor a la geometría, a diferencia de solo usar volúmenes de colisión envolventes, y determinar la existencia de una colisión de manera jerárquica, es posible determinar de manera más exacta la existencia de una colisión. Además de que permite un mejor manejo de los recursos de computo tanto de uso de memoria como de procesamiento, reflejándose esto en el numero de frames por segundo que no se ve afectado significativamente, cuando se calcula la existencia de una colisión con los volúmenes añadidos por el usuario.

Ya que la aplicación sólo pretendía mostrar las ventajas del uso de la metodología descrita, algunas de las modificaciones futuras que se le podrían hacer para hacer para mejorar su desempeño, sin decir con esto que no cumplió con su objetivo y para su uso inmediato en aplicaciones finales, sería el optimizar los algoritmos de dibujo para no disminuir significativamente el número de frames por segundo en el modo de test, el incluir la opción de permitirle al usuario añadir y guardar la

información sobre volúmenes de colisión como esferas o cilindros, además de la caja alineada a los ejes. Esto dependiendo de las necesidades de la aplicación final, es decir, si el modelo o modelos tridimensionales del ambiente tridimensional solo tienen desplazamiento sobre los ejes coordenados XYZ, el volumen de colisión mas adecuado seria la caja alineada a los ejes coordenados, si el o los modelos tienen además rotación sobre un solo eje, el volumen de colisión recomendado seria el cilindró, pero si el modelo o los modelos debe poder rotar sobre cualquier eje, el volumen de colisión que daría mejores resultados seria la esfera, aclarando que sólo se recomienda hacer uso de un solo tipo de volumen de colisión para la aplicación final, ya sean cajas orientadas a los ejes, esferas o cilindros.

El uso de esta aplicación, la implementación de los algoritmos y el uso de esta metodología en aplicaciones ya sean de realidad virtual o videojuegos para optimizar el uso de recursos de cómputo, puede ser inmediato, ya que los algoritmos no dependen de las tecnologías utilizadas. Todos los algoritmos para detectar la colisión entre volúmenes de colisión están descritos y explicados detalladamente y la aplicación final sólo hace uso de el archivo de volúmenes de colisión de extensión *.fmc.

Bibliografía

[FOLEY96] FOLEY, James D. et al. *Introducción a la Graficación por Computador.* Mexico: Addison-Wesley Publishing Company, 1996.

[HEARN96] HEARN, Donald; BAKER, M. Pauline. *Computer Graphics, C Version.* 2nd Ed. Portland: Prentice Hall, 1996.

[JIMÉNEZ06] JIMÉNEZ DELGADO, Juan J. "*Detección de colisiones mediante recubrimientos simpliciales*". Dirección: Francisco R. Feito Higueruela. Tesis Doctoral. Universidad de Granada, 2006.

[LEIVA06] LEIVA TORRES, Luis. "*Simulación de entornos virtuales en dispositivos móviles. Implementación en gráficos vectoriales*". Dirección: Ruth Manzanares Rubio. Tesina. Universidad Politécnica de Valencia, 2006.

[LORÉS01] LORÉS, Jesús et al. *Introducción a la interacción persona – ordenador.* Universidad de Lleida, 2001.

[MTL10] "*Material Template Library - Wikipedia, the free encyclopedia*". 2010 [ref. 26 Septiembre 2010]. Disponible en web: http://en.wikipedia.org/wiki/Material_Template_Library

[NEIDER10] NEIDER, Jackie et al. "*The OpenGL Programming Guide - The Redbook".* 2010. [ref. 26 Septiembre 2010]. Disponible en web: http://www.opengl.org/documentation/red_book/

[OBJ10] *"Obj - Wikipedia, the free encyclopedia".* 2010. [ref. 26 Septiembre 2010]. Disponible en web: http://en.wikipedia.org/wiki/Obj

[WAVEFRONT10] *"Wavefront OBJ: Summary from the Encyclopedia of Graphics File Formats"*. 2010. [ref. 26 Septiembre 2010]. Disponible en web: http://www.fileformat.info/format/wavefrontobj/egff.htm