Universidad Autónoma de Sinaloa

Facultad de Informática Culiacán
Facultad de Ciencias de la Tierra y el Espacio

Maestría en Ciencias de la Información

Modelado de Objetos Rígido-Deformables Para Apoyo a la Realización de Tareas de Ensamble Virtual

TESIS

Como requisito para obtener el grado de

Maestro en ciencias de la información

Presenta:

Luis Ángel Darío Osuna Castañeda

Directores de Tesis:

Dr. Ulises Zaldívar Colado

Dra. Xiomara Penélope Zaldívar Colado

Culiacán, sinaloa, méxico septiembre 2017

Dedicatoria

Este trabajo se lo dedico a toda familia, a mi mama Rocío y a mi papa Darío por siempre creer en lo que estoy haciendo y por apoyarme en esta decisión. A mis hermanos Daniel y Saúl por todos los paros que hacen (uiiiiii). Se que el cumplir este objetivo los hará sentirse muy orgullosos.

Agradecimientos

Agradezco principalmente, a mis directores de tesis el Dr. Ulises y a la Dra. Xiomara por todo el apoyo brindado desde que cursaba la licenciatura y mas recientemente en estos últimos dos años durante la maestría y que mas allá de tener un aprecio hacia ellos por ser mis profesores, también lo tengo por haber sido mis amigos durante todo este tiempo. Le doy gracias a mi familia por apoyarme en esta decisión que tome de estudiar un posgrado, por apoyarme no solo económicamente si no también emocionalmente. Agradezco a los compañeros de la maestría por todo el tiempo que tuvimos la oportunidad de compartir. En general a todos los doctores con los que tuve la oportunidad de expandir mis conocimientos, en especial al Dr. Yee por haber sido profesor y amigo durante este tiempo. A la familia de mis directores y en especial a Caro por haberme adoptado como un hijo durante todo el tiempo que duro la estancia. A mi novia Perla por toda la paciencia que ha tenido y por apoyarme durante todo este tiempo que hemos estado un poco mas retirados. Al Roberto por ser mi amigo desde hace mucho tiempo y por toda la ayuda que me ha brindado desde hace mucho tiempo.

También a Daniel Bernal (tati) por haberme apoyado y por haber confiado en mi mientras tuvimos la oportunidad de trabajar juntos.

A la Universidad Autónoma de Sinaloa por ser mi casa de estudios desde la licenciatura y por ayudarme a crecer de manera académica y personal.

Al CONACyT por otorgarme la beca con la cual pude dedicarme por completo a mis estudios de maestría, y con la cual fui capaz de realizar una estancia internacional en Francia.

Lista de Figuras

Lista de Tablas

Lista de Ecuaciones

Lista de Códigos

Índice general

Abstract

The development of a solid-deformable 3D model is presented in this research, the mentioned model can advantage of the low computational cost of solid 3D models and the ability to represent deformation like deformable 3D models. The use of triangular meshes facilitates the creation of mass-spring-damper models, so this model can be deformed in particular regions of it. This regions can be represented by "depth deformation levels", where the depth level determines the region of the model that can be deformed.

To restore deformable model to its original shape, a technique implementing constraint springs inspired in [6, 29] is presented. By using this type of springs, force can be exerted onto its associated masses until their positions became the same as their initial position.

Using techniques like space partitioning, hierarchies and bounding volumes, a novel collision detection system for this kind of models is developed. Implementing spheres as bounding volumes in both models and hierarchies. Sphere were used because of its low memory cost, fast overlapping test and mainly for its rotational invariance.

At last, a technique inspired in the improved snap-fitting technique [49] is presented. This technique establish an area where a pair of parts (primary part and secondary part) can be assembled. During the assembly process an assembly area is defined into the secondary part, meanwhile the primary part is represented by a vector, such as in the improved snap-fitting technique. When the vector that represents the primary part is inside the assembly area defined in the secondary part, the primary part performs the neccesary geometric transformations to be placed into its final assembly position with respect to the secondary part.

Resumen

En este trabajo se presenta el desarrollo de un modelo rígido-deformable, dicho modelo es capaz de aprovechar el bajo costo computacional que otorgan los modelos rígidos y a su vez tener la capacidad de representar cambios en el modelo al aplicar una determinada fuerza sobre el mismo, tal y como ocurre con los modelos deformables.

Se considera que el uso de objetos representados a través de mallas triangulares para tener la facilidad de implementar el sistema masa-resorte-amortiguador (MRA) para representar las zonas deformables del modelo. Dichas zonas permiten niveles de profundidad de deformación, donde a mayor sea el nivel de profundidad, mayor será la región del modelo que podrá deformarse. Para restaurar el modelo deformable a su forma original, se utiliza una técnica inspirada en [6, 29]. Al implementar restricciones sobre los resortes del modelo, es posible que estos apliquen fuerzas sobre sus masas asociadas para que estas retornen a su posición inicial.

Además del modelo rígido-deformable, se presenta el desarrollo de un sistema detector de colisiones para modelos deformables, apoyándonos en el enfoque de división espacial del entorno, representando al ambiente virtual y a los modelos rígido-deformables a través de jerarquías. Las jerarquías implementadas utilizan esferas como volúmenes delimitadores, estos fueron seleccionados debido a su simplicidad de representación, bajo costo de memoria, rápida prueba de intersección y sobre todo por ser rotacionalmente invariables.

Por último, se propone una técnica para el ensamble de objetos inspirada el snap-fitting mejorado utilizado en [49], dicha técnica se basa en definir áreas de ensambles en las cuales un par de objetos (pieza primaria y pieza secundaria) serán ensambla-

dos. Durante el ensamble de las piezas, la pieza secundaria contiene un espacio que representara el área de ensamble, mientras que la pieza primaria contiene un vector de ensamble tal y como el que se utiliza en el snap-fitting mejorado. Cuando este vector este completamente contenido dentro del área de ensamble, la pieza primaria será afectada por las transformaciones geométricas necesarias para localizarse en su posición final de ensamble con respecto a la pieza secundaria.

Capítulo 1

Introducción

1.1. Planteamiento del problema

El proceso de desarrollo de un producto puede ser divido en tres fases: la fase de diseño, la fase de ingeniería y la fase de producción. Debido a que la primera fase representa un mayor costo (hasta un 70 %) [7, 37], se han buscado alternativas que permitan reducir estos costos. La realidad virtual resulto ser una buena opción durante esta fase del desarrollo, principalmente debido a que no es necesario generar prototipos físicos para determinar cuál será el resultado del producto desarrollado. El uso de prototipos virtuales presenta numerosas ventajas, entre ellas: su rápida creación, modificación y disponibilidad de las piezas en todo momento [34].

Existen diversos sistemas de ensamble virtual. Una característica común entre estos es que solo utilizan modelos rígidos para su funcionamiento [16, 22, 42, 50] y en el mejor de los casos es posible realizar un proceso de ensamble realista al no permitir la interpenetración de los objetos que se encuentran en el entorno virtual [14]. Sin embargo, las técnicas que se utilizan para llevar a cabo el ensamble suelen ser poco realistas, mostrando ventanas emergentes para determinar si en realidad se desea ensamblar dos piezas, al realizar el ensamble automáticamente cuando dos piezas se tocan o cuando se encuentran a una distancia lo suficientemente cercana,

inclusive si las piezas no se encuentran en la orientación correcta, o en caso de que se encuentre correctamente orientada existen casos en los que el ensamble se lleva a cabo aunque las piezas se encuentren en una posición muy diferente a la que se encontrarían una vez ensambladas. Además de esto, en la realidad es posible que en las piezas ocurran deformaciones al aplicar la suficiente fuerza sobre ellas, por lo tanto ignorar los cambios que sufre el modelo puede no representar correctamente el comportamiento de estas [24].

El uso de modelos deformables permite representar los cambios que ocurren en las piezas, sin embargo el costo computacional de estos hace que el rendimiento de los sistemas de ensamble virtual disminuya, lo anterior se debe a que es necesario realizar cálculos constantemente sobre ellos, incluso si no existe interacción alguna con los mismos.

1.2. Motivación

Actualmente el uso de modelos rígidos predomina en los sistemas de ensamble virtual. El uso de modelos deformables puede ser una alternativa que otorgue un mayor realismo en estas tareas. Ya que el uso de modelos deformables representa un alto costo computacional es estos sistemas, la implementación de un modelo híbrido (rígido y deformable) puede disminuir el costo computacional obtenido en los modelos deformables, y a su vez aprovechar la naturaleza de estos para representar un comportamiento realista en los objetos a ensamblar.

1.3. Solución propuesta

Con la finalidad de aumentar el realismo durante las tareas de ensamble virtual al implementar modelos de piezas que permitan representar deformaciones y cuyo costo computacional no resulte restrictivo, se propone el desarrollo de un modelo rígido-deformable. Dicho modelo tomara la principal ventaja de los modelos rígidos

al requerir un bajo costo computacional para su representación, mientras se encuentre sin interactuar con otros objetos del entorno virtual y una vez que interactúe con estos, se representarán deformaciones en las regiones cercanas a los puntos de contacto. De esta manera el modelo propuesto podrá beneficiarse de los cambios que se pueden representar en los modelos deformables.

1.4. Justificación

El uso de modelos deformables en las distintas aplicaciones de realidad virtual representa un alto costo computacional. Al modelar objetos utilizando el sistema MRA se es capaz de hacer representaciones realistas de estos. Sin embargo, para que estos objetos sean lo más parecidos a la realidad es necesario utilizar una gran cantidad de masas y resortes que los representen, lo cual conlleva una gran cantidad de cálculos.

Dependiendo de la aplicación en la que se utilicen los modelos deformables, no siempre es necesario calcular las deformaciones que ocurren en todo el objeto, esto debido a que existen materiales que solo llegan a representar deformaciones notorias en zonas cercanas a el área con la cual exista una interacción.

Comúnmente en los sistemas de ensamble virtual, el uso de modelos deformables es restringido debido a su costo computacional, y solamente se utilizan modelos rígidos e incapaces de representar deformaciones en los objetos. Debido a que existen ocasiones en que los materiales que se utilizan en el proceso de ensamble de un producto pueden sufrir deformaciones, los modelos rígidos no representan un buen acercamiento del comportamiento que los objetos deberían mostrar. Además de esto, cuando las piezas a ensamblar presentan diseños complejos, el realismo con el que se lleva a cabo la fase final de la tarea de ensamble se ve reducido.

Es por esto que el uso de un modelo que permita representar deformaciones y a la vez no afecte en gran medida el rendimiento de las aplicaciones, representa una alternativa atractiva en los sistemas de ensamble virtual. Además se pueden mejorar

la eficacia de las técnicas utilizadas para llevar a cabo el proceso de ensamble debido a que estos modelos otorgan mayor flexibilidad durante su interacción.

1.5. Objetivo General y específicos

Desarrollar el modelo de un objeto tridimensional rígido-deformable que pueda ser manipulado por un operador de ensamble virtual. Dicho objeto se comportará como un sólido rígido antes y durante la primera etapa de manipulación. Durante la interacción con otros objetos de su entorno, el objeto manipulado responderá a deformaciones en su área o en sus áreas cercanas al contacto con los otros objetos.

Objetivos específicos

Para lograr el objetivo general se proponen los siguientes objetivos específicos:

- Desarrollar de un modelo tridimensional rígido que pueda ser manipulado por un operador de ensamble virtual.
- Desarrollar de un modelo tridimensional deformable que pueda ser manipulado por un operador de ensamble virtual.
- Determinar la cantidad de deformación de un modelo a partir de aplicar una fuerza en alguna zona del mismo.
- Recuperar la forma de los objetos deformables que utilizan el sistema masa-resorte-amortiguador.
- Implementar zonas de deformación en los objetos deformables, a través de la conformación de vecindades utilizando las características del modelo.
- Desarrollar de un sistema de detección de colisiones que permita representar las deformaciones que ocurren en un objeto del entorno virtual al interactuar con otros.

1.6. Hipótesis

La implementación de un modelo 3D rígido-deformable basado en el sistema masa-resorte-amortiguador que represente deformaciones en regiones de interés, incrementara el desempeño en comparación a los modelos 3D deformables.

1.7. Organización de la tesis

A continuación se presenta la organización de este documento de tesis.

En el capítulo II se detalla el marco teórico, donde se especifica todo el sustento teórico de este trabajo. Se presentan los elementos por los cuales están conformados los modelos realizados a partir de una de las técnicas más importantes para la creación de modelos deformables basados en física (Sistema masa-resorte-amortiguador). También se detallan todos los aspectos necesarios para la creación de un sistema detector de colisiones, desde el uso de volúmenes delimitadores para representar los objetos del entorno virtual hasta estructuras de datos avanzadas utilizadas para reducir la complejidad en el proceso de detección de colisiones.

El capítulo III se analizan algunos de los sistemas de ensamble virtual existentes, así como las técnicas de ensamble que se utilizan. Además se mencionan algunos de los trabajos más representativos sobre modelos deformables, haciendo énfasis en aquellos que utilizan el sistema masa-resorte-amortiguador. Las características de la implementación de aquellos modelos que no utilizan el sistema masa-resorte-amortiguador fueron de ayuda para encontrar solución a problemas que se presentaron durante el desarrollo del modelo. Además de mencionar los modelos deformables, se mencionan los estudios realizados para la detección de colisiones, analizando cuales son las técnicas más importantes para su desarrollo, y los resultados que estas otorgan en comparación de otras ya existentes.

El capítulo IV está compuesto por la metodología implementada para el desarro-

llo del modelo, desde determinar cuáles son las posibles áreas de deformación de un objeto a partir de una fuerza ejercida en alguna región de este, hasta su implementación utilizando un sistema de detección de colisiones personalizado para los modelos rígido-deformables. En adición a esto se presenta la experimentación realizada, donde se muestran las ventajas y desventajas de utilizar el modelo propuesto. Las conclusiones finales del trabajo de tesis y trabajo futuro se presentan en el capítulo V. Por último, se presentan las referencias bibliográficas utilizadas en el trabajo y los anexos.

Capítulo 2

Marco teórico

2.1. Modelos tridimensionales

Un modelo tridimensional o modelo 3D, generalmente es una representación poligonal en tres dimensiones de un objeto, usualmente mostrado en una computadora o a través de algún otro dispositivo de video. Estos objetos pueden ser representaciones de una entidad del mundo real o cualquier objeto ficticio, desde objetos atómicos hasta objetos gigantes. Cualquier cosa tangible que pueda existir en el mundo físico, puede ser representada como un modelo 3D [19].

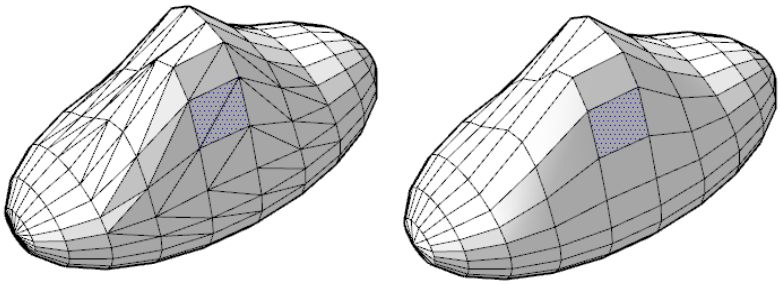

Figura 2.1: Figuras con diferente malla poligonal; a la izquierda una con una malla poligonal compuesta por triángulos y a la derecha una malla poligonal compuesta por cuadrados.

Una de las técnicas más utilizadas para representar objetos 3D, es el uso de mallas poligonales (comúnmente triángulos o cuadrados) (Figura 2.1). Estas mallas están compuestas por al menos 3 elementos fundamentales: vértices, aristas y caras. Y por la descripción de su topología y geometría.

La topología del objeto está dada por como la caras de este están conectadas, y la geometría por cómo se encuentran ubicadas en el espacio (Figura 2.2).

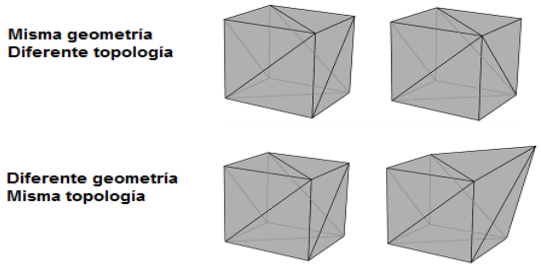

Figura 2.2: Diferencias entre topología y geometría de un modelo.

Cada uno de los vértices que componen al objeto 3D contiene las coordenadas (x, y, z) en el espacio en el que este se encuentra. Las caras se forman a partir de los vértices que la componen, y las aristas se pueden determinar por la unión de cada par de vértices de la cara (Figura 2.3).

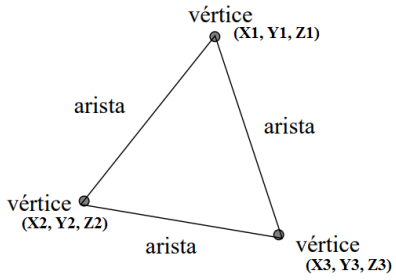

Figura 2.3: Composición un triángulo utilizado en los modelos 3D.

Los objetos 3D se dividen en dos categorías [13]:

- Objetos 3D rígidos
- Objetos 3D deformables

Los objetos 3D rígidos se caracterizan principalmente por ser sencillos de representar, debido a que es considerado como un objeto sólido, este tipo de modelos no es capaz de representar deformaciones, por lo que al aplicar alguna transformación geométrica sobre alguna de sus vértices implica que esta misma transformación se aplicara sobre todos sus vértices restantes.

Los objetos 3D deformables surgen por la necesidad de poder representar con mayor realismo a los objetos modelados. En la realidad todos los objetos son deformables hasta cierto punto, ya que al existir un contacto con estos, se producen deformaciones, que aunque pueden ser pequeñas, en algunas aplicaciones son significativas. Sin embargo el uso de este tipo de modelos requiere de mayor poder computacional para su representación [35].

Entre las técnicas utilizadas para el modelado de objetos deformables se encuentran el uso de splines, forma libre, modelos MRA y modelo de elemento finito (FEM) [15]. De las técnicas mencionadas anteriormente, la FEM es utilizada ampliamente en distintas áreas de investigación como la animación [21] y medicina, y sobre todo en aplicaciones de ingeniería, debido a los resultados precisos que se obtienes con este tipo de modelos. Sin embargo, un inconveniente que estos presentan es que su costo computacional es alto. Como alternativa a estos modelos, este trabajo se centra específicamente en el modelado de objetos utilizando el sistema MRA, debido a la sencillez en el modelado de los objetos y a la rapidez que ofrece en las simulaciones, debido a la poca cantidad de cálculos necesarios en cada resorte [6].

2.2. Modelo masa-resorte-amortiguador

Este tipo de modelos ha sido utilizado ampliamente en el modelado de objetos deformables. Un objeto de este tipo está conformado por un conjunto de masas conectadas por resortes en una estructura de reja [15], la cual fácilmente puede ser adaptada a partir de cualquier malla poligonal sustituyendo los vértices de esta por masas y las aristas por resortes [6].

Normalmente los resortes que componen estos modelos son lineales (Figura 2.4), también llamados resortes hookeanos, sin embargo se pueden modelar resortes no lineales que permitan modelar objetos con propiedades no isotrópicas, como la piel humana [4].

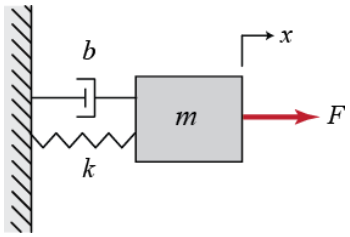

Figura 2.4: Representación de un modelo masa-resorte-amortiguador compuesto por un resorte lineal y una masa.

La fuerza ejercida por un resorte a una masa, es proporcional al cambio de longitud del resorte y siempre está en dirección de su posición inicial. Como se muestra en la Figura 2.5, es posible calcular esta fuerza de acuerdo a la ley de Hook $\boldsymbol{F}=\boldsymbol{-kx}$, que establece que la fuerza que ejerce un resorte se calcula a partir de la diferencia que existe entre la longitud de equilibro del resorte y su longitud actual ($\boldsymbol{x}$) multiplicada por una constante de rigidez del resorte $\boldsymbol{k}$. Cuando un resorte se estira o se compacta, la diferencia en su longitud cambia en $\boldsymbol{x}$ con respecto a su estado inicial, la fuerza ejercida es conocida como fuerza de restauración, que actúa sobre el resorte para que este retorne a su longitud original (Figura 2.4), las variables restantes del sistema son la constante de amortiguación $\boldsymbol{b}$ y la fuerza $\boldsymbol{F}$ ejercida por el resorte hacia la masa m.

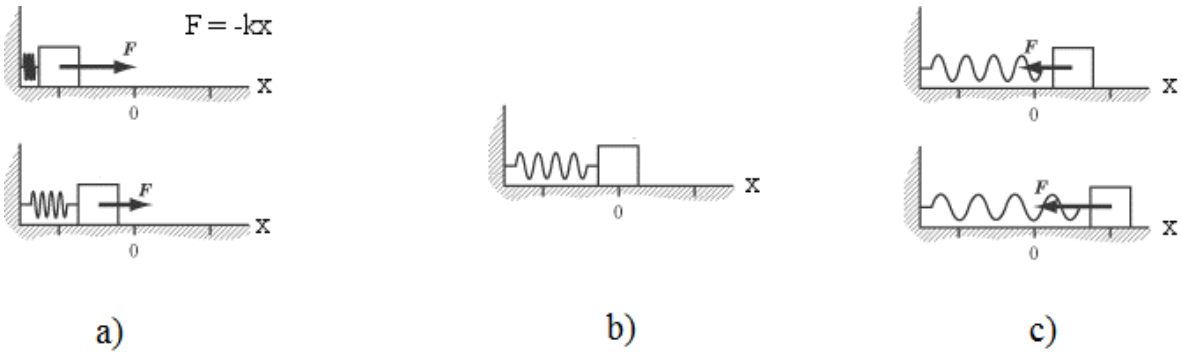

Figura 2.5: a) Modelo MRA donde la longitud del resorte es menor a su longitud inicial b) Modelo MRA en estado de reposo (el resorte no ejerce ninguna fuerza) c) Modelo MRA donde la longitud del resorte es mayor a su longitud inicial.

En un sistema dinámico, el movimiento de una masa en el modelo MRA es gobernado por la segunda ley de newton $ma{=}F$ (Ecuación 2.1).

$$m_i\ddot{x}_i = -\gamma\dot{x}_i + \sum_j g_{ij} + f_i \tag{2.1}$$

Donde m_i representa una masa, $x_i \in \mathcal{R}^3$ su posición y los términos de la derecha son fuerzas que actúan sobre la misma. El primer término de la derecha es una fuerza de amortiguación dependiente de la velocidad, g_{ij} es la fuerza aplicada a la masa i por el resorte que une a las masas i y j, y por ultimo f_i es la suma de otras fuerzas externas tales como la gravedad u otras fuerzas aplicadas por el usuario sobre la misma masa.

2.2.1. Representación del modelo masa-resorte-amortiguador

El uso de los modelos basados en el sistema masa-resorte-amortiguador ha sido previamente adoptado, principalmente por su sencillez de modelar los objetos. Para la representación de los modelos 3D rígidos es necesario conocer cuáles son los vértices,

aristas y caras que lo componen. Por otra parte para los objetos modelados utilizando el sistema MRA es necesario conocer las caras, las masas y los resortes que lo componen, además de determinar algunas de las propiedades de estos componentes como la constante de rigidez y de amortiguación de los resortes.

Una manera simple de representar un modelo MRA (Figura 2.6) a partir de una malla poligonal conocida, es agregar los atributos necesarios a los vértices y aristas, para que estos cumplan con las condiciones necesarias para implementar el modelo.

Cada vértice $i \in [1, \ldots, N]$ de un modelo dado representa una masa en el sistema y está se compone por:

- m_i = masa en kg.
- x_i = posición de la masa.
- v_i = velocidad de la masa.

Cada resorte $i \in [1, \ldots, M]$ de un modelo dado representa una arista en el sistema y está se compone por:

- k_j = constante de rigidez del resorte.
- b_j = constante de fricción del resorte (amortiguador)..
- $m_a y m_b$ = masas asociadas al resorte.

Se presenta la versión más sencilla de un modelo MRA, sin embargo existen otro tipos de representaciones más complejas, y que además permiten representar distintos tipos de materiales, tales como: piel humana, cabello, músculos, etc...

Además de la representación de estos modelos, es necesario implementar un sistema que permita detectar cuando estos estén en contacto con otros objetos del entorno virtual, para esto existen distintos métodos de detección de colisiones, dentro de los

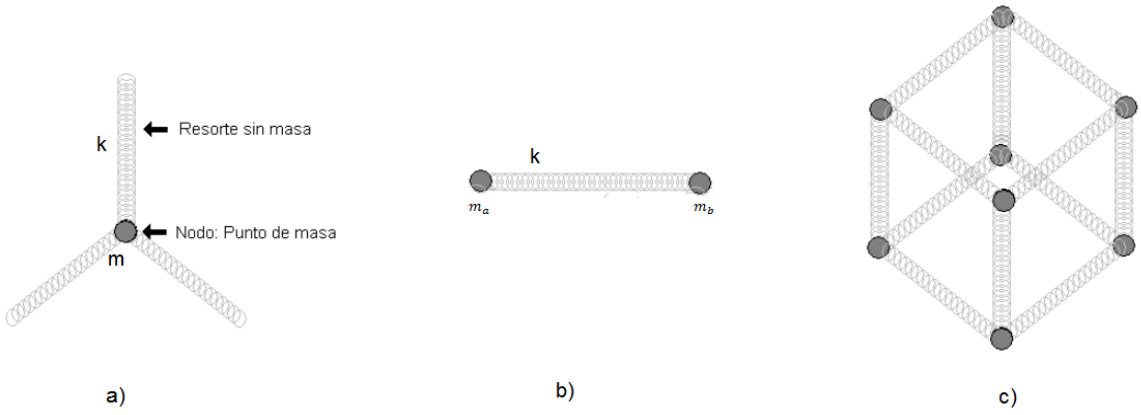

Figura 2.6: a) Representación del modelo MRA en una parte de la malla poligonal b) conformación del resorte en el modelo MRA (un resorte asociado a dos masas) c) Cubo compuesto por resortes y masas.

cuales destaca el uso del sistema de detección de colisiones por jerarquías.

2.2.2. Rotaciones y traslaciones

OpenGL es una librería para la manipulación de gráficos 3D. Dicha librería proporciona la capacidad de crear aplicaciones interactivas que renderizan imágenes de alta calidad a partir de objetos 3D [47]. Esta librería cuenta con funciones que permiten realizar traslaciones y rotaciones en los objetos renderizados en el ambiente virtual, el problema con estas trasformaciones es que solo son representadas visualmente, por lo que las funciones de traslación y rotación como glTranslatef() y glRotatef() resultan de nula ayuda cuando es necesario obtener información para determinar si existen colisiones entre los objetos.

Las rotaciones pueden ser aplicadas en el eje x, el eje y, el eje z o en un eje arbitrario donde la x+y+z=1.

Las rotaciones básicas se aplican sobre cualquiera de los ángulos del sistema de coordenadas y sus matrices de rotación son representadas como se muestra en la

Ecuación 2.2,2.3 y 2.4. Cabe resaltar que las rotaciones que se representan en estas matrices son al sentido contrario del reloj.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{2.2}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.3}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Para realizar rotaciones en un modelo MRA con un conjunto de masas y resortes definidos es necesario crear una matriz de rotación que represente la cantidad de grados y el eje en el que se rotara. Una vez creada esta matriz será necesario realizar una multiplicación de la matriz de rotación por el vector de posición de cada una de las masas que contiene el modelo MRA para obtener su nueva posición en el espacio (Ecuación 2.5).

$$R_z(\theta) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \tag{2.5}$$

El caso de aplicar una traslación a una masa del modelo MRA es menos complicado, se suma al vector de posición de esta el vector de traslación deseado, lo cual resultara en la nueva posición de la masa (Ecuación 2.6).

$$\begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 \\ 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 7 \\ 3 \\ 0 \end{bmatrix} \tag{2.6}$$

2.2.3. Transformaciones en el objeto

Para representar las traslaciones y rotaciones de un modelo MRA en el ambiente virtual, y que además estas sigan siendo de ayuda para el sistema detector de colisiones, es necesario que al especificar un movimiento en el objeto, este movimiento se aplique a cada uno de las masas y a su vez actualice su información. Es por eso que se utiliza una matriz de transformación en cada objeto, que permita conocer los cambios que se han aplicado en esta, y a la vez sea de utilidad para determinar posibles colisiones.

La matriz de transformación del objeto es representada por una matriz de 4x4 la cual almacenara las rotaciones y traslaciones que han ocurrido en este tal y como se muestra en la Ecuación 2.7

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Aplicar una rotación sobre la matriz de transformación del objeto, se realiza de la misma manera que en la Ecuación 2.5, con un cambio en la matriz de rotación generada; ya que es necesario que esta también sea una matriz de 4x4. Para que esto ocurra se utiliza la matriz de 3x3 generada de acuerdo a la rotación requerida y los espacios restantes se llenan con ceros y unos para conservar la homogeneidad de la matriz, esto si las rotaciones se darán de acuerdo al centro del objeto. En caso contrario la última columna se llenara con los valores de la posición con respecto a la cual se hará la rotación (Ecuación 2.8).

$$R_z(90) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

El resultado de la transformación del objeto en la Ecuación 2.8 da como resultado una nueva matriz de transformación, la cual se utiliza para calcular la posición de las masas del modelo, multiplicando a la matriz por el vector de posición de cada una de estas para obtener su nueva posición. Debido a que el vector de posición solo contiene 3 elementos (las posiciones de 'x', 'y' y 'z') es necesario homogeneizar el vector agregando un 1 para permitir la multiplicación entre estos (Ecuación 2.9).

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 3 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 12 \\ 0 \\ 1 \end{bmatrix} \tag{2.9}$$

2.3. Sistema detección de colisiones

Un entorno virtual es un mundo creado por computadora con objetos virtuales dentro de este. Estos ambientes deben brindar al usuario la sensación de presencia, lo que debe hacer que las imágenes que se encuentran en este se perciban como objetos sólidos. Por ejemplo, los objetos no deben pasar a través de otros, y estos deben de actuar como en la realidad cuando son empujados, jalados o incluso agarrados. Para lograr estas acciones, es necesario un sistema detector de colisiones preciso que permita brindar una sensación realista [12]. Es posible que el mundo virtual este compuesto por cientos, o incluso miles de objetos, por lo que los algoritmos de fuerza bruta pueden llevar mucho tiempo en determinar si existen colisiones entre los objetos.

A continuación, se explica cuáles son los tipos de consulta más comunes que se requieren en un sistema detector de colisiones, las fases que los conforman, así como las técnicas más utilizadas para llevar a cabo esta tarea.

2.4. Tipos de consulta

En un entorno virtual compuesto por una N cantidad de objetos, la consulta más importante es saber si alguno de los objetos con los que se interactúan entra en contacto con algún otro objeto de su entorno, lo cual genera una respuesta booleana. Sin embargo, en algunas ocasiones esto no es suficiente, y es necesario determinar también cuales son los puntos de contacto en los objetos que se intersectan. Además de esto, otra consulta importante y que no siempre es requerida, es conocer la interpenetración de dos objetos que se encuentran colisionando.

2.5. Fases de un sistema detector de colisiones

En un mundo virtual compuesto por N objetos, es necesario realizar $O(n^2)$ pruebas para determinar cuáles son los objetos que en un determinado momento se encuentran colisionando. Debido a la complejidad cuadrática que se encuentra al poder determinar que objetos colisionan, el usar métodos de fuerza bruta para verificar cada uno de los objetos existentes en el entorno virtual se vuelve rápidamente muy costoso de calcular, incluso si el tamaño de N no es demasiado grande.

Para reducir el número de pruebas que se realizan en un método de fuerza bruta, la tarea de detectar las colisiones se divide en dos fases: fase amplia y fase estrecha (broad phase y narrrow phase del inglés). Durante la fase amplia se detectan pequeños grupos de objetos que es posible que se encuentren colisionando, mientras que en la fase estrecha se realiza una prueba para determinar si en realidad los objetos arrojados por la fase amplia están colisionando.

2.5.1. Fase amplia

Es la primera fase en la cual se divide el sistema detector de colisiones, la finalidad de esta fase es determinar cuáles son los objetos que existen en el entorno virtual que posiblemente estén colisionando. Para lograr esto se han utilizado distintas técnicas, entre las cuales, las más importantes son la división del espacio y el uso de volúmenes delimitadores (Figura 2.7), con los cuales de aquí en adelante se utilizara el termino bounding volume (BV) con el cual son comúnmente referidos en la literatura.

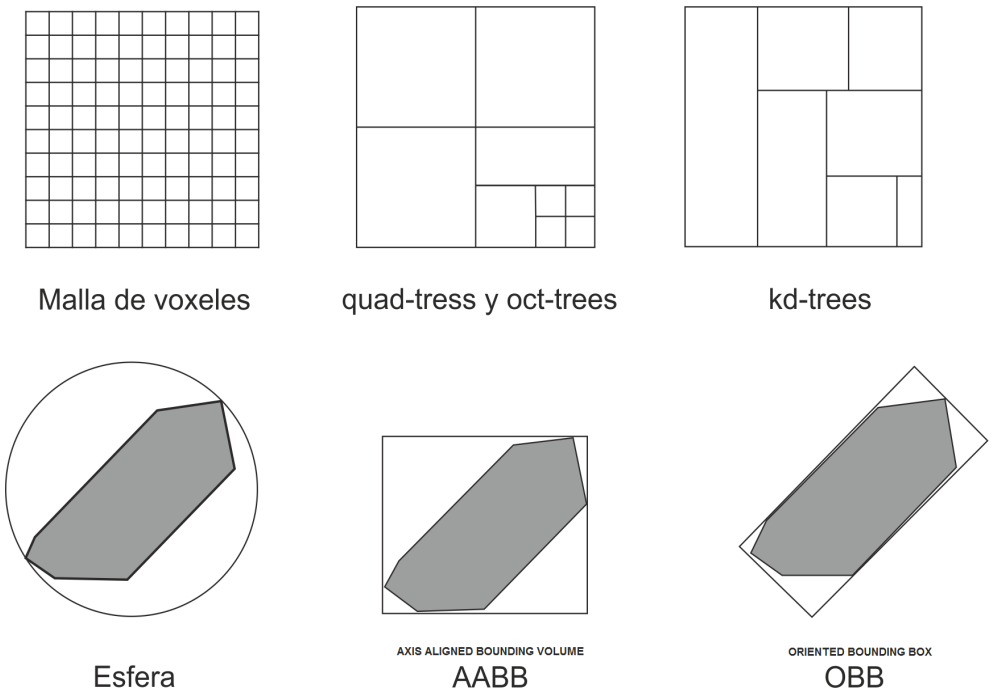

Figura 2.7: En la parte superior se encuentran las técnicas de división espacial que están centradas en dividir al entorno virtual, en la parte inferior se encargan de dividir el espacio a partir de los modelos que lo componen.

División del espacio

Existen distintas técnicas de división espacial. Entre las más conocidas se encuentran: cuadriculas uniformes [48], tablas hash jerárquicas [27], quadtrees, octrees [2], kd-trees y los espacios de partición binaria (BSP-trees) [30]. La principal desventaja de

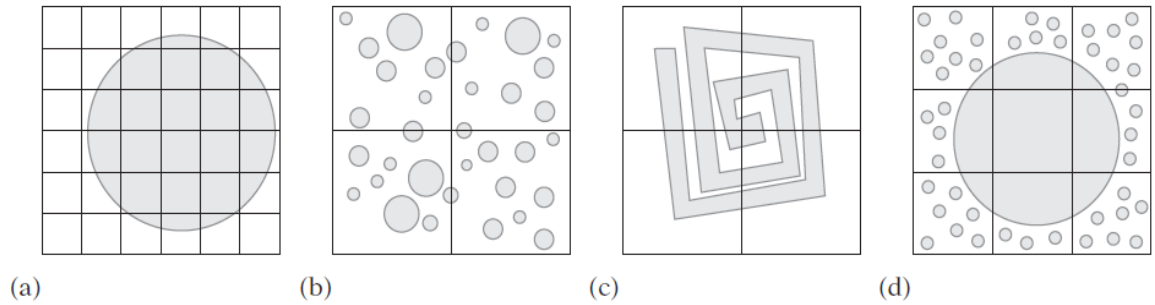

Figura 2.8: (a) La cuadricula es demasiado fina (b) La cuadricula es muy grande (con respecto a el tamaño del objeto) (c) La cuadricula es muy grande (con respecto a la complejidad del objeto) (d) La cuadricula es demasiado fina y grande a la vez. Tomado de [10].

este tipo de técnicas es su naturaleza estática, que necesita rehacerse o actualizarse cada vez que algún objeto cambia de posición en el espacio [46].

Cuadriculas uniformes

Una manera efectiva de dividir el espacio, es el uso de una cuadricula uniforme o malla de voxeles (Figura 2.9). Esta cuadricula divide el espacio en un determinado número de regiones o celdas, de un mismo tamaño. Posteriormente cada objeto se asocia con la celda en la cual se encuentra. Y solo si dos objetos comparten el espacio de una misma celda es posible que estos estén en contacto, más adelante se realizan pruebas a detalle para determinar si en realidad están en contacto o solo comparten el espacio de la celda. Mientras más esparcidos estén los objetos unos de los otros, menor es la posibilidad de que estos compartan la misma celda. Dependiendo de la complejidad de los objetos que existan en el mundo virtual, diferentes tipos de problemas pueden surgir en el tamaño de la división del espacio los cuales son:

1. La cuadrícula es demasiado fina. Si las celdas son muy pequeñas, un gran número de ellas se actualizarán para asociar la información del objeto que se

encuentra dentro. Y si los objetos están en movimiento, continuamente se tiene que localizar y actualizar las celdas que los contienen.

2. La cuadrícula es muy grande (con respecto al tamaño del objeto). Si las celdas son muy grandes y los objetos del mundo virtual muy pequeños con respecto a las celdas, es posible que una sola celda contenga información de varios objetos, con los que posteriormente se debe que determinar si existe una colisión entre ellos, lo que podría conllevar $O(n^2)$ pruebas para determinar cuáles objetos colisionan.

3. La cuadrícula es muy grande (con respecto a la complejidad del objeto). En este caso las celdas envuelven de manera correcta al objeto, pero este es demasiado complejo. Las celdas deberían ser de menor tamaño para realizar un mejor ajuste.

4. La cuadrícula es demasiado fina y grande a la vez. Esto es posible si existen objetos que varíen considerablemente de tamaño en el mundo virtual. Lo que provoca que las celdas sean demasiado grandes para los objetos de menor tamaño, o demasiado pequeñas para los objetos de mayor tamaño.

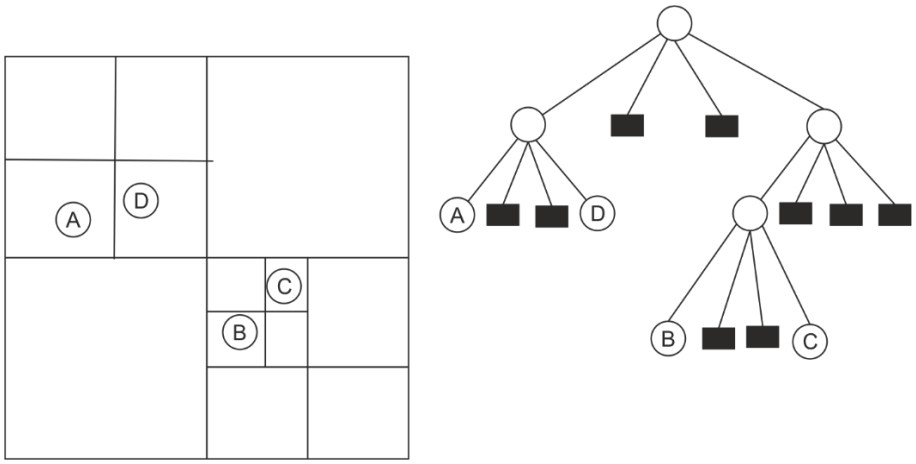

Figura 2.9: A la izquierda la división de un espacio utilizando quad-trees y a la derecha su representación en una jerarquía de árbol.

BSP-trees, Quadtrees, Octreees y Kd-trees

Existen otro tipo de técnicas las cuales dividen el espacio del mundo virtual en una jerarquía donde cada nodo padre tiene dos (BSP), cuatro (quadtree), ocho (octree) o k (kd-tree) nodos hijo, y cada nodo representa un determinado volumen del espacio total. El nodo raíz representa el espacio donde se encuentran todos los objetos existentes, posteriormente este volumen es subdividido en un dos, cuatro, ocho, o k celdas, los cuales representaran a los hijos de un determinado nodo padre y estos subvolúmenes serán divididos recursivamente de la misma manera (Figura 2.9). El criterio de parada para la generación de este tipo de jerarquía comúnmente es dado hasta que el árbol alcanza una determinada profundidad que el usuario especifica.

2.5.2. Fase estrecha

En un sistema de detección de colisiones, la fase amplia se caracteriza por encontrar los pares de objetos que posiblemente estén en contacto. En la fase estrecha se determina si en realidad los pares de objetos filtrados en la fase amplia están en contacto o no al aplicar pruebas a mayor detalle.

Un enfoque por fuerza bruta para determinar si un par de objetos están en contacto consiste en comparar todos los triángulos (en caso de que el objeto consista en una malla poligonal) contra todos los triángulos del otro objeto. Lo que conlleva a una técnica de complejidad cuadrática. Esto enfoque puede ser utilizado siempre y cuando el número de objetos a comprar sea pequeño y si los objetos están constituidos por una cantidad pequeña de triángulos, sin embargo, hoy en día los objetos modelados consisten miles de polígonos, por lo que este enfoque resulta prohibitivo.

La fase estrecha de un sistema de detección de colisiones puede dividirse en dos partes. Primero, las partes de un objeto que no se encuentren colisionando son eliminadas. Y segundo, se aplica una prueba con mayor precisión a los pares de polígonos que no fueron descartados en la primera etapa.

Mientras que en la fase amplia se realizó una división del espacio del mundo virtual, en esta fase, se aplican las técnicas de división a los objetos, creando jerarquías de estos que permitan un chequeo más eficiente en las colisiones entre ellos.

2.6. Bounding Volumes

Para determinar si existe una colisión entre la geometría de dos objetos dentro del entorno virtual, es necesario realizar pruebas que son sumamente costosas, sobre todo si el número de polígonos de la malla que compone al objeto es grande. Una manera de disminuir este costo, es el utilizar un BV que contengan a los objetos y solo si existe una colisión entre los BV proceder a realizar la prueba entre los objetos que estos contienen.

Un BV es un volumen simple que encapsula uno o muchos objetos complejos. La idea principal del uso de los BV es que estos sean objetos simples como esferas o cubos, los cuales permiten determinar de manera menos costosa si existe una colisión entre ellos. Esto resulta en una manera rápida de evaluar colisiones además de presentar un ahorro considerable de recursos computacionales.

Si bien es cierto, cuando existe una colisión entre dos objetos, el haber realizado una prueba anteriormente entre los BV resulta en un incremento en el tiempo computacional. Sin embargo, en la mayoría de las situaciones los objetos no siempre están en contacto, por lo que utilizar este tipo de técnica resulta en un aumento considerable en el desempeño de la aplicación, debido a que se conoce rápidamente cuando no existe un contacto, lo que justifica que algunas ocasiones se realicen un poco más de cálculos para determinar las colisiones existentes. Algunos de los BV más utilizados se pueden observar en la Figura 2.10. Cada uno de ellos cumple con la finalidad de delimitar los objetos del mundo virtual, además de ignorar rápidamente los objetos que no se encuentren en contacto con otro.

Dependiendo del BV utilizado se presentan ventajas y desventajas, que de mane-

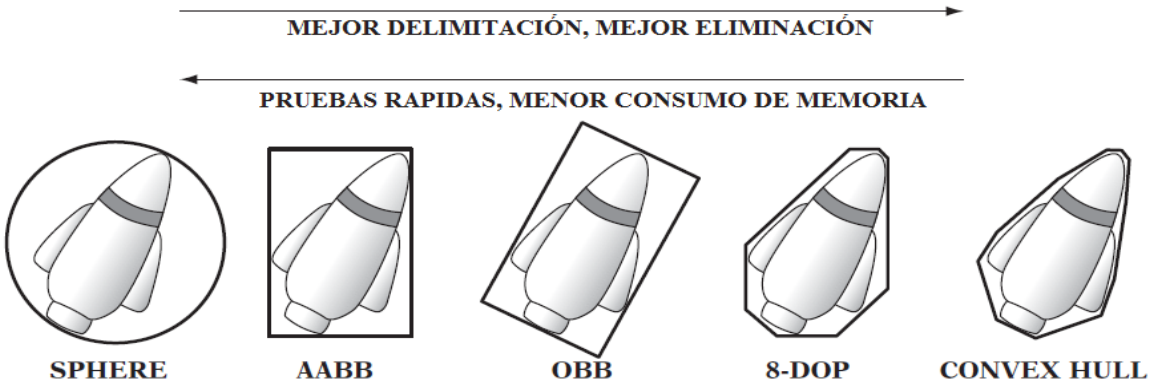

Figura 2.10: BV más utilizados ordenados de acuerdo a su ajuste y costo de pruebas de intersección. Basado en [10]

ra general consisten en costo computacional de las pruebas, la memoria utilizada por el BV, la delimitación del objeto, y su eficiencia para filtrar colisiones. A continuación se describirán algunos de los BV más utilizados, tales como esferas, Axis-aligned bounding box (AABB) y oriented bounding boxes (OBB), además de las ventajas y desventajas que estos presentan.

2.6.1. Axis-aligned bounding volumes

Este BV es uno de los más comunes. Es una caja rectangular de 6 o 4 lados (dependiendo de si el mundo virtual es en 3D o en 2d), su característica principal es que sus caras están orientadas de tal manera que sus normales siempre están paralelas con los ejes del sistema de coordenadas que se utiliza. La mayor ventaja de este BV es que la prueba para determinar si existe una colisión entre dos AABB's es muy rápida, debido a que comparan directamente las coordenadas individuales de cada BV.

Existen tres tipos de representaciones para este tipo de BV: la representación por mínimos y máximos la cual representa la región que existe entre dos de los puntos de las esquinas del BV; la representación por anchura mínima, la cual a partir de la

```
int TestAABBAABB(AABB a, AABB b)
{
    // Exit with no intersection if separated along an axis
    if (a.max[0] < b.min[0] || a.min[0] > b.max[0]) return 0;
    if (a.max[1] < b.min[1] || a.min[1] > b.max[1]) return 0;
    if (a.max[2] < b.min[2] || a.min[2] > b.max[2]) return 0;
    // Overlapping on all axes means AABBs are intersecting
    return 1;
}
```

Figura 2.11: Prueba de intersección para AABBs representados con mínimos y máximos.

esquina mínima del BV se determina la distancia de las aristas en cada uno de los ejes que la compone; por último la representación por punto central que determina el radio encada unos de los ejes.

Como se puede observar en la Figura 2.11 para mínimos máximos y la Figura 2.13 para representación central, la pruebas para determinar si dos AABBs intersectan son muy sencillas y su costo computacional es pequeño. Dos AABBs se traslapan si y solo si cada uno de sus ejes se intersecta.

Deducir cómo funciona el algoritmo para determinar si dos AABBs se traslapan es simple, si las coordenadas mínimas son menores a las máximas o si las coordenadas máximas son mayores a las mínimas en cualquiera de los ejes, se determina que no existe una intersección y se retorna una respuesta negativa, de lo contrario si todas las condiciones se cumplen existe una intersección entre los AABBs (Figura 2.12).

En el caso de representar los AABBs con su punto central, determinar si dos AABBs intersectan representa una mayor cantidad de cálculos, sin embargo la cantidad de memoria necesaria es menor. Determinar la intersección al igual que el algoritmo anterior es simple, basta con determinar la distancia absoluta desde el centro de ambos

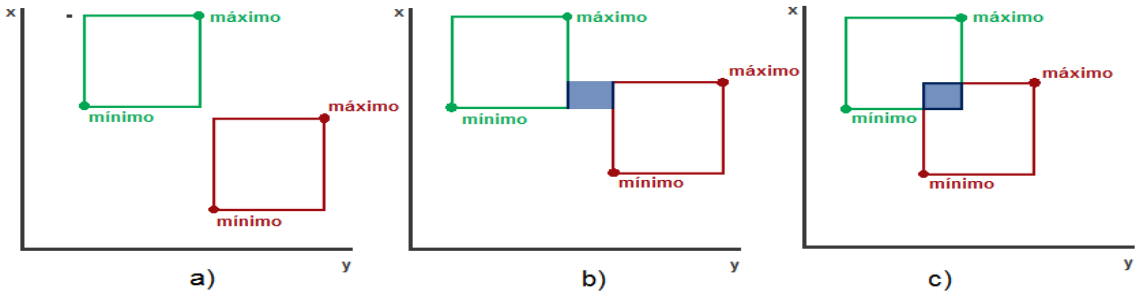

Figura 2.12: Representación en 2D de casos en las pruebas de intersección de AABB. a) Ninguno de los ejes se intersecta. b) Solo existe interseccion en uno de los ejes. c) Ambos ejes intersectan.

AABBs en cada uno de los ejes del sistema de coordenadas, si la distancia entre estos es mayor a la suma de los radios del eje que se está comparando, se garantiza que no existe una intersección, de lo contrario esta condición deberá cumplirse en todos los ejes para retornar una respuesta positiva.

```
int TestAABBAABB(AABB a, AABB b)
{
    if (Abs(a.c[0] - b.c[0]) > (a.r[0] + b.r[0])) return 0;
    if (Abs(a.c[1] - b.c[1]) > (a.r[1] + b.r[1])) return 0;
    if (Abs(a.c[2] - b.c[2]) > (a.r[2] + b.r[2])) return 0;
    return 1;
}
```

Figura 2.13: Prueba de intersección para AABBs con representación por punto central.

La diferencia en el costo computacional requerido en realizar ambas prueba es mínima, debido a que la operaciones que se realizan requieren de poco poder computacional, y aunque a primera vista la solución de mínimos máximos parece una mejor apuesta, es necesario tomar en cuenta que cuando los objetos están en movimiento, es necesario actualizar la información de cada uno de los BVs existentes, lo cual genera

una carga de cálculos adicionales. En un caso donde solamente ocurren traslaciones en los objetos, en el AABB representado con mínimos y máximos será necesario actualizar la información de todas las variables (6 actualizaciones en 3D, 4 en 2D), mientras que en la representación de punto central se requieren un menor número de cálculos al solo actualizar el valor del centro (3 en 3D, 2 en 2D).

AABB	MIN-MAX	Punto central
Sumas y restas	0	6
Comparaciones	6	3
Absoluto	0	3
Total	6	12
Translación		
Actualización	6 variables	3 variables

Tabla 2.1: Diferencias entre las representaciones de AABBs min-max y punto central.

En conclusión, a partir de los datos presentados en la Tabla 2.1, la elección de la representación del AABB dependerá en mayor medida del propósito de la aplicación a realizar. Si los objetos estarán en movimiento constantemente y la mayor parte del tiempo no existirá una intersección entre ellos, es conveniente utilizar la representación por punto central, de lo contrario, es recomendable el uso de AABBs mínimos máximos.

2.6.2. Esferas

Las esferas son un BV ampliamente utilizado, y como los AABB's, realizar una prueba de intersección tiene un bajo costo. Además las esferas tienen el beneficio de ser rotacionalmente invariante, lo que significa que independientemente de las traslaciones y rotaciones que ocurran en el objeto, la esfera siempre tendrá el mismo radio y lo único que se debe actualizar es la posición del centro de la misma.

En la Figura 2.14 se muestra la estructura con la que un BV de este tipo está

conformado, estando definido por las coordenadas del centro de la esfera y un valor correspondiente al radio de la misma.

```
struct Sphere {
    Point c;  // Sphere center
    float r;  // Sphere radius
};
```

Figura 2.14: Estructura para representar el BV de una esfera.

Las esferas son el BV más eficientes en términos de memoria utilizada, y la información del centro de este BV comúnmente está asociada al centro del objeto que la esfera envolverá. Sin embargo determinar el radio optimo que debe tener la esfera es complicado, para esto existes algoritmos especializados para dar un valor lo más pequeño posible.

```
int TestSphereSphere(Sphere a, Sphere b)
{
    // Calculate squared distance between centers
    Vector d = a.c - b.c;
    float dist2 = Dot(d, d);
    // Spheres intersect if squared distance is less than squared sum of radii
    float radiusSum = a.r + b.r;
    return dist2 <= radiusSum * radiusSum;
}
```

Figura 2.15: Prueba de intersección entre dos esferas.

La prueba para determinar la intersección entre dos esferas es simple. Es necesario determinar la distancia entre los centros de las esferas, dicha distancia también es conocido como distancia euclidiana o distancia ordinaria y se deduce a partir del teorema de Pitágoras aplicando la fórmula de la Ecuación 2.10. Con el fin de reducir el

costo computacional de dicha ecuación el problema se separa el dos partes, en primer lugar se realiza una resta de vectores y posteriormente se le aplica el producto punto o producto escalar para obtener la longitud al cuadrado de dicho vector se aplica la Ecuación 2.11 ó 2.12, de esta manera se evita realizar una operación costosa computacionalmente, la cual es obtener la raíz del valor obtenido. El resultado obtenido se compara con la suma de los radios al cuadrado y si este es menor o igual significa que existe contacto entre ambas esferas, de lo contrario se garantiza que no hay una colisión.

$$d_E(P,Q) = \sqrt{(P_1 - Q_1)^2 + (P_2 - Q_2)^2 + (P_3 - Q_3)^2} \tag{2.10}$$

$$a.b = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + ... + a_n b_n \tag{2.11}$$

$$a.b = |a.b^T| \tag{2.12}$$

Sin duda el uso de esferas como BV es una elección que permite un eficiente uso de memoria y una manera rápida de verificar si existen intersecciones entre dos objetos, sin embargo todo esto se ve contrarrestado por el poco ajuste que estos BVs ofrecen (Figura 2.16). Si bien en algunos casos estos pueden ofrecer un mejor ajuste en determinados modelos, generalmente es el BV con el ajuste más pobre de todos.

2.6.3. Oriented bounding volumes

Los OBBs son BVs parecidos a los AABBs, la diferencia principal entre estos es que el tamaño de este BV no cambia de acuerdo a las transformaciones que se realicen sobre el objeto que este contenga. Para realizar esto se utiliza una estructura que almacena el punto central, la matriz de rotación y la mitad de la anchura hacia cada uno de los ejes (Figura 2.17).

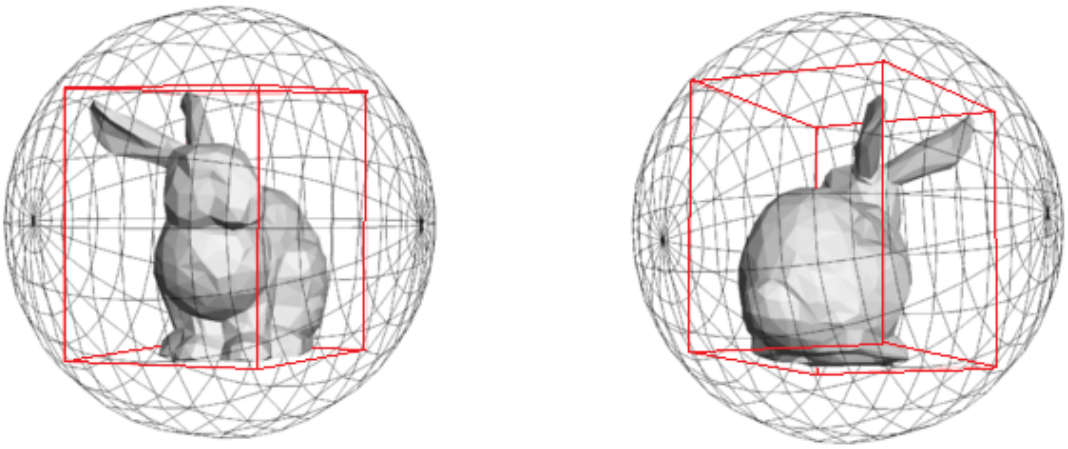

Figura 2.16: Comparación entre el ajuste de un AABB y una esfera.

```
struct OBB {
    Point c;      // OBB center point
    Vector u[3];  // Local x-, y-, and z-axes
    Vector e;     // Positive halfwidth extents of OBB along each axis
};
```

Figura 2.17: Estructura para representar un OBB.

El costo de memoria y el ajusto de cada OBB es mayor que el de los AABBs y esferas, sin embargo el costo de la prueba para determinar si existe una intersección entre dos OBBs es muy costosa.

Las pruebas para determinar una intersección entre dos AABBs y dos esferas son sencillas en comparación a la de los OBBs. Dados un OBB A y un OBB B de dos dimensiones, se comprueba si no existe una intersección entre estos cuando todos los vértices de A están fuera de los planos de B y viceversa, pero esta solución no aplica cuando los OBBs son de tres dimensiones. Para esto se utiliza un método llamado "prueba de ejes separados", que se puede observar en la Figura 2.18.

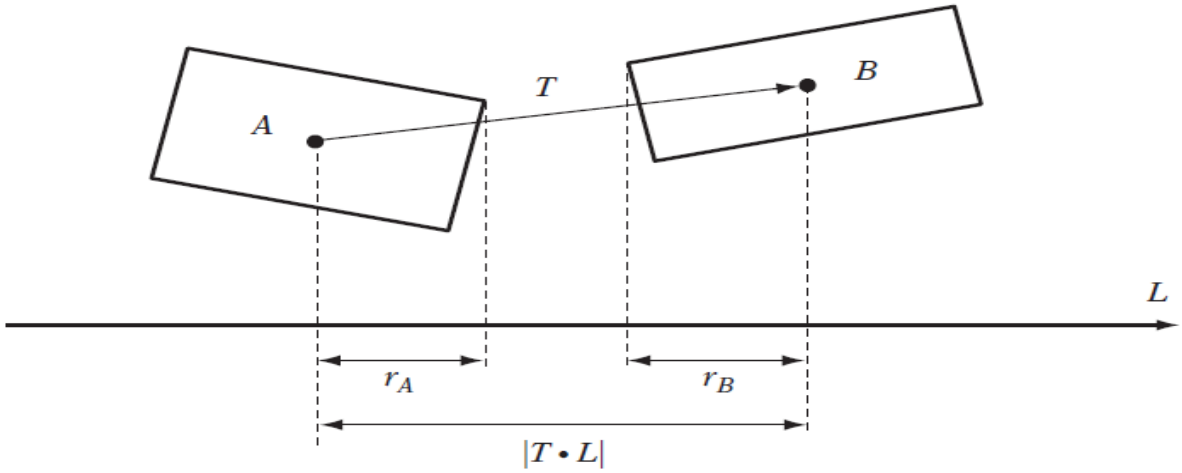

Figura 2.18: Dos OBBs se encuentras separados si la suma de las distancias de sus radios proyectados sobre un determinado eje L es menor que la distancia de la proyección de sus centros.

2.7. Jerarquías de bounding volumes

Al utilizar BVs como contenedores para los objetos que existes en el mundo virtual, se aumenta la rapidez con las que se detectan los pares de objetos en los cuales no existe una intersección, sin embargo, aunque las pruebas de intersección de han simplificado, el número de pruebas necesario para determinar los pares que se encuentran en contacto sigue siendo el mismo. Por lo tanto la complejidad de esta prueba sigue siendo la misma. Una solución a este problema es el uso de bounding volumes hierarchies (BVH), los cuales permiten reducir la complejidad del número de pruebas hasta *O(log n)*.

La jerarquía consiste en crear un árbol donde cada nodo es capaz de tener dos, cuatro, ocho o n hijos. Para explicar la generación del árbol se elabora una representación binaria para simplificar este proceso y que resulte simple de entender.

Todos los nodos hoja que contenga el árbol serán cada uno de los objetos que envuelven los BVs que se utilicen en el ambiente virtual. Estos nodos a su vez se agrupan en conjuntos pequeños que quedan envueltos en un BV más grande que permita

delimitar el espacio en el que se encuentra este nuevo conjunto de objetos, a su vez estos nuevo nodos serán agrupados de la misma forma de manera recursiva, donde eventualmente todos los objetos están agrupados en un solo BV que estará representado en la jerarquía creada como el nodo raíz (Figura 2.19).

Una de las ventajas principales de esta técnica, es que para determinar si un objeto colisiona con otro es necesario verificar si existe una intersección entre este y los nodos padres. Si esta prueba resulta negativa, ya no es necesario volver a realizar pruebas sobre los nodos que se encuentran en la misma rama en los niveles inferiores, ya que al no haber una intersección con el nodo padre, se garantiza que tampoco la hay con ninguno de los nodos hojas.

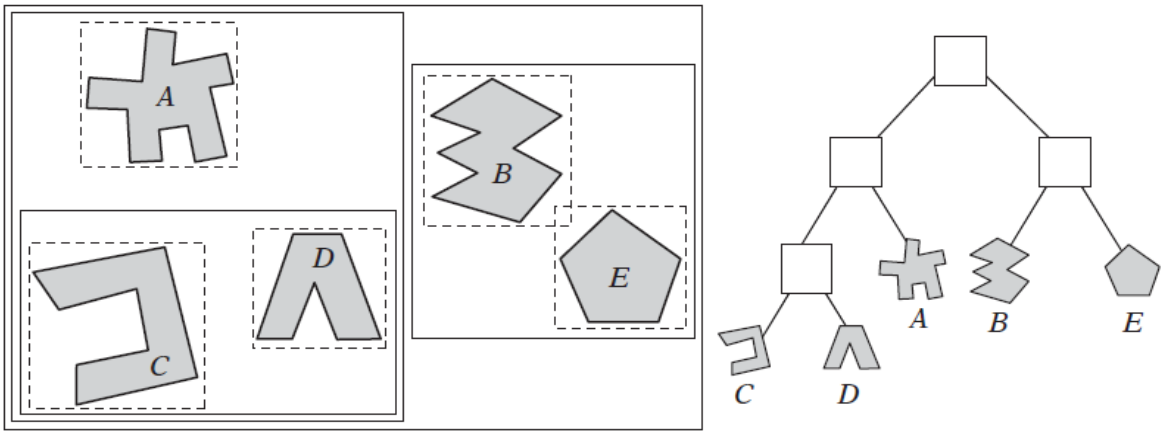

Figura 2.19: División del espacio a través del uso de BVs y su BVH correspondiente utilizando una jerarquía de árbol.

Al comparar los BVHs y las técnicas de división espacial descritas en la Figura 2.21, la principal diferencia es que en los BVHs un mismo espacio puede ser ocupado por distintos BVs, al contrario de como sucede en los esquemas de división espacial, donde cada celda ocupa un lugar predeterminado el cual no cambia con el paso del tiempo. Además los objetos en el BVH solo están contenidos en los nodos hoja de la jerarquía, a diferencia de la otra técnica mencionada donde un objeto puede estar

contenido en más de una celda, lo que implica que se encuentra en más de uno de los nodos de este esquema (Figura 2.20).

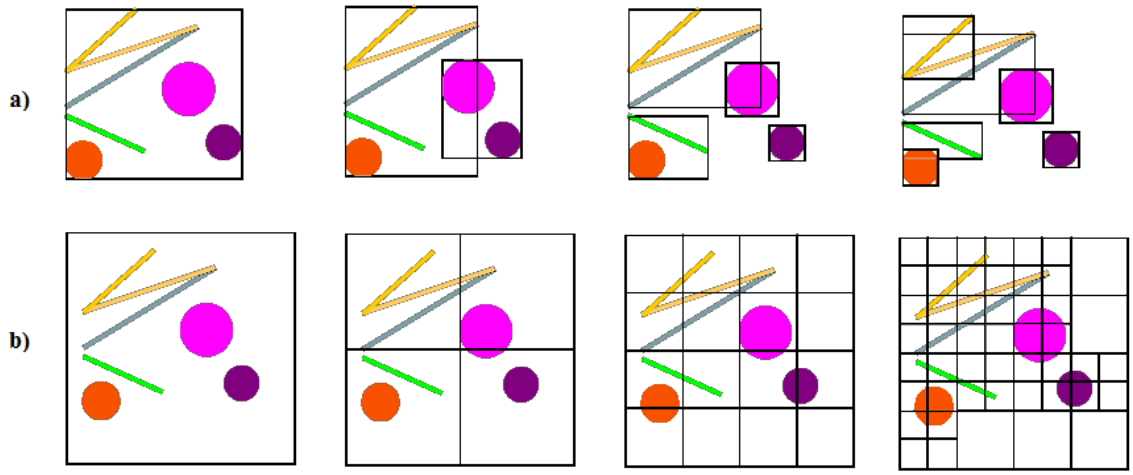

Figura 2.20: a) División del espacio utilizando BVs b) División del espacio utilizando quad-trees .

En la práctica, el uso de árboles binarios es el más común para la generación de los BVHs, debido a tres factores importantes: son simples de generar, su representación es sencilla, y recorrer este tipo de árboles no es complicado. El uso de árboles con mayor grado, tiene como ventaja que el recorrido que se tiene que realizar desde el nodo raíz hasta un nodo hoja es más corto, sin embargo el número de nodos visitados en cada nivel del árbol será mayor.

2.8. Construcción de bounding volumes

Dependiendo del número de objetos que se encuentren dentro del ambiente virtual, mayor será el número de árboles posibles a crear, si bien es posible crear una jerarquía manualmente, esta tarea resulta difícil y los resultados pueden ser insatisfactorios. Para lograr esto, se analizan los dos tipos de métodos más utilizados para generar el BVH de un ambiente virtual (Figura 2.21):

- Método descendiente (Top-down) : el método más popular para la construcción

de un BVH, se empieza por un BV que contenga a todos los objetos dentro del ambiente virtual y que será el nodo raíz de la jerarquía, posteriormente, el espacio es divido y se crean nuevos BVs que contengan cada uno de estos espacios, los cuales pasaran a ser los nodos hijos del nodo raíz, esta operación se repite recursivamente en los nodos generados hasta que en el espacio divido solo se encuentre un objeto, el cual será un nodo hoja de la jerarquía. La manera más simple de realizar la división del espacio es escoger el eje con mayor longitud y dividirlo a la mitad.

- Método ascendente (Bottom-up): este método empieza con los BVs de los nodos hoja y los une recursivamente hasta que solo se cree un nodo cuyo BV envuelva todo el ambiente virtual.

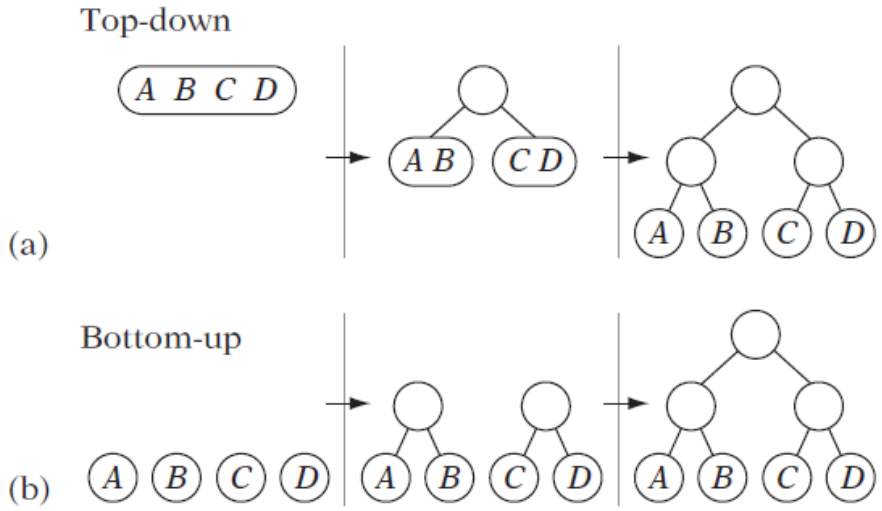

Figura 2.21: Métodos de generaciones de BVH.

2.9. Recorrido del árbol

Debe de existir una manera organizada para realizar el recorrido de los BVHs. Dos técnicas básicas para realizar esta tarea son: recorrido por anchura y el recorrido en profundidad Figura 2.22, donde el primero realiza un recorrido de todos los nodos de

un determinado nivel antes de pasar a un nivel con mayor profundidad, el segundo, en lugar de analizar los nodos del mismo nivel, prioriza analizar los nodos hijos del ultimo nodo revisado, haciendo esta rutina hasta llegar a un nodo hoja.

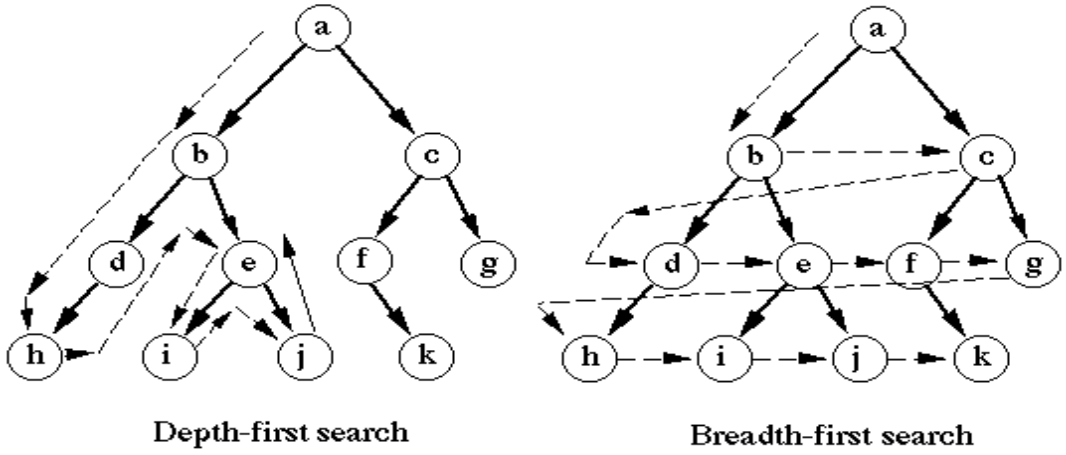

Figura 2.22: Distintos recorridos de un árbol.

Capítulo 3

Antecedentes

3.1. Sistemas de ensamble virtual

El término 'realidad virtual' (RV) fue adoptado por Jaron Lanier a finales de 1980, pero los orígenes de esta tecnología se remonta al trabajo de Ivan Sutherland en la computación interactiva y el uso de gafas de realidad virtual a mediados de 1960. Fue hasta 1970, que Sutherland y su equipo desarrollaron el primer sistema interactivo para gafas de realidad virtual operacional. A partir de este punto, se identificaron distintas aplicaciones que eventualmente dispararían el uso de esta tecnología a finales de 1980. Dichas aplicaciones se encuentran principalmente en el área del arte, simulación de vuelos, robótica, aplicaciones militares, e investigación espacial.

Myron Krueger fue la primera persona en explorar el potencial de los dispositivos para interactuar con la RV, la cual prefería llamar 'realidad artificial'. Fue a principios de 1970 cuando este creo distintas aplicaciones, en las cuales los usuarios podían interactuar con los ambientes creados por computadoras. La diferencia principal entre las aplicaciones creadas por Krueger y los sistemas de RV inmersivos, radica en que este no intentaba que los usuarios tuvieran la impresión de que su cuerpo estaba presente en el mundo virtual.

Además de identificar distintas áreas donde la realidad virtual podría ser de utilidad, un factor importante que logró que esta tecnología despegara, fue el rápido incremento en poder de computo asequible, requerido especialmente para la creación de gráficos por computadora, sobre todo en la creación de mundos realistas tridimensionales [40].

Dentro de las áreas en las cuales la realidad virtual representaba un papel importante se encuentra el ensamble virtual. Por sí solo, el ensamble se refiere al proceso de manipular y unir piezas para formar un todo [32], donde este todo puede ser una el producto final, o simplemente una pieza que lo compone. Dicho proceso suele llegar a representar hasta un 70 % del costo total de la producción de un producto [8], por lo cual fue necesario encontrar alternativas que redujeran este costo. La RV fue clave en este punto debido a que con el uso de esta tecnología se evita el uso de piezas físicas y estas se ven sustituidas por modelos virtuales que tienen sus mismas características, incluyendo mejoras como evitar el daño de las mismas por el uso constante y poder realizar cambios rápidamente a la pieza sin necesidad de crear nuevas físicamente. Existen muchos sistemas que permiten realizar tareas de ensamble virtual, a continuación se mencionan algunos de los sistemas de ensamble virtual más conocidos y de sus principales características.

Uno de los sistemas más populares para las tareas de ensamble virtual es VADE (Virtual Assembly Design Environment) [22], dicho sistema fue diseñado para poder utilizar diversos dispositivos de realidad virtual tales como Flock of Birds, CyberGlove™, y cascos de realidad virtual. Dentro de las principales características de este sistema se encuentran:

- La creación de ambientes realistas y la posibilidad de establecer un diseño personalizado en la configuración inicial del ambiente donde se llevara a cabo el ensamble.

- Es posible realizar operaciones de ensamble con una o dos manos.

- Las piezas utilizadas en el ensamble poseen un comportamiento físico.

- La información de las piezas para el ensamble es trasladada automáticamente a partir del sistema CAD al ambiente virtual.
- Simula de manera sencilla deslizamientos y rotación en las piezas a ensamblar a través de restricciones.
- Permite realizar modificaciones a las piezas en tiempo de ejecución.

Otro sistema popular es SHARP (System for Haptic Assembly and Realistic Prototyping), el cual utiliza modelos basados en física y permite la simulación realista de interacciones entre pieza-pieza y mano-pieza en el entorno virtual. Para la detección de colisiones y el modelado físico de las piezas utilizan Voxmap Point Shell (VPS) [26], debido a que es posible manejar piezas con geometrías complejas. VPS funciona correctamente cuando el número de piezas en el entorno virtual es pequeño y además esta optimizado para mantener la taza de actualización de fuerza háptica hasta los 1000Hz. Los modelos utilizados en SHARP, son discretizados en un conjunto de voxeles, los cuales son pequeños cubos que permiten la detección de colisiones y el cálculo de los comportamientos físicos. Para el cálculo de la fuerza de contacto entre dos piezas se utilizaron puntos que se encuentran en el centro de cada voxel que compone a la pieza, la cantidad de fuerza dependerá de la magnitud con la que este punto penetra dentro de los voxeles de la pieza a ensamblar.

Además de esto, cuando el usuario agarra un pieza con la mano virtual se crea un resorte-amortiguador entre esta y la pieza, lo cual provee al sistema de una retroalimentación háptica, la cual se calcula a partir de la suma de todas las fuerzas que se ejercen sobre la pieza. Estas son las fuerzas generadas por el resorte que une a la mano con la pieza y la fuerza ejercida por las posibles colisiones que existan con otras piezas.

Y como en muchas tareas de ensamble es necesario utilizar ambas manos, este sistema ofrece la capacidad de utilizar dos dispositivos con retroalimentación háptica para este propósito. Estas interfaces proveen una manera eficiente e intuitiva para interactuar con el entorno virtual [42].

VEDAP-II (Virtual Environment for Design and Assembly Planning) es otro sistema utilizado para las tareas de ensamble virtual, los modelos de este sistema son creados en SolidWorks™ y posteriormente son convertidos en mallas poligonales para su visualización en el entorno virtual utilizando la librería de gráficos OpenGL.

Este sistema consiste en un cuarto, con una mesa de trabajo y una mano virtual que representa la presencia del usuario dentro del entorno virtual, y las partes a ensamblar. Para rastrear los movimientos que el usuario realiza y representarlos dentro del entorno virtual se utiliza el dispositivo háptico CyberGlove™. Para la detección de colisiones de las piezas utiliza el algoritmo para piezas convexas V-CLIP, dichas piezas son creadas automáticamente por el software Virtual Hand Toolkit™ a partir de las piezas no convexas que se utilizan en el ensamble. Se utiliza el motor de física Physx™ [33]para otorgar un comportamiento dinámico a la piezas no convexas, y para la detección de colisiones de estas se utilizan técnicas de volúmenes delimitadores (Bounding Volumes). Para prevenir la interpenetración entre los modelos virtuales de las piezas a ensamblar, este sistema utiliza una técnica en la cual cada pieza es representada por dos modelos 3D: el modelo rastreado y el modelo visible, los cuales están unidos mediante de una serie de resortes-amortiguadores lineales y torsionales, con los cuales se pretende crear un comportamiento visual dinámico que permita al usuario percibir con mayor realismo el proceso de ensamble del producto. Al utilizar esta técnica los modelos visibles tienen un comportamiento dinámico utilizando Physx™ y permite visualizar con mayor realismo el comportamiento que de las piezas al interactuar con otras piezas del entorno virtual. Mientras que el modelo rastreado solo tienen un comportamiento cinemático y es seguido por el modelo visible. Cuando existe una colisión del modelo visible, este y el modelo rastreado se separan haciendo que el resorte que los une genere una fuerza que se utiliza para proporcionar una retroalimentación háptica y a su vez otorgar mayor realismo en la tarea de ensamble [14].

Un sistema más reciente es HAMMS (Haptic Assembly, Manufacturing and Machining System), el cual está compuesto por tres módulos:

- Módulo de visualización: encargado de renderizar los objetos virtuales y el

ambiente virtual.

- Módulo de simulación física: para simular el comportamiento dinámico de las piezas.
- Módulo de retroalimentación háptica: para proveer una retroalimentación de fuerzas, manipulación de las partes, detección de colisión y sensación de tanto al usuario.

Una característica de este sistema es que permite realizar una operación bimanual durante la tarea de ensamble, además es posible seleccionar que motor de física utilizar para el comportamiento dinámico de las piezas (Physx™ o Bullet Physics). El mayor aporte de este sistema es que los parámetros de simulación pueden ser modificados durante el tiempo de ejecución de la simulación [16].

3.2. Técnicas de ensamble

Una característica a tomar en cuenta dentro de los sistemas de ensamble virtual, es la técnica que se utiliza para la unión de las piezas, comúnmente se utilizan dos técnicas:

1. Collision Snapping.
2. Proximity Snapping.

Ambas técnicas propuestas en [9], consisten en la manipulación de dos piezas, la pieza primaria, la cual es la que el usuario agarra en el entorno virtual, y la pieza secundaria, la cual es el componente donde la pieza primaria se ensamblara. En el caso de la técnica "collision snapping" ambas piezas se unirán en el momento que se detecte que ocurre una colisión entre estas, lo cual es poco realista. En el caso de la técnica "proximity snapping", se ofrece una manera más refinada y realista para llevar a cabo el ensamble, la cual consiste el ensamble se llevara a cabo siempre y cuando la pieza primaria este lo suficientemente cerca a su posición final con respecto a la pieza secundaria. Sin embargo esta técnica no toma en cuenta algunas configu-

raciones de las piezas en las cuales su orientación puede ser errónea y de cualquier manera realiza el ensamble. Para solventar este detalle, Zaldivar [49] propone una representación vectorial de las partes a ensamblar, con ambos vectores teniendo una posición al origen y al extremo de estos, es necesario que los orígenes y los extremos de ambos vectores estén lo suficientemente cerca para llevar a cabo el ensamble.

Independientemente del software utilizado para el modelado de las piezas que se utilizan dentro de estos sistemas, existe un común denominador en un aspecto de estas y es que todos los sistemas utilizan modelos rígidos. La ventaja principal de estos modelos es que no requieren de mucho poder computacional para ser representados, así como asignarles un comportamiento dinámico a estos suele ser menos complicado. Sin embargo, en la vida real cuando un par de objetos entrar en contacto, estos sufren pequeñas deformaciones, que aunque en ocasiones son muy pequeñas, estas no dejan de ser significativas [35].

3.3. Modelos deformables

Para una representación más realista de los objetos, es posible utilizar modelos deformables, estos modelos fueron mencionados por primera vez por Terzopoulos en [43]. Estos modelos hacen posible representar cambios en los objetos a través del uso de métodos físicos principalmente, aunque también existen métodos no físicos que están basados en técnicas geométricas, estos no son tan utilizados ya que dependen altamente en la experiencia para modelar del diseñador [15]. Dentro de los modelos físicos, existen distintas técnicas como:

- Splines [39].
- Transformación libre.
- Modelos masa-resorte-amortiguador (MRA).
- Método de elemento finito (FEM).

Comúnmente un objeto deformable se define en su forma no deformada, también llamada configuración de equilibrio, reposo o forma inicial. Y por un conjunto de parámetros que define como este se deformara una vez que se apliquen fuerzas a dicho objeto [31].

FEM es un método para el modelado de objetos deformables utilizado comúnmente en el área de la ingeniería debido a que brinda resultados muy precisos, pero implica un costo computacional muy grande, una alternativa al FEM es el modelo MRA debido a la simplicidad que ofrece en el diseño de sus modelos, el bajo costo computacional que requiere en comparación al FEM y la rapidez que ofrece en sus simulaciones [6].

El modelo MRA amortiguador no solo es utilizado en el modelado de objetos deformables y se pudo observar con anterioridad en los sistemas de ensamble virtual [14], por lo que esta técnica puede tener un amplio campo de oportunidades de implementación.

El desarrollo de modelos deformables fue impulsado principalmente para su uso dentro del área del cómputo gráfico, sin embargo su uso se ha ampliado a otras áreas, el sistema MRA ha sido utilizado ampliamente en investigaciones médicas, de animación, para el modelado de cabello, modelado de tejido humano, modelado de ropa entre otros.

En [41] describen el proceso que se realizó para simular distintos tipos de cabello. La simulación del cabello es un fenómeno difícil de simular debido a la gran cantidad de cabello que existe en la cabeza y a la complejidad de su movimiento. En el modelo propuesto se utiliza el sistema MRA para el modelado de cada uno de los cabellos que se simulara. Ya que con MRA es difícil modelar torceduras, Selle implementando un nuevo modelo de resorte llamado resorte de altitud capaz de representar torsiones en el modelo. Debido a los cambios que implementa en los resortes del sistema MRA, es posible aplicar esta técnica en la simulación de ropa o en la conservación de forma para tetraedros.

En la medicina el uso de modelos deformables se ha aplicado en imágenes generadas por distintos medios tales como rayos x, tomografías, resonancias magnéticas y ultrasonidos. Modelos deformables en dos y tres dimensiones han sido utilizados para segmentar, visualizar, monitorear y cuantificar distintas estructuras anatómicas que varias desde escalas macroscópicas hasta microscópicas. En estos se incluyen el desarrollo de modelos que representen por ejemplo: el cerebro, corazón, cara, riñones, pulmones, cráneo, incluso objetos como tumores cerebrales, fetos, hasta estructuras celulares, neuronas o cromosomas [25].

Uno de los usos que se les ha dado a los modelos deformables, es para la representación de expresiones faciales. Esta tarea ha supuesto un gran reto, debido a que generar un modelo computacional que sea capaz de sintetizar los distintos matices de los movimientos faciales rápida y convincentemente. Si bien, el uso de estos modelos suele ser utilizado ampliamente en la animación de avatares virtuales, existen otros campos en los cuales es necesario este tipo de modelos, tales como cirugía plástica o investigaciones criminales. En este trabajo [44] se presenta una propuesta para el modelado de los músculos faciales al utilizar enrejados deformables los cuales se representaron como un modelo de 3 capas que representara el tejido facial. Estas capas representan al tejido cutáneo, tejido subcutáneo y al musculo. Una característica que distingue a cada una de las capas, es que la constante de rigidez en cada una de estas es distinto, con lo cual se logra representar de manera realista el comportamiento del tejido real.

En el trabajo [28] se presenta un modelo deformable que puede ser utilizado en cirugías virtuales. Dicho modelo es creado utilizando el sistema MRA. La idea principal de este trabajo es encontrar una estructura que permita representar el comportamiento del tejido de las cuerdas vocales. Se utilizan objetos en formas de barras con distintas densidades de masas en su malla y se aplican fuerzas de manera radial y perpendicular para observar el comportamiento de esta. En el primer caso se observa que la barra sufre una pequeña deformación, la cual aparenta que es doblada y en el segundo caso, la fuerza aplicada hace que la barra sufra una deformación mayor y

esta no pueda volver a su forma original. Además de estos experimentos se realizaron pruebas para simular la elasticidad y ruptura del tejido de la piel. A partir de estos experimentos se genera un nuevo modelo para representar las cuerdas vocales, al cual se le aplican deformaciones para que simule la el comportamiento de las cuerdas vocales reales. Se concluye que el uso del sistema MRA es propicio para simular este tipo de tejidos, ya que se observa que es posible imitar las deformaciones que sufren las cuerdas vocales en la realidad de manera similar.

Eriksson [11] evalúa la viabilidad de utilizar el sistema MRA para modelar tejido biológico, además de presentar un método para crear automáticamente objetos basados en este sistema a partir de imágenes médicas 3D.

La generación del modelo se da al tomar la información arrojada por la imagen médica 3D, la cual está representada por voxeles los cuales están definidos con un valor de intensidad. Para ajustar esta información al sistema MRA el método se divide en dos fases, la primera generando la cantidad de masas necesarias para representar el modelo (nube de puntos) y la segunda consiste en la generación de los resorte-amortiguadores que conectan la masas generadas en la primera fase.

En el área de simulación de ropa, en [38] se describe el desarrollo de un modelo basado en física, que implementa el sistema MRA, aplicando mejoras a este para tomar en cuanto las propiedades no elásticas de la tela tejida.

Si bien el uso de modelos deformables tiene aplicación es distintas áreas del conocimiento, el modelado de telas es uno de los más estudiados. Una tela es muy parecida a una superficie deformable compuesta de masas y resortes, sin embargo existen problemas cuando una gran cantidad de estrés se aplica en una determinada zona, lo cual provoca comportamientos no realistas en las simulación de la deformación de los textiles. Una de las soluciones más sencillas para solventar este problema, era incrementar la constante de rigidez de los resortes del modelo para generar un modelo con comportamiento menos elástico.

La estructura del modelo se basa en la creación de una malla de masas y resortes, y de la implementación de restricciones en los mismos. Se utilizan resortes con 2 finalidades, la primera es darle estructura a la malla, la segunda es darle mayor resistencia a cortes y dobleces en la tela para que la simulación sea más realista. El uso de restricciones es sencillo de implementar, ya que se debe establecer un límite de deformación en los resortes del modelo, por ejemplo, cuando la el valor de la restricción es 0.1, esto indica que los resortes no pueden superar el 10 % de su tamaño en reposo, y si bien este valor puede ser aún menor, esto ayuda a que la simulación de la tela sea más realista.

Para el modelado de modelos con características isotrópicas y anisotrópicas. En [4] se proponen dos tipos de estructuras para los modelos MRA, la primera es un tetraedro, el cual está compuesto por 4 masas y 6 resortes, y hexaedros compuestos por 8 masas, 12 resortes y 4 resortes de soporte. Para la conservación del volumen de los modelos se utiliza el baricentro de cada estructura aplicando una fórmula de fuerza cada vez que el volumen de las estructuras se vea afectado, logrando que la estructura recupere su forma original.

En este trabajo garantizan la anisotropía de los modelos al utilizar estas estructuras con diferentes parámetros para las variables de rigidez y de amortiguación, ya que comúnmente los objetos modelados con este sistema utilizan una sola malla con masas y resortes que conectan a las mismas, utilizando un valor contante para cada una de las variables de este. Por otro lado, al modelar el objeto a partir de las estructuras propuestas, es posible definir en qué áreas se requieren distintos parámetros, logrando con esto controlar con mayor eficiencia el comportamiento de este.

Los objetos modelados utilizando el sistema MRA están constituidos por un conjunto de masas y de resorte-amortiguador que conectan a estas masas, las cuales forman una malla poligonal. Independientemente de la forma del polígono, estos pueden fácilmente simplificarse hasta convertirse en triángulos. Dichos triángulos están constituidos por 3 vértices y 3 aristas los cuales representan a las masas y resortes respectivamente.

Un problema que se encuentra durante la simulación de los modelos MRA, es que los triángulos pueden colapsar al utilizar resortes longitudinales, provocando una mala simulación de las deformaciones. Esto se debe a que en algún momento de la simulación los resortes no generaran una fuerza que obligue a las masas conectadas a estos a cambiar de posición, o se produjo la suficiente fuerza para cambiar la orientación de los resortes completamente. El primer caso se da cuando el triángulo está en su estado de reposo, lo cual no afectaría al modelo, pero si afectaría cuando la fuerza producida por dos resortes se hace que estas se cancelen entre sí, lo que provocaría que la masa no sufriera ningún cambio en su posición. El segundo caso se da cuando una fuerza lo suficientemente grande se produce en algunos de los vértices lo cual produce que el triángulo cambie completamente de posición, y que en esta posición las longitudes de los resortes sean igual a la longitud de reposo de cada uno de ellos.

Una solución que se da a este problema es el uso de resortes no lineales, estos resortes a diferencia de los lineales, producen una fuerza diferente dependiendo de la longitud del mismo, lo cual previene que estos colapsen [6].

Los métodos más utilizados para el modelado de objetos deformables son aquellos basados en física. Sin embargo existen otros métodos como el propuesto por Müller [29] Este método ofrece una alternativa que radica en manipular directamente la posición de los objetos, omitiendo problemas como la acumulación de fuerzas o la inestabilidad de los modelos que presentan los métodos basados en física.

La dinámica basada en posición (PBD Position Based Dynamics en inglés) es muy parecido al sistema MRA, sin embargo PBD omite el cálculo de velocidades en las masas y la contante de amortiguador. Centrándose solamente en estados de equilibro o no equilibrio de las restricciones impuestas en el modelo, dichas restricciones son similares a los resortes que se utilizan en el sistema MRA. El estado de las restricciones dependerá de la distancia que existe entre dos puntos, al igual y como sucede con los resortes en el sistema MRA, si la longitud es menor o mayor a la de su estado inicial este generara una fuerza sobre las masas involucradas. En el caso de las

restricciones este paso de generación de fuerzas se omite y se calcula directamente la nueva posición que las masas tendrán.

Aunque este método no es capaz de simular con certeza situaciones reales, es muy utilizado debido a que en ocasiones no es necesario que las deformaciones de los modelos se comporten como en la realidad, solamente se necesita que sean sencillas de calcular y la visualización de lo suficientemente realista.

En [35] se propone un modelo que se beneficie de las ventajas de los modelos rígidos y deformables, llamado modelo quasi-rigido. Debido a que los modelos rígidos son los más utilizados hoy en día, a pesar de que estos no pueden reproducir deformaciones. Las razones principales por las cuales estos son utilizados, son debido a su eficiencia (bajo costo computacional) y que son simples de modelar. Además, en la realidad, todos los objetos son deformables hasta cierto punto, al existir un contacto con estos, se producen deformaciones que aunque pueden ser pequeñas son significativas, por lo cual es necesario modelar objetos que permitan representar deformaciones (modelo deformable). La desventaja de estos modelos es que suelen ser computacionalmente costosos, debido al continuo cálculo de la posición de las masas y de la fuerza que generan los resortes sobre las mismas, incluso cuando no existe interacción alguna con el modelo.

El modelo propuesto siempre trata a los objetos como modelos rígidos para que el costo computacional asociado a la representación de estos siempre sea mínimo. Cuando algún otro objeto entra en contacto con el modelo, se crean puntos de contacto en dicho modelo, estos puntos se convertirán posteriormente en pequeñas zonas deformables.

Una característica importante de estos modelos es que los objetos están representados mediante nubes de puntos, las cuales ofrecen un modelado rápido, y pueden manejar grandes áreas de contacto.

3.4. Detección de colisiones

Para detectar cuando ocurre una intersección entre dos objetos es necesario utilizar un buen sistema detector de colisiones, aunque existen distintos tipos de BVs y de jerarquías que pueden ser utilizadas, no existe una que supere en todos los aspectos a la otra, por lo que la implementación de alguna de ellas siempre está de la mano con la finalidad de la aplicación.

Entre los primeros surveys que hablan sobre detección de colisiones se encuentra [5] en el cual trata el problema más sencillo de detección de colisiones, el cual consta de determinar si ocurrirá o no una colisión dada una ruta predeterminada. Para resolver el problema de detección de colisión, se debe decidir si ocurrirá una colisión entre cualquier par de objetos de un determinado conjunto de objetos con formas y movimientos conocidos.

Se estudian tres métodos para la detección de choques: en el primero se toman muestras del movimiento un número finito de veces y se realiza una prueba de intersección en cada uno de ellos; en el segundo crean modelos de las formas y sus movimientos en el espacio-tiempo, buscando intersecciones entre las entidades 4D creadas; y en el tercero se crean modelos de los volúmenes de barrido de los objetos.

El crear algoritmos para la detección de colisiones entre objetos es un tema de estudio amplio. Uno de los grupos de trabajo que resaltan en esta área es el grupo GAMMA de la Universidad de Carolina del Norte, lo cuales han creado distintos sistemas detectores de colisiones para una variedad de problemas.

I-COLLIDE [5] es el primer método de detección de colisiones que presenta el grupo GAMMA, donde presentan un sistema capaz de detectar colisiones en ambientes complejos e interactivos. Uno de los atractivos principales de este método, es que se reduce las $O(N^2)$ posibles interacciones de n objetos en movimientos a $O(n+m)$ donde m es el número de objetos "muy cercanos" a un determinado objeto. En la práctica m suele ser un número pequeño y en el peor de los casos las interacciones

vuelven a ser $O(N^2)$. En dicho algoritmo se toman en cuenta tanto los objetos que se encuentran en movimiento como los que no, cada uno de los objetos está representado por su BV que en este caso son cubos de dos tipos: el primero tiene el tamaño suficiente para contener un objeto en cualquier orientación dada y el segundo cubo cambia de tamaño conforme el objeto se mueve. Una vez que se detecta cuáles son los objetos "muy cercanos" se determina cuáles son los BVs que se encuentran en contacto, al final se determina si en realidad existe una colisión entre ambos objetos.

Una de las limitantes de este algoritmo, es que solo puede detectar colisiones entre poliedros convexos. Para tratar a los objetos no convexos en el entorno virtual, es necesario descomponerlos en un conjunto de objetos convexos para el correcto funcionamiento de este.

Dos años después, los creadores de I-COLLIDE presentan un nuevo algoritmo llamado V-COLLIDE, en el cual ya no es necesario descomponer los objetos no convexos, y requiere que los objetos que se utilicen estén compuestos por mallas triangulares, además de permitir agregar o eliminar dinámicamente objetos dentro del entorno.

Otro algoritmo del grupo GAMMA es RAPID (Robust and Accurate Polygon Interference Detection) [17], este algoritmo permite detectar colisiones de manera eficiente a través del uso de OBBs en cualquier malla poligonal (triángulos). Se pre computan las representaciones jerárquicas de cada uno de los objetos utilizando OBBs. Mientras que en tiempo de ejecución, el algoritmo recorre los árboles y realiza una prueba de intersección entre los OBBs utilizando el teorema de los ejes separados (Separated Axis Theorem) para determinar si esta ocurre, además de comparar su rendimiento en comparación de otras estructuras de datos jerárquicas. La razón principal por la cual se utilizan los OBBs como BV de este algoritmo es debido a que el ajuste de estos BVs es mejor que el ajuste de los AABB y de las esferas, por lo tanto suele ser necesario realizar un recorrido menor en los niveles del árbol para determinar si dos objetos colisionan.

Para determinar colisiones, el algoritmo este algoritmo es más rápido asintótica-

mente que aquellos basados en AABB y esferas en situaciones donde los objetos se encuentran próximos los unos a los otros, también se puede detectar la colisión exacta entre dos objetos ya que este detecto cuales son los pares de triángulos que se encuentran en contacto sin importar la complejidad geométrica, ni el tamaño de los triángulos.

PQP (Proximity Query Package) [23] es una versión actualizada de RAPID, en esta versión se permite calcular la distancia aproximada y la distancia exacta entre dos objetos.

PQP es un algoritmo para determinar la proximidad entre dos objetos de manera rápida implementando una nueva familia de BVs que denominan Esferas de barrido (Swept spheres). La ventaja principal de estos BVs es que es posible otorgar distintos ajustes en los objetos que estos representan. Dicha familia la dividen en tres posibles volúmenes (Figura 3.1): el primero es la esfera de barrido de punto que representa el mismo volumen que el de una esfera: la segunda son las líneas de esferas de barrido cuyo volumen es obtenido al representar el espacio el cual ocupa la esfera al realizar un recorrido a través de una línea o a través de un rectángulo donde pasan a ser llamadas rectángulos de esferas de barrido.

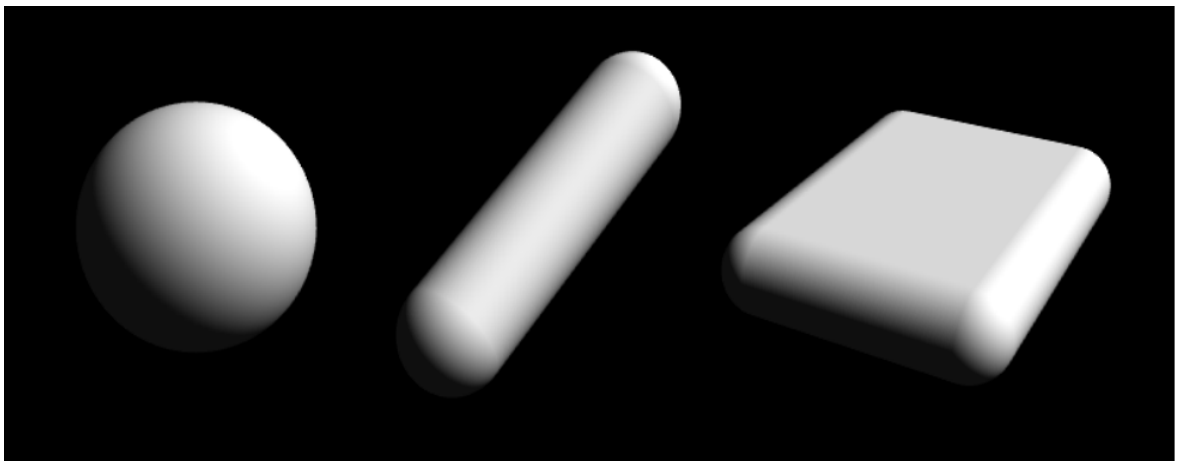

Figura 3.1: Diferentes tipos de esferas de barrido; el BV de la izquierda es una esfera, el del centro una línea de esfera de barrido, y el de la derecha un rectángulo de esfera de barrido tomada de [23].

Una característica común de los algoritmos mencionados hasta ahora es que todos los cálculos realizados por estos son a través del uso del CPU y están dirigidos a detectar colisiones entre modelos rígidos, aunque RAPID y PQP pueden ser adaptados para soportar modelos deformables, el rendimiento de los algoritmos con este tipo de modelos no es óptimo.

CULLIDE [18] es el primer algoritmo para detectar colisiones entre objetos deformables creado por el grupo GAMMA. Uno de los atractivos de este nuevo algoritmo es que utilizan GPU para realizar gran parte de los cálculos para determinar colisiones. Esto lo hacen al crear conjuntos que posiblemente colisiones (PCS), dichos conjuntos son creados a partir de realizar consultas de visibilidad calculada con la tarjeta gráfica. Dichos conjuntos están compuesto por todos los pares de triangulo que se encuentran colisionando o que posiblemente están colisionando. Para determinar la colisión exacta entre los triángulos, realizan los cálculos necesarios utilizando CPU.

Dentro de las ventajas de este algoritmo se encuentran que funciona con cualquier objeto que esté compuesto por una malla poligonal (triángulos), además de esto no necesita tiempo de pre computo ni estructuras de datos adicionales para representar a los objetos, por lo que su uso de memoria es menor a otros algoritmos.

Las jerarquías de cada modelo que existe en el mundo virtual se crean en una etapa de pre procesamiento de los datos. Mientras que en tiempo de ejecución se realizan pruebas sobre los posibles pares que se encuentran colisionando al hacer un recorrido de ambos árboles, empezando por el nodo raíz de cada una de las jerarquías de los objetos, si resulta que los BVs existe una intersección entre los nodos raíz de ambas jerarquías, este proceso se repite en los nodos hijos de la raíz y así sucesivamente hasta llegar a un nodo hoja (Figura 3.2), donde se realiza una prueba precisa para determinar una intersección entre ambos polígonos.

Los BVs más comunes para la generación de estas jerarquías son las esferas [20], los AABBs [36] y los OBBs [1]. Sin embargo cada BV posee ventajas y desventajas que son necesarias tomar en cuenta para su buena elección. En el caso de las esfe-

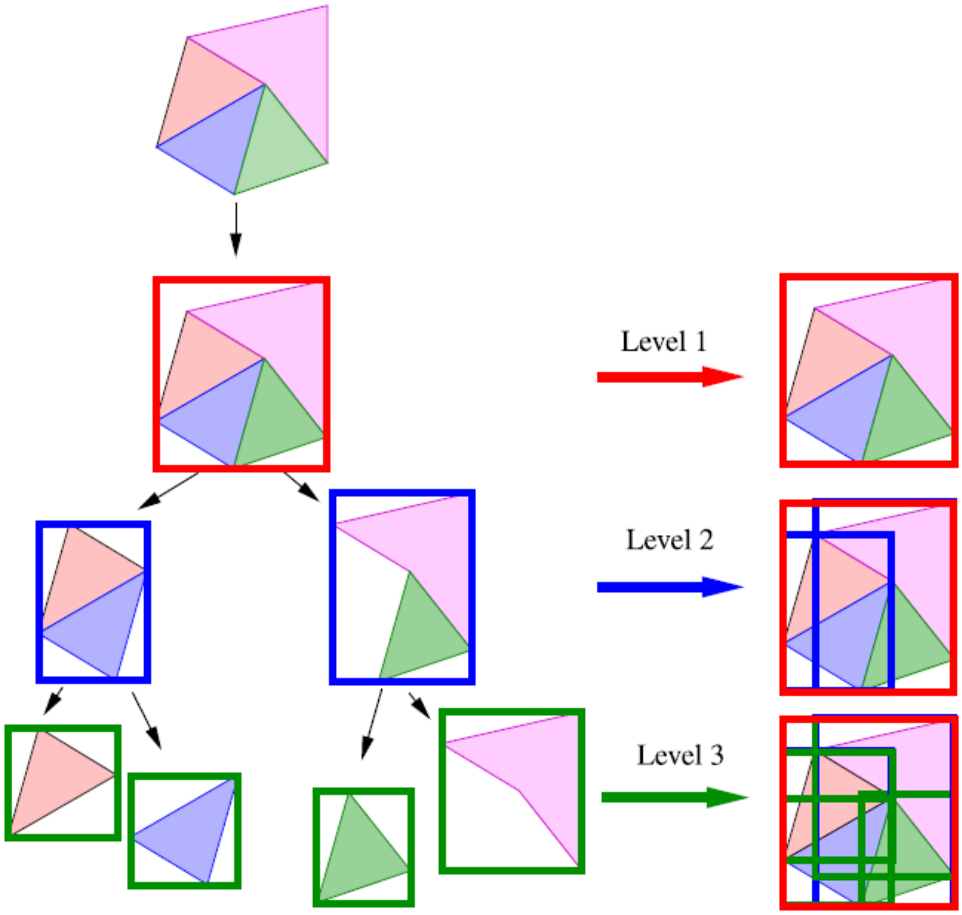

Figura 3.2: Descomposición de un modelo para crear su BVH.

ras, estas poseen la prueba de intersección más rápida de todos los BVs, así como un manejo eficiente de memoria y su principal ventaja es que son rotacionalmente invariables, pero el ajuste que tiene en los objetos es pobre, sobre todo si se trata de objetos planos. Los AABBs cuentan con el manejo de memoria más eficiente de todos los BVs, además de un mejor ajuste que las esferas, sin embargo, es necesario actualizar constantemente la información de estos BVs debido a que cualquier rotación en el objeto puede provocar grandes cambios en el tamaño de este BV, sobre todo cuando son objetos complejos. Por último los OBBs, que solventan los problemas de los AABBs a cambio de un mayor uso de memoria para su representación y pruebas

de intersección computacionalmente más costosas.

Weghorst et al. [45] proponen una función de costo para el análisis de métodos jerárquicos utilizando trazado de rayos, posteriormente, esta misma función fue retomada por Gottschalk et al. [17] Adaptándola para los métodos de detección de colisiones utilizando jerarquías (ver Ecuación 8).

3.5. Análisis de resultados

Los sistemas de ensamble virtual necesitan trabajar en tiempo real, es decir, necesitan que las interacciones que ocurren entre el usuario y los elementos del sistema sucedan de manera fluida y de la manera más realista posible. Es por esta razón que el uso de modelos 3D rígidos predomina en estas aplicaciones, debido a que en este tipo de modelos no se necesitan demasiados recursos computacionales para su llevar a cabo su representación.

El ensamble de dos piezas es un fenómeno difícil de representar debido a la complejidad que puede existir para llevarse a cabo, para lograr esto se utilizan técnicas que simplifican esta etapa de la fase de ensamble. El ensamble por contacto o por proximidad, son algunas de las técnicas que utilizadas en esta etapa, sin embargo, el realismo que estas ofrecen suele ser muy pobre, principalmente debido a los cambios repentinos que sufren las partes para llegar a su posición final de ensamble. Es por esto que el uso de técnicas como el snap-fitting mejorado permite aumentar el realismo sin incrementar el costo computacional de estas.

Dentro de los modelos deformables, predominan los creados a partir del sistema masa-resorte-amortiguador y FEM para distintas aplicaciones. FEM otorga resultados precisos en las deformaciones de los objetos, pero el número de cálculos necesarios para representar las deformaciones suele ser excesivo, lo cual implica una perdida en el rendimiento de la aplicación, si a esto se le agrega un modelo de gran tamaño, el número de cálculos a realizar resulta inviable para su aplicación en los sistemas

de ensamble virtual. El modelo masa-resorte-amortiguador ofrece un realismo menor que los objetos modelados con FEM, pero en cambio la rapidez con la que se calculan las deformaciones de este tipo de modelos es superior a la del anteriormente mencionado. Aunque el cálculo de estos modelos es mas rápido, si es necesario representar un modelo cuyos número de masas y resortes y muy elevado, el tiempo necesario para representar los cambios en el modelo también aumenta, es por esto que el uso de modelos deformables en los sistemas de ensamble virtual no es común.

Detectar si existe una colisión entre dos partes es una tarea fundamental en los sistemas de ensamble virtual, ya que en la realidad los objetos no pueden ocupar el mismo espacio al mismo tiempo debido a la impenetrabilidad. Una manera sencilla y rápida de atender esta necesidad es utilizar los detectores de colisiones que ofrecen los motores de física existentes, tales como PhysX^{TM}, ODE, Havok [3], entre otros.

A partir de lo anterior, se concluye que la mayoría de los sistemas de ensamble virtual utilizan modelos rígidos para representar a las partes del producto a ensamblar, sin embargo, uno de los problemas en estos sistemas es que una gran variedad de estructuras mecánicas están conformadas a partir de materiales deformables, tales como las llantas de caucho de una bicicleta. Es por esto que el uso de modelos rígidos no representa un acercamiento lo suficientemente realista al momento de realizar las tareas de ensamble. Lo anterior puede mejorar con el uso de modelos deformables, pero esta solución es rápidamente descartada debido al costo computacional necesario de estos modelos.

En este trabajo se presenta una alternativa a los modelos deformables, la cual es el desarrollo de un nuevo modelo hibrido que mientras este sea manipulado por el usuario y no exista alguna interacción con otros objetos se comportara como un modelo rígido, de lo contrario el modelo será capaz de representar deformaciones en aquellas regiones cercanas a los puntos de contacto con los otros objetos. Este modelo se denomina modelo rígido-deformable, y al permitir representar deformaciones en regiones determinadas del modelo, el número de cálculos necesarios de este se ve reducido, lo que permitiría implementar este tipo de modelos en tareas de ensamble

virtual.

Debido a que el modelo que se propone en este trabajo es nuevo, el comportamiento dinámico del mismo no es capaz de representarse en los actuales motores de física, por lo anterior se desarrolla un sistema detector de colisiones para modelos de este tipo. Este sistema detector de colisiones permite que estos modelos puedan interactuar entre sí, y proporciona la información necesaria para realizar las deformaciones solo en las regiones de interés.

Capítulo 4

Desarrollo del modelo rígido-deformable

En este capítulo se describe el proceso del desarrollo del modelo rígido deformable. Este proceso de dividió en 4 etapas que van desde el desarrollo de un modelo deformable hasta la implementación de los modelos rígido-deformables. En la primera etapa se implementa un modelo basado en el sistema masa-resorte amortiguador y se determina cual es la región de deformación del objeto a partir de aplicar una fuerza en alguna zona de este. Durante la segunda etapa se detalla el desarrollo del modelo rígido-deformable, explicando la conformación de vecindades de masas, resortes y caras del modelo, además de la implementación de niveles de profundidad de deformación en el mismo. El desarrollo de un sistema de detección de colisiones para los modelos propuestos fue desarrollado en la tercera etapa, por último, en la cuarta etapa se llevan a cabo tareas de ensamble virtual con los modelos propuestos.

4.1. Carga del modelo deformable utilizando el sistema masa-resorte-amortiguador

En esta etapa se realiza la carga de los modelos rígidos a partir de un archivo, posteriormente se realiza una conversiones de estos modelos para transfórmalos en modelos deformables utilizando en el sistema masa-resorte-amortiguador. Además de esto, se propone una solución al problema de la restauración de forma de los modelos utilizados. Por último, se determina cual es la región que presenta una mayor deformación al aplicar una fuerza al modelo.

4.1.1. Arquitectura de software

La arquitectura de software utilizada para el desarrollo de esta etapa se muestra en la Figura 4.1.

El sistema fue escrito en el lenguaje de programación C++, haciendo uso de la

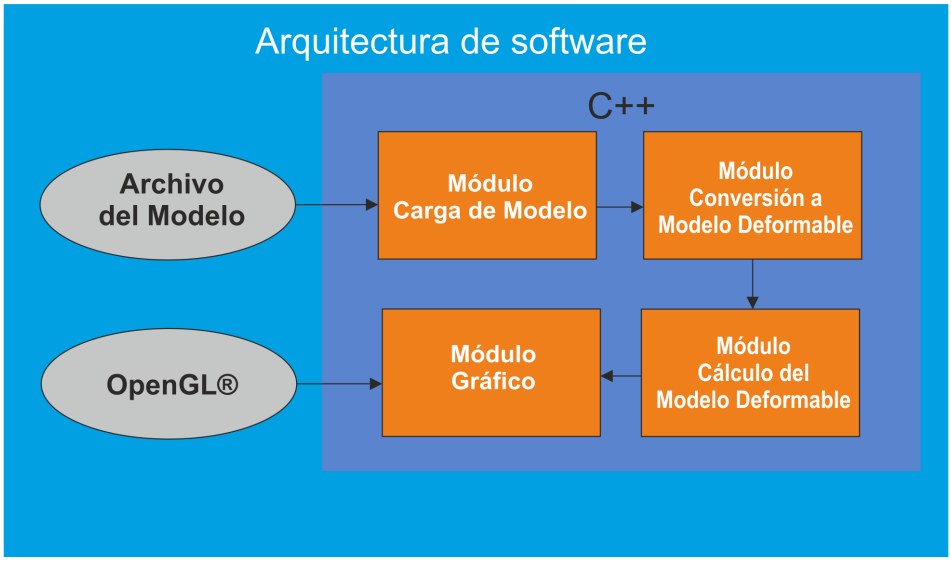

Figura 4.1: Arquitectura de software de la etapa 1.

plataforma Microsoft® Visual Studio® 2010. El módulo de carga del modelo es

el encargado de almacenar la información necesaria para representar a la pieza que se desea mostrar en el entorno virtual, la descripción de dicha pieza se encuentra en un archivo con extensión ".tri". Una vez obtenida la información necesaria para representar la pieza, esta pasa por el modelo de conversión para obtener un modelo deformable a partir de la misma. Posteriormente dentro del módulo de cálculo del modelo deformable se determina cuáles son las fuerzas que se ejercen sobre este, lo que posteriormente permite reflejar los cambios en el modelo durante la renderización de los mismos en el módulo gráfico en el cual se utiliza OpenGL.

4.1.2. Carga del modelo rígido

Para la carga del modelo rígido, se lleva a cabo el análisis de un archivo con extensión ".tri", dicho archivo contiene la información necesaria para generar un modelo rígido y la estructura de este es la siguiente:

- Nombre del modelo
- Numero de vértices
- Numero de caras
- Coordenadas de cada uno de los vértices
- Vértices que conforman cada una de las caras

A partir de esta estructura, se crean las variables necesarias para almacenar la información. Tanto el número de vértices, como el número de caras son almacenados en variables de tipo entero, las coordenadas de los vértices se almacenan en un vector de datos de tipo flotantes, y los vértices que conforman cada una de las caras son almacenados en un vector de datos de tipo enteros.

4.1.3. Generación del modelo deformable

La generación del modelo deformable se a partir de los datos obtenidos de la carga del objeto dado en el modelo ".tri". El modelo a crear está basado en el sistema MRA cuyos componentes principales son las masas y los resortes. Para la creación del modelo deformable se divide el proceso en dos fases: la primera es la generación de las masas, y en la segunda se crean todos los resortes que compondrán al modelo. Cada una de las masas del modelo es representada por una instancia a la clase Mass que está estructurada como se muestra en el código 4.1.

Código 4.1: Estructura de la clase Mass

```
1   class Mass{
2       public:
3           float mass;
4           Vector3D position;
5           Vector3D velocity;
6           Vector3D force;
7
8           int anchor;
9           Vector3D fixedPosition;
10  }
```

Cada instancia a esta clase almacena la masa en kilogramos de cada masa del modelo en la variable mass.Position almacena la posición donde se encuentra en un determinado momento la masa en una coordenada (x,y,z). Velocity es una variable que guarda el vector de velocidad que tiene la masa .Force almacena que se le aplique a esta. Las variables anchor y fixedPosition se detallaran en la Figura 4.22, ya que estas se utilizan en el modelo rígido-deformable.

En el caso de las variables tipo vector3D son un tipo de variable que almacena 3 valores de tipo flotantes y que además permiten realizar distintas operaciones de vectores de manera rápida (código 4.2).

Código 4.2: Estructura de la clase Vector3D

```
1    class Vector3D
2    {
3        public:
4            float x;     // the x value of this Vector3D
5            float y;     // the y value of this Vector3D
6            float z;     // the z value of this Vector3D
7    }
```

El primer paso se centra en la generación de las masas, el número de masas a crear dependerá de la cantidad de vértices por el que está compuesto el objeto cargado desde el archivo ".tri", con lo cual se procede a la creación de las N instancias a la clase Mass. Usualmente el valor para la variable mass es 1, sin embargo este valor puede ser alterado en cualquier momento. Una vez creados todas las instancias, es necesario capturar la posición de en la que se encuentran cada una de ellas a partir de la posición de los vértices que se encuentran en el archivo, por lo que se realiza un recorrido desde el vértice 0 hasta el vértice N y se realiza una copia del valor de la posición de los ejes 'x', 'y' y 'z' en la variable position de la masa correspondiente (código 4.3). Las variables restantes de la masa son inicializadas con valores 0.

Código 4.3: Estructura repetitiva implementada para almacenar la posición de las masas

```
1    for ( int i=0; i < numOfMasses ; i++)
2    {
3        masses[i]->position.x= vertex[i][0]; //x
4        masses[i]->position.y= vertex[i][1]; //y
5        masses[i]->position.z= vertex[i][2]; //z
6    }
```

La segunda fase que consiste en la creación de los resortes no es tan directa como la generación de las masas, ya que la información necesaria para elaborarlos no se encuentra de manera explícita en el archivo ".tri", pero es posible obtenerla a partir

del análisis del mismo. Los atributos necesarios del modelo se encuentras en la clase Spring (código 4.4), consta de dos masas las cuales son las que están asociadas al resorte, la longitud del resorte y las constantes que definen su rigidez (stiffness) y su amortiguado (damping).

Código 4.4: Estructura de la clase Spring (resortes)

```
1   class Spring
2   {
3       public:
4           Mass* mass1
5           Mass* mass2;
6
7           float springConstant;
8           float springLength;
9           float frictionConstant;
10  }
```

Una de las dificultades para establecer cuáles son los resortes que componen al modelo, es encontrar todas las posibles combinaciones de resortes en cada una de las caras y almacenar todos los resortes únicos que existan en el modelo. Debido a que en la descripción del archivo solo se menciona cuáles son los vértices que componen cada cara, es posible que caras distintas compartan un mismo par de vértices. Esto genera un conflicto al momento de generar los resortes, por lo que es necesario determinar cuáles son estos pares de vértices que se repiten y crear solamente un resorte a partir de estos (Figura 4.2).

Otro dato que no se encuentra especificado en el archivo ".tri" es la longitud de los resortes, pero obtener esta información es simple ya que hasta este punto se conocen las posiciones de las masas y cuáles son las masas asociadas a cada resorte. Por lo tanto para obtener la longitud del resorte se logra al calcular la distancia euclidiana entre las coordenadas de la posición de las masas.

Realizado el filtrado de los resortes duplicados, es posible llevar a cabo el proceso de creación de los resortes, en los cuales se necesitan especificar cuáles son los

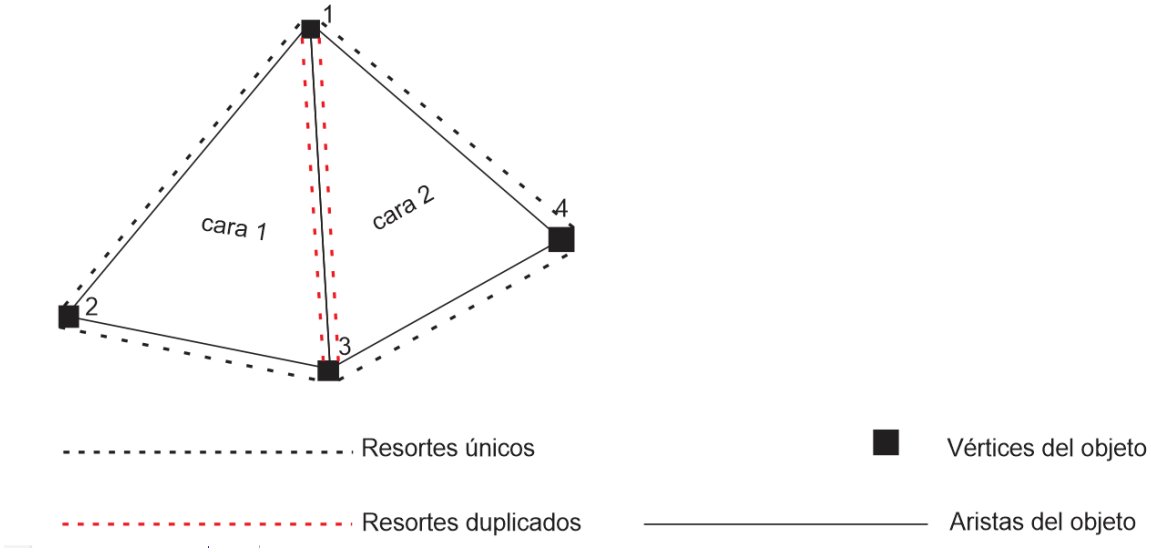

Figura 4.2: Conformación del modelo MRA en un objeto simple.

sus masas asociadas m_a y m_b, la longitud del resorte, y su valor para las contantes de rigidez y amortiguación, estos últimos valores no es necesario calcularlos y simplemente se asignan los que resulten convenientes para la experimentación.

4.1.4. Interacción del modelo deformable

Una vez creado el modelo deformable, la primera tarea a realizar al interactuar con el modelo es observar el comportamiento del modelo al aplicar alguna fuerza y su rendimiento en el equipo de cómputo utilizado. Para la interacción con el modelo deformable, se utilizó el teclado de la computadora, con el cual se seleccionan una o más masas del modelo y se les aplica una fuerza en alguna dirección en concreto. Uno de los problemas encontrados en esta etapa del desarrollo de los modelos deformables, es que al aplicar una fuerza lo suficientemente grande estos perdían su forma y resultaba imposible retornarlo al estado inicial del modelo (Figura 4.3).

Para solucionar este problema se establecieron un conjunto nuevo de masas y resortes, lo cuales conforman un nuevo modelo el cual es denominado “modelo invisible”, este nuevo modelo es una copia del modelo deformable que de momento se denomi-

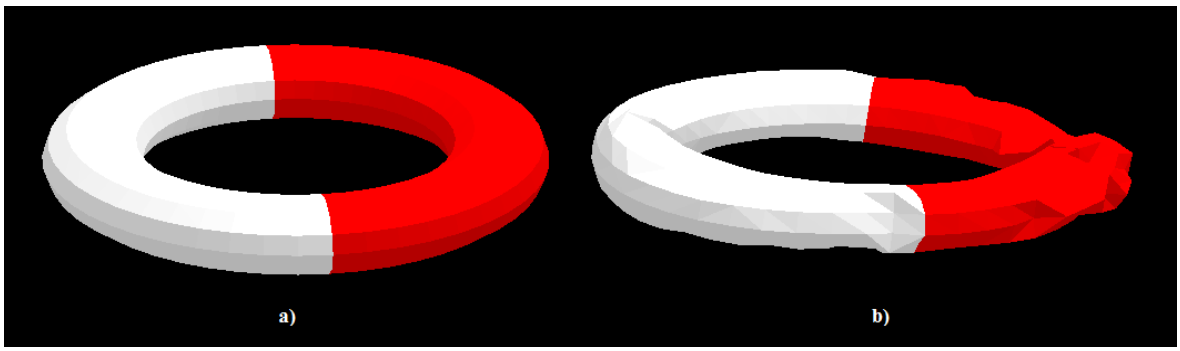

Figura 4.3: a) Torus deformable en su estado inicial b) Torus deformado que no regresa a su forma de inicio.

nara "modelo visible", con una distinta configuración de resortes. En lugar de asociar dos resortes del modelo invisible, estos resortes asocian a cada una de sus masas con la masa correspondiente en el modelo visible, es decir, la masa 1 del modelo visible con la masa 1 del modelo invisible, la masa 2 del modelo visible con la masa 2 del modelo invisible, etc.... La longitud de los resortes de este nuevo modelo es de longitud cero, la rigidez y la amortiguación de los resortes se elige de acuerdo al comportamiento que se pretende obtener (Figura 4.4).

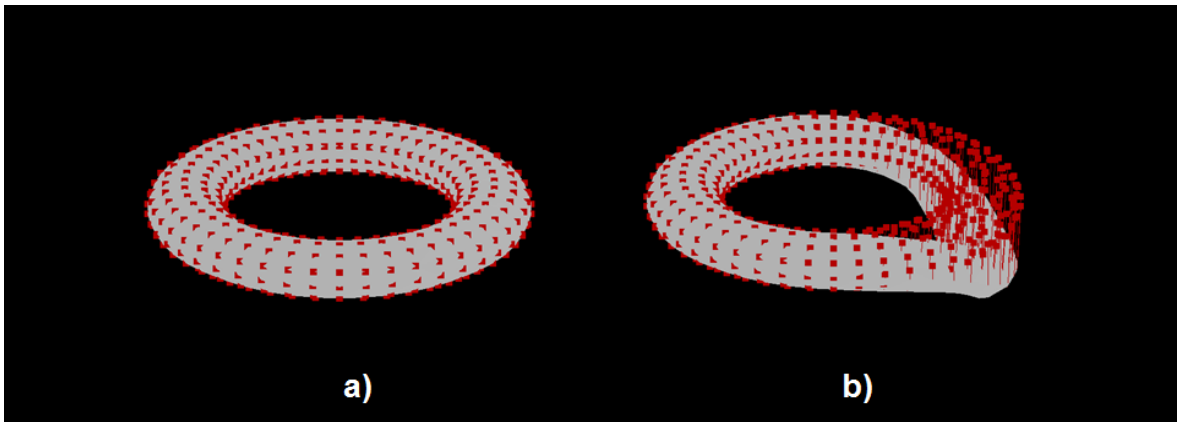

Figura 4.4: a) Torus deformable en su estado inicial b) Torus deformado que vuelve a su forma de inicio utilizando un resorte en cada una de sus masas.

La idea principal tras la implementación del modelo invisible es que el modelo visible

al usuario pueda recuperar su forma, lo cual se logra realizar con éxito, dependiendo de la configuración de rigidez y amortiguación del modelo invisible, se pueden simular materiales que pueden ser tan elásticos o tan rígidos como se desee. Uno de los comportamientos que se observaron en la implementación de ambos modelos, es que en las masas más alejadas a la zona de deformación no se ejerce una fuerza lo suficientemente grande como para que estas cambien su posición de manera notoria. Por lo que se determina que el cálculo de las fuerzas que se ejercen en estas masas es una pérdida de tiempo y de recursos.

Se analizaron los experimentos realizados en la Tabla 4.1, el objeto con el que se experimentó se trata de un Torus compuesto por 517 vértices y un total de 1728 resortes.

	Experimento 1	Experimento 2	Experimento 3
Torus	Rígido	Elastico	Superelastico
Rígidez	1 kg/s^2	0.1 kg/s^2	0.01 kg/s^2
Amortiguación	0.1 kg/s	0.01 kg/s	0.001 kg/s

Tabla 4.1: Configuración de los experimentos realizados con el torus.

En los experimentos realizados era necesario determinar si en todos los casos el modelo deformable podía restaurar su forma y el tiempo que le llevaba hacerlo. En la Figura 4.5 Resultados del experimento con un valor de amortiguación de 0.01 $\frac{kg}{s^2}$ se observa que en todas las configuraciones de prueba el modelo es capaz de recuperar su forma, esto ocurre cuando el número de resortes activos es igual a cero, sin embargo el tiempo en el que este lo hace es variable, de acuerdo a la configuración del modelo invisible.

Dados los resultados de la Tabla 4.2, se observa la cantidad la máxima cantidad de resortes activos en un determinado momento de la deformación y el tiempo que le tomo al modelo regresar a su forma original. Mientras mayor sea la rigidez de

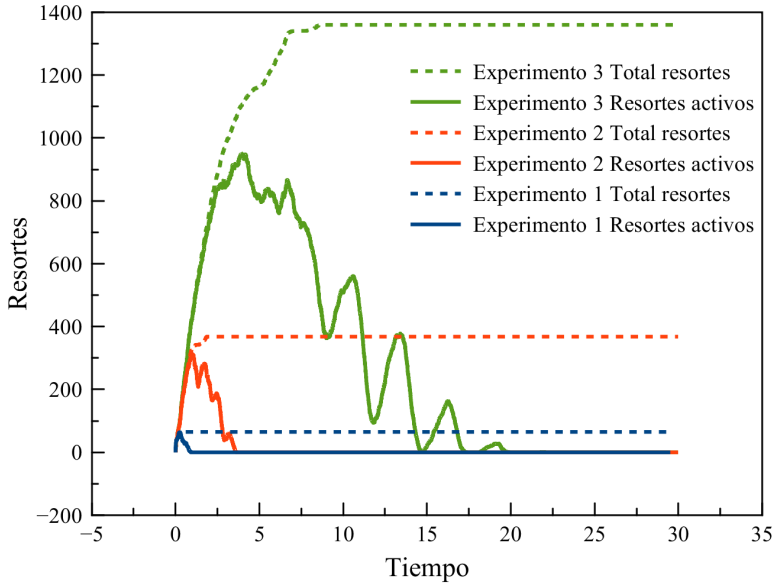

Figura 4.5: Resultados del experimento con un valor de amortiguación de 0.01 $\frac{kg}{s^2}$.

los resortes, el modelo es menos propenso a cambios drásticos, y se puede notar por el número de resortes que se registraron activos. Mientras que el máximo de resortes activos, es el número máximo de resortes que un determinado tiempo de la simulación estuvieron activos a la vez, el total de resortes activos representa la cantidad de resortes en los cuales se ejercicio una fuerza lo suficientemente grande para considerar que hubo un cambio significativo en ellos. Los rangos que se utilizaron para considerar que se había ejercido suficiente fuerza en un determinado resorte fueron +-0.01 Figura 4.5 y +-0.1 Figura 4.6 unidades de diferencia con respecto a su longitud inicial l^0.

Al analizar la tabla anterior se observa que en el experimento 1 la cantidad de resortes que se considera que sufrieron un cambio notorio es muy pequeña, lo que nos indica que la zona de deformación cercana a los vértices en los cuales se aplicó una fuerza es relativamente pequeña (Figura 4.7).

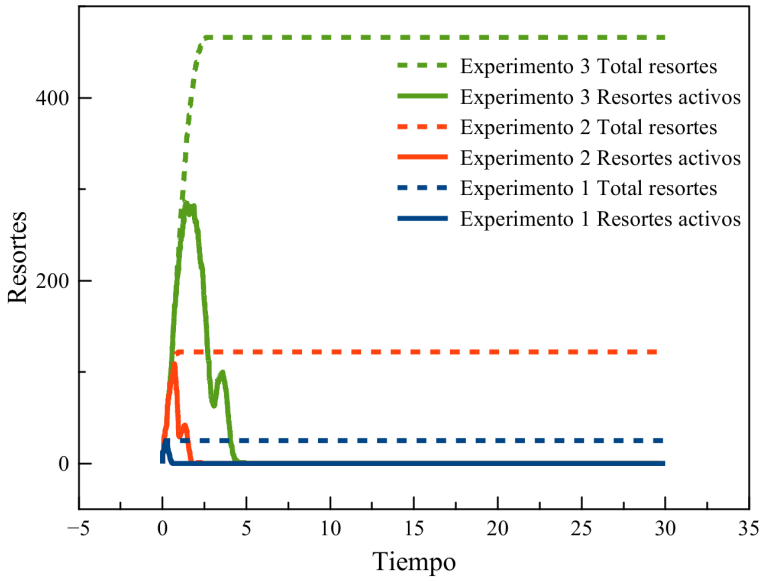

Figura 4.6: Resultados del experimento con un valor de amortiguación de 0.1 $\frac{kg}{s^2}$.

	Experimento 1	Experimento 2	Experimento 3
Torus	Rígido	Elastico	Superelastico
Tiempo de restauración	0.954 s	3.794 s	24.969 s
Rango de actividad con respecto a l^0	+-0.01 unidades		
Máximo de resortes activos	64	312	952
Total resortes activos l^0	65	368	1360
Rango de actividad con respecto a l^0	+-0.1 unidades		
Máximo de resortes activos	25	109	287
Total resortes activos l^0	25	122	466

Tabla 4.2: Resultados de la experimentación.

En el experimento 2 (Figura 4.8) se observa que el número total de resortes activos incremento notablemente con respecto al experimento 1, sin embargo la zona de deformación sigue siendo relativamente pequeña y que estos resortes representan un aproximadamente un 18 % y un 6.3 % del total de los resortes del modelos, por lo que se aprecia que el cálculo de fuerza de los resortes restante resulta sin importancia. En el experimento 3, se observa que una gran parte de los resortes que componen al

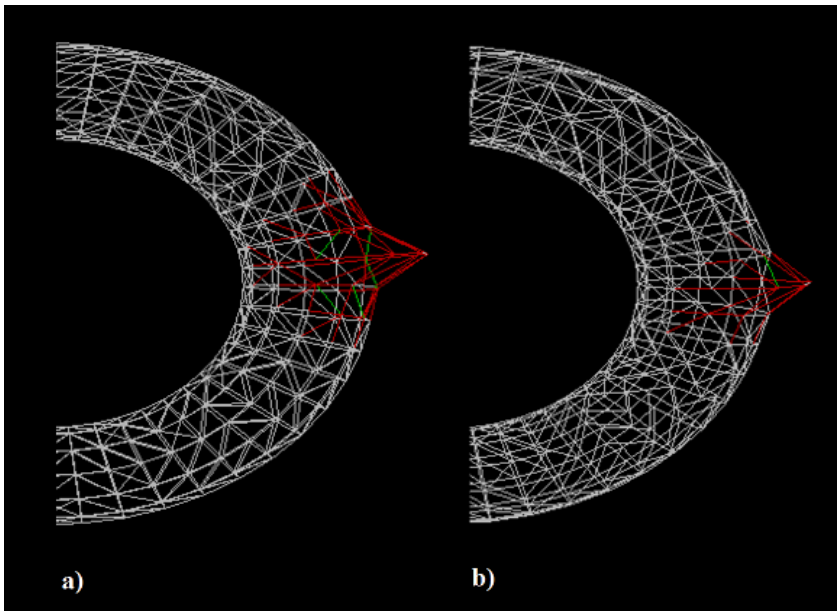

Figura 4.7: a) Zona de deformación del torus con actividad de resortes en +-0.01 b) Zona de deformación del torus con actividad de resortes en +-0.1 .

modelo son utilizados, sin embargo, en estos modelos tampoco es necesario el cálculo de todos los resortes del modelo, esto debido a que se calcularon aproximadamente un 78.7 % y un 27 % de los resortes del modelo.

Además de observar el área de deformación y la cantidad de resortes que se encuentran activos en el modelo visible e invisible, se realizó un análisis con respecto a la cantidad de fotogramas por segundo (FPS) con los que el modelo funciona. Weller [46] menciona que un aplicación de realidad virtual es necesaria que sea ejecutada al menos a 30 FPS para que exista una sensación de realidad suficiente. Debido a que los distintos valores que puedan tener las masas y los resortes del modelo son indiferentes con respecto a los FPS a los que se ejecuta la aplicación, se realizaron dos experimentos los cuales comparan al modelo deformable que no recupera su forma y la adición del modelo invisible (Figura 4.9).

Mantener la forma del modelo deformable con la técnica aplicada resulta ser costoso,

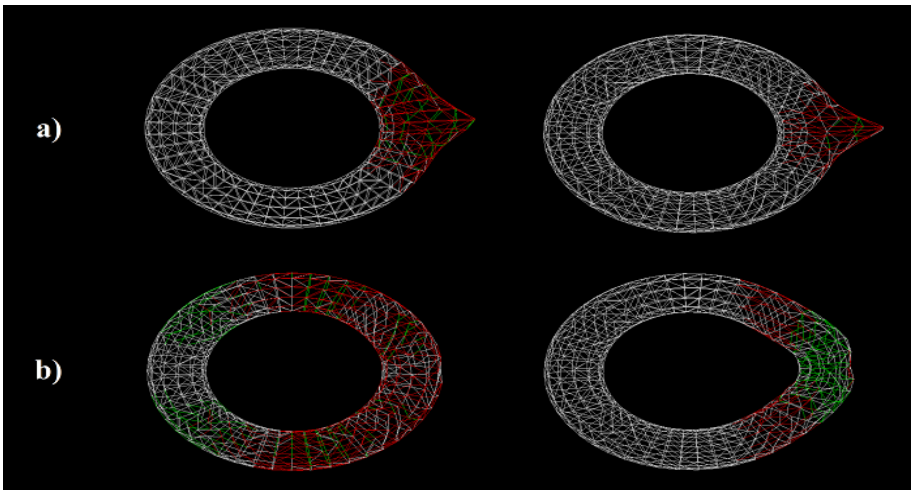

Figura 4.8: Resortes activos durante al aplicar una fuerza sobre el torus; a) el número de resortes activos es menor debido que la constante de rigidez del resorte b) un número mayor de resortes activos debido a un valor de rigidez menor que en a).

debido a que es necesario implementar una cantidad extra de resortes igual a la cantidad de masas que componen al modelo, es por esto que los FPS se ven reducidos considerablemente cuando se aplica la técnica para recuperar la forma del modelo.

4.1.5. Conclusiones de la primera etapa

Al finalizar esta etapa del desarrollo del modelo, fue necesario determinar qué tan viable era seguir aplicando la técnica del modelo invisible para recuperar la forma del modelo deformable. Esto nos lleva a la conclusión de que esta técnica es eficaz, debido a que cumple su objetivo el cual es hacer que las masas regresen a su posición original y por ende el modelo no termine deformado después de aplicar alguna fuerza, sin embargo esta solución conlleva una carga de cálculos adicional. Lamentablemente el uso del sistema MRA ya representa un costo computacional elevado, debido al constante cálculo de las fuerzas generadas en los resortes, incluso si no existe ninguna fuerza actuando sobre ellos, por lo tanto agregar una carga mayor de cálculos se vuelve inviable, sobre todo si el objeto a cargar en el entorno virtual está compuesto por una gran cantidad de triángulos. Es por esto que es necesario buscar una alternativa a esta solución, dicha alternativa debe agregar la menor cantidad

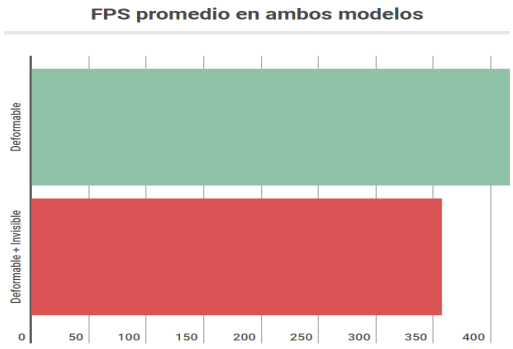

Figura 4.9: Comparación en el rendimiento de un modelo deformable y la implementación del modelo deformable + invisible que recupera su forma.

posible de cálculos, además de ser capaz de retornar los objetos a su estado inicial de configuración. De acuerdo a los resultados de experimentos 1,2 y 3 presentados en esta sección se pudo observar que el porcentaje de resortes que tienen una participación significativa en la deformación del modelo al aplicar alguna fuerza es muy pequeño, sobre todo si el modelo representa algo parecido a un objeto sólido. Cuando el modelo simula un objeto elástico, también se observa que la zona de deformación y los resortes utilizados no llega a afectar por completo al modelo, por lo que inclusive en objetos con altas tasas de deformación existen zonas en las cuales no se llega a aplicar ninguna fuerza que sea significativa. Los resortes de un modelo deformable se calculan continuamente, incluso si no hay interacción alguna con el modelo, por lo tanto, el hecho de que en algunos casos la mayoría de los resortes que no participan en la deformación del modelo sigan calculándose, resulta en un derroche de poder computacional que puede ser ahorrado en la aplicación. Con base en esto se propone la creación de un modelo llamado "modelo rígido-deformable" que permita evitar el cálculo de aquellos elementos del modelo deformable que no se encuentren en uso.

4.2. Desarrollo del modelo rígido-deformable

A partir de los resultados obtenidos durante la etapa anterior, se observa que al aplicar una fuerza sobre el modelo deformable existen regiones en las cuales los

resortes del modelo no generan una fuerza lo suficientemente grande para deformar al modelo. En esta etapa se simulan estas regiones al establecer niveles de profundidad de deformación a partir de una masa o de una cara del modelo, para lograr esto se definieron los vecindarios de estas, los cuales están conformados por todas las masas y caras adyacentes del modelo.

4.2.1. Arquitectura de software

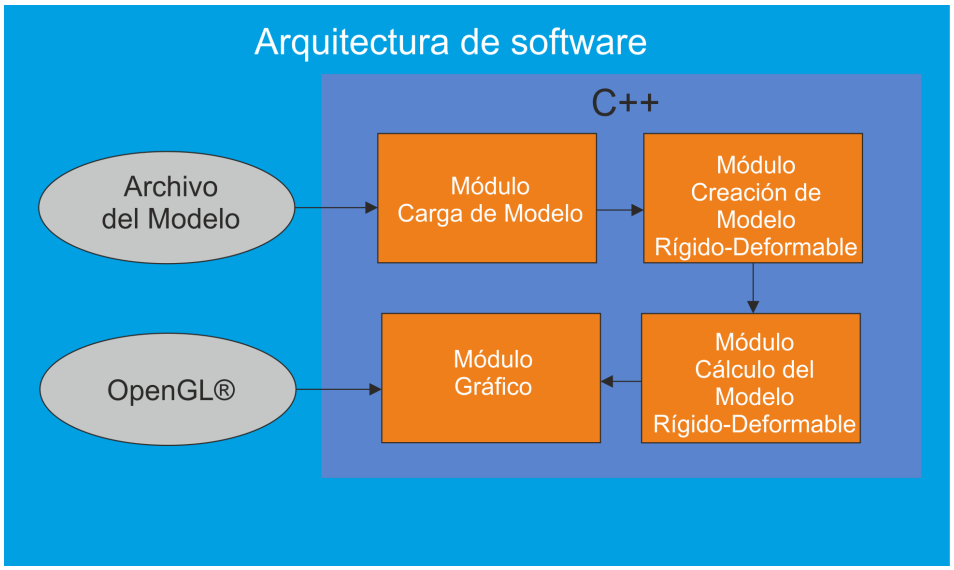

Figura 4.10: Arquitectura de software etapa 2.

Durante el desarrollo del modelo rígido-deformable se sustituyeron los módulos de conversión a modelo deformable y de cálculo del modelo deformable por módulos especializados en el manejo de modelos rígidos deformables. La generación del modelo rígido-deformable se lleva a cabo en el módulo de conversión a modelo rígido-deformable. En este módulo se definen los vecindarios de las caras y masas del modelo propuesto, mientras que en el modelo del cálculo del modelo rígido-deformable se calculas los cambios en el modelo de acuerdo a las distintas regiones de deformación y niveles de profundidad que se requieran.

4.2.2. Conformación del modelo rígido-deformable

Se define al modelo propuesto como una pieza rígido-deformable (Figura 4.11) como un modelo basado en el sistema MRA, que se comporta como un modelo rígido antes y durante su manipulación, y cuando exista una interacción entre dos o más modelos del entorno virtual este presentara deformaciones solo en aquellas áreas donde exista contacto, mientras que el resto del modelo se comportara como un modelo 3D rígido. La primera etapa del desarrollo del modelo rígido-deformable es idéntica a la de un modelo deformable, se crean las masas con sus atributos, se identifican y generan los resortes que compondrán al modelo con sus respectivas longitudes, variables de rigidez y amortiguación.

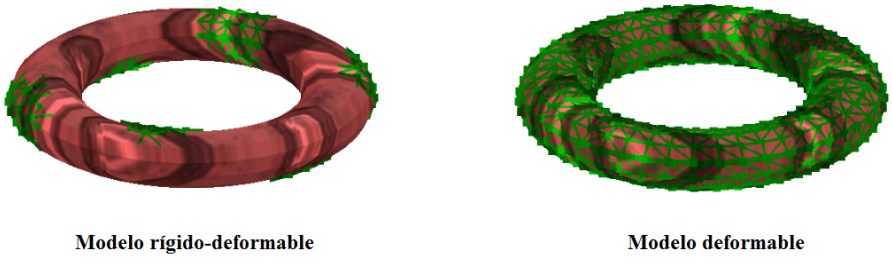

Figura 4.11: A la izquierda un modelo rígido-deformable con zonas de deformación activas y a la derecha el mismo modelo completamente deformable.

Hasta este punto se tiene un modelo deformable que independientemente de si existe interacción o no con el modelo sus masas y resortes se sigues calculando cada vez que la imagen es renderizada.

4.2.3. Vecindario de masas y resortes del modelo

El siguiente objetivo es evitar estos cálculos y por consiguiente incrementar el desempeño de la aplicación. Para lograr esto, la primera tarea que se lleva a cabo es determinar el vecindario de masas y caras que componen al modelo y almacenarlo en

vectores. La razón principal del almacenamiento en vectores, fue para evitar la creación de una matriz de adyacencia, no por la complejidad en crear una, sino en el derroche de memoria que representa una estructura de este tipo.

Aunque parezca que el uso de una matriz de adyacencia o vectores para almacenar la información relacionada a las masas y caras adyacentes sea irrelevante, en realidad es una decisión de suma importancia. Es importante que el modelo que se pretenda crear sea robusto y no solo funcione con determinadas configuraciones en los objetos a cargar a la aplicación. Normalmente una masa cuenta con una cantidad pequeña de masas adyacentes con respecto a la cantidad total de masas de un modelo, por lo tanto el utilizar una matriz de adyacencia representaría una cantidad mayor de derroche de almacenamiento cuando los modelos son lo suficientemente grandes (arriba de 20,000 triángulos). Lo anterior debido a que solo unos cuantos espacios de esta contaran con información lo que provoca obtener una matriz dispersa, es por esto, que conformar las vecindades de las masas y caras en vectores representa una mayor eficiencia en el almacenamiento.

Determinar cuáles son las masas adyacentes a una masa en particular es simple, esta información es posible obtenerla a partir de los datos del archivo ".tri" del objeto que se cargó en el entorno virtual. Cada cara está compuesta por 3 vértices, por lo tanto, se necesita obtener cada una de las posibles duplas de la cara, es decir, dado una cara cualquier compuesta por los vértices 'a', 'b', y 'c', las posibles duplas existentes a partir de la cara dada son (a,b), (a,c), (b,a), (b,c), (c,a) y (c,b), a partir de esto se realiza el mismo procedimiento para todas los vértices del objeto (Figura 4.12).

Conociendo la vecindad de las masas o vértices, también es necesario determinar cuáles son los resortes asociados a cada una de las masas, dicha información es más complicada de obtener, debido a que en la descripción del objeto en el archivo ".tri" no se especifica un número determinado de resortes ni cuáles son los que se asocian a cada masa. La información necesaria se obtiene una vez que se han generado los resortes del modelo deformable, esto debido a que cada resorte tiene asociadas dos masas, una en cada una de sus puntas. Por lo que es necesario realizar un recorrido

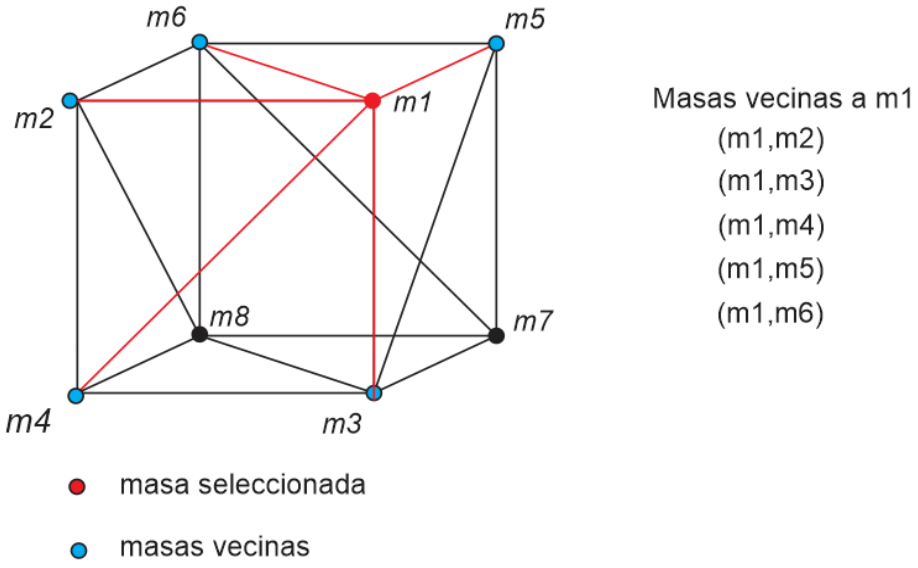

Figura 4.12: Vecindario de masas a partir de una masa seleccionada.

por los resortes del modelo, observar cuales son las masas asociadas a dicho resortes y almacenar en cada una de estas el índice del resorte asociado (Figura 4.13).

4.2.4. Niveles de profundidad de deformación

Realizados estos procedimientos es posible determinar distintas áreas de deformación a partir de un vértice. Dichas áreas de deformación se representan en distintos niveles, partiendo desde el nivel 0 donde el modelo se convierte en un objeto rígido en el cual ninguna de sus masas y resortes son calculados, nivel 1, donde a partir de una masa seleccionada todos los resortes asociados a esta serán calculados, nivel 2, a partir de una masa seleccionada todas las masas vecinas a esta y sus resortes serán activos y por ende se calculara la fuerza que estos ejercen, en un nivel 3, son todas las masas seleccionadas en el nivel 2 agregando las masas vecinas de las masas del nivel anterior. Para los siguientes niveles se vuelve a repetir el proceso mencionado.

Las áreas de deformación permiten concentrar los cálculos en una sola área determi-

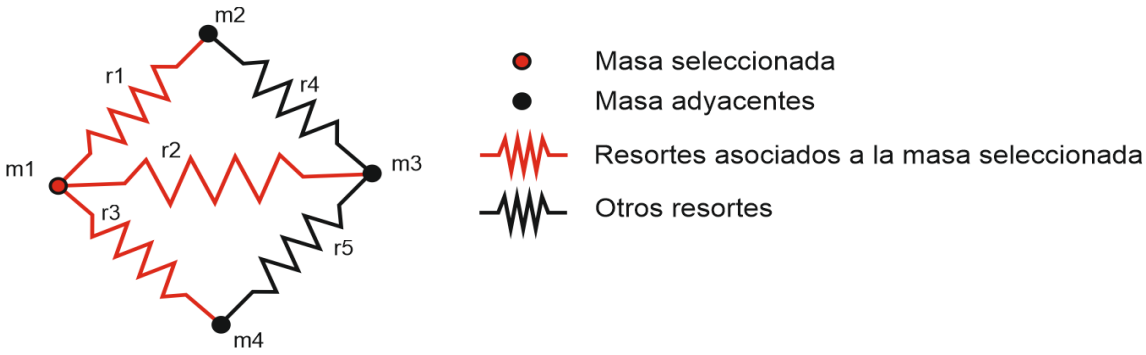

Figura 4.13: Resortes asociados a una masa seleccionada.

nada Figura 4.14, que dependiendo del nivel de profundidad de la deformación del objeto, aumentara o decrecerá la cantidad de resortes a calcular.

Debido a que las interacciones en un objeto ocurren con mayor frecuencia en sus caras, esta técnica puede ser implementada a partir de las mismas. Explicado cómo determinar la vecindad de masas y resortes asociados a estas, el trasladar este enfoque para conformar la vecindad de caras se puede observar en la Figura 4.16.

El archivo ".tri" que describe al objeto, nos muestra información relacionada a las caras, por lo que solamente es necesario realizar un análisis de los datos y determinar cuáles son las caras adyacentes a una cara seleccionada. Una característica por la cual se conoce si dos caras son vecinas, es por que estas comparten al menos uno de sus vértices o aristas. Teniendo esta información en cuenta, resta determinar el vecindario y almacenarlo en los vectores de adyacencia, tal y como se realizó en la masas. Si bien, esta es otra manera de determinar las áreas cercanas a un área en específico del modelo, esto no significa que sea necesario realizar otro método especifico para este enfoque. Al contrario, es posible reutilizar el enfoque de las masas cercanas y aplicarlo a las caras, esto debido a que cada cara esta compuesta por 3 masas, por lo tanto, es necesario utilizar la vecindad de estas a un determinado nivel para obtener el área de deformación.

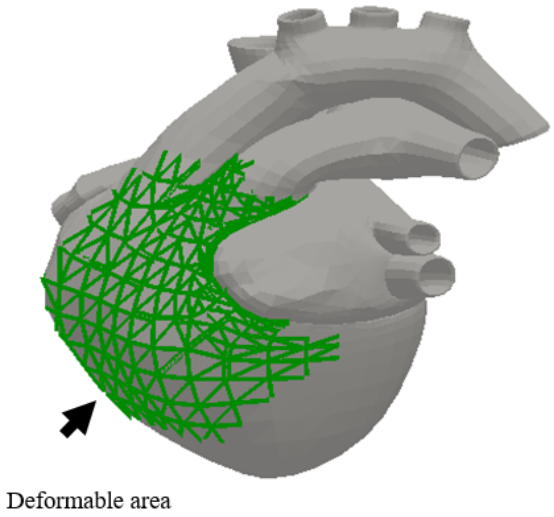

Figura 4.14: Modelo rígido-deformable de un corazón (El área verde es la zona que permite representar deformaciones).

En este etapa del desarrollo del modelo rígido-deformable resta observar cual es el comportamiento de este en comparación con el modelo deformable convencional. La principal diferencia de ambos modelo radica en que el modelo propuesto evita calcular aquellos resortes lejanos a la zona de contacto, para esto se establecen areas de deformación, que dependiendo del nivel de profundidad del área, habrá una menor o mayor cantidad de resortes que es necesario que sean calculados. A continuación se presentan los diferentes experimentos con los cuales se validan el rendimiento en FPS del modelo propuesto en comparación con el modelo deformable. Se utilizarón tres modelos (Tabla 4.3) con distinta cantidad de masas y resortes cada uno. En adición a esto se observaron los FPS a los que corre la aplicación con cada uno de los modelos mencionados y con distintos niveles de profundidad.

Es importante mencionar que independientemente de la fuerza que se aplique a una masa y de la variables de los resortes el rendimiento de la aplicación es el mismo, lo único en lo que esto puede afectar es en la cantidad de ciclos en los que un resorte puede retornar a su longitud de reposo, además de esto en la aplicación no se presenta ningún otro inconveniente. En todos los modelos se establecieron hasta 6 áreas

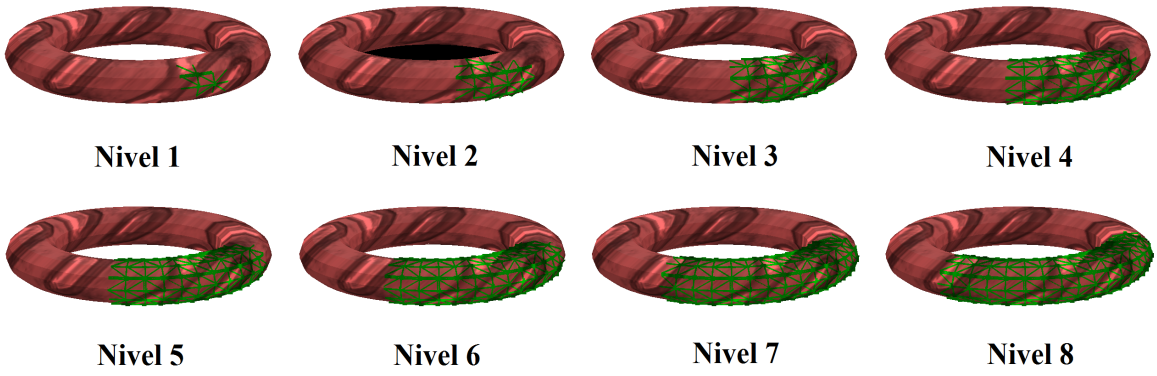

Figura 4.15: Retorno de una masa a su posición original a través del uso de restricciones.

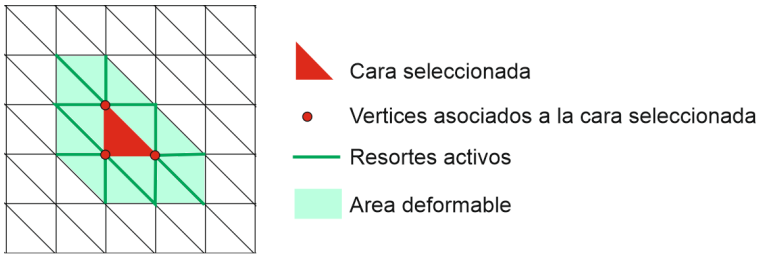

Figura 4.16: Niveles de profundidad de deformación de un modelo rígido-deformable.

de deformación, donde cada una de estas zonas abarcaba un área de deformación de hasta 8 niveles. El nivel 0 no fue necesario analizarlo porque simplemente no existe ninguna deformación en el modelo independientemente del número de áreas de deformación que se establezcan. En la Tabla 4.4 se muestran los resultados obtenidos al representar cada uno de los modelos de la experimentación como modelos completamente deformable, utilizando el sistema MRA, mientras que en la Figura 4.17 se observa el rendimiento del Torus 1, el rendimiento de los otros modelos se encuentre en (anexos)

Cuando solamente existe una zona de deformación con un nivel de profundidad 1,

Modelo	Elementos	
	Masas	Resortes
Torus 1	576	1728
Torus 2	1566	4600
Corazón	7349	22087

Tabla 4.3: Numero de masas y resortes de los modelos utilizados.

Modelo MRA	FPS		
	Mínimo	Máximo	Promedio
Torus 1	348	357	353
Torus 2	183	187	185
Corazón	34	38	36.5

Tabla 4.4: Comparación en el rendimiento de los modelos utilizados .

el modelo rígido-deformable presenta un aumento en el rendimiento hasta 5 veces mayor que el modelo deformable, lo cual representa excelentes resultados en la implementación de este modelo y por lo tanto es posible utilizar una mayor cantidad de modelos con el ahorro de cálculos que se presenta en estas circunstancias.

Por otra parte, cuando existen suficientes áreas de deformación y estas operan a un nivel de profundidad lo suficientemente alto para que el modelo rígido-deformable se comporte como un modelo deformable convencional, el rendimiento en la aplicación se ve reducido hasta un 15 %. En el caso del Torus 1, las zonas de deformación del modelo rígido-deformable se vuelve lo suficientemente grande para que todos los resortes del modelo se encuentren activos cuando su profundidad de deformación se encuentra en nivel 7, por lo que incrementar el nivel de profundidad no influirá en el desempeño del mismo.

Cuando solamente existe una zona de deformación con un nivel de profundidad 1, el modelo rígido-deformable presenta un aumento en el rendimiento hasta 5 veces mayor que el modelo deformable, lo cual representa excelentes resultados en la implementación de este modelo y por lo tanto es posible utilizar una mayor cantidad de modelos con el ahorro de cálculos que se presenta en estas circunstancias.

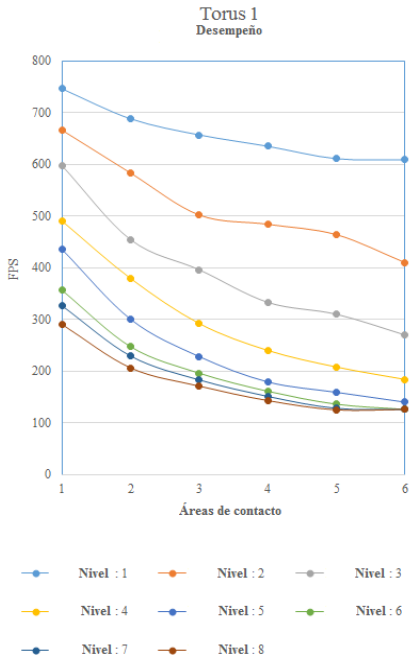

Figura 4.17: Desempeño del modelo torus 1 con distintos puntos de contacto y niveles de profundidad de deformación .

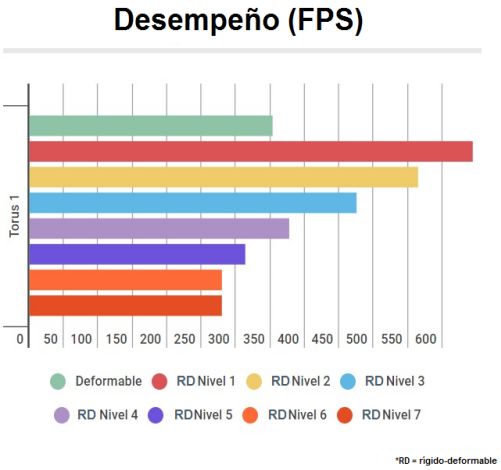

Figura 4.18: Rendimiento del modelo (torus 1).

Para mayor claridad, en la Figura 4.18,Figura 4.17 y Figura 6.3 se presenta el desempeño de todos los modelos con el máximo de áreas de deformación presentadas y con sus distintos niveles de profundidad de deformación.

4.2.5. Restauración de la forma del modelo

Para restaurar la forma de los modelos rígido-deformables, se toma como inspiración los trabajos [6, 29] al implementar restricciones sobre los resortes. Dichas restricciones se basan en determinar si las masas activas se encuentran en su posición inicial. Para lograr esto se utiliza una variable (fixedPosition) en la clase Mass que almacena la posición en la que se encuentran las masas en todo momento, incluso si existen transformaciones en el objeto. fixedPosition termina representando a las masas como si de un modelo rígido se tratase, debido a que en estas se presentaran traslaciones y rotaciones, pero la fuerza que ejerzan los resortes no tendrán impacto sobre ella.

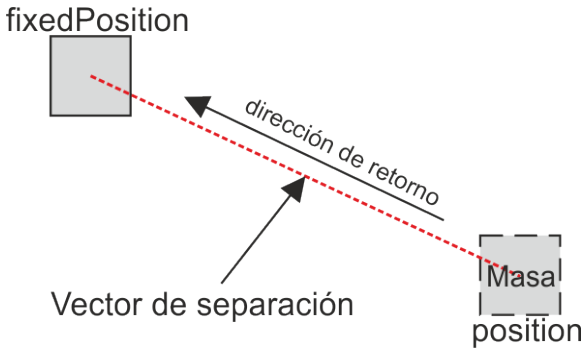

Figura 4.19: Retorno de una masa a su posición original a través del uso de restricciones.

Si las variables position y fixedPosition no se encuentran en la misma posición, la restricción implementada sobre las masas se activa y se identifica cual es el vector de separación en que se encuentran separadas, identificado este vector la masa retornara a su fixedPosition en dirección de su vector de separación en un porcentaje de acuerdo a la distancia de este vector Figura 4.19.

4.2.6. Conclusiones de la segunda etapa

Se ha demostrado que el modelo rígido-deformable tiene un mejor desempeño que el modelo deformable cuando no ocurre ninguna deformación, e incluso si existen grandes áreas con resortes que se encuentren calculándose. Además se observa que en los experimentos realizados con los modelos deformables de la Tabla 3, que al aplicar una deformación en el modelo, gran parte de este no sufre cambios, por lo que el uso de modelos rígido-deformables puede ser de utilidad.

El modelo propuesto funciona con un mayor rendimiento cuando las áreas de deformación y sus niveles de profundidad de deformación son pequeños y en los mejores resultados de las pruebas estos muestran hasta un rendimiento 5 veces mayor. Sin embargo, aunque estos resultados resultan bastante prometedores, cabe mencionar que en muchas ocasiones los modelos necesitan ser completamente deformables, por lo que la implementación del modelo rígido-deformable no es recomendable, lo anterior debido a que se observó que mientras más grande sea el área de deformación, la cantidad de resortes a calcular incrementa considerablemente, por lo que el algún algunas configuración del modelo, puede llegar a tener el mismo rendimiento que los modelos deformables convencionales, incluso si el área deformable del modelo propuesto no abarca a todo el objeto. En el peor de los casos, cuando el área de deformación del modelo rígido-deformable cubre a todo el objeto, se tendrá un peor desempeño en este, por lo que en estas ocasiones específicas, un modelo deformable representaría una mejor opción.

El modelo rígido-deformable representara una mejor alternativa, cuando no se necesite modelar con precisión el comportamiento de un objeto, si no en ocasiones donde sea necesario observar algunas pequeñas o medianas deformaciones en los objetos modelados. Se menciona que las masas más alejadas con respecto a una deformación en alguna zona del modelo sufren cambios muy pequeños, por lo que en ningún momento dejaron de cambiar su posición. Es por lo anterior, que se concluye que la fuerza que se ejerce sobre ellas es tan sutil que es posible evitar el cálculo de estas y al mismo tiempo obtener un resultado visual similar.

Además de esta experimentación, es necesario desarrollar un sistema detector de colisiones que permita interactuar con este tipo de modelos, esto debido a que ninguno de los motores de física actuales implementa este tipo de comportamiento en sus modelos. Puede haber una interacción entre dos modelos rígidos, entre un modelo rígido y un modelo deformable o entre dos modelos deformables, pero implementar este tipo de modelos en estos motores nos impide que estos puedan interactuar con otros modelos, por lo que en la siguiente etapa del desarrollo, se presenta un sistema de detección de colisiones personalizado para el modelo rígido-deformable.

4.3. Desarrollo del sistema detector de colisiones para modelos rígido-deformables

Durante esta etapa se desarrolla un sistema de detección de colisiones para los modelos rígido-deformables utilizando técnicas de división espacial y volúmenes delimitadores para representar a los objetos dentro del entorno virtual. Además de lo anterior, se detalla cómo se llevan a cabo las deformaciones de los objetos rígido-deformables al existir una interacción con otros objetos del entorno virtual.

4.3.1. Arquitectura de software

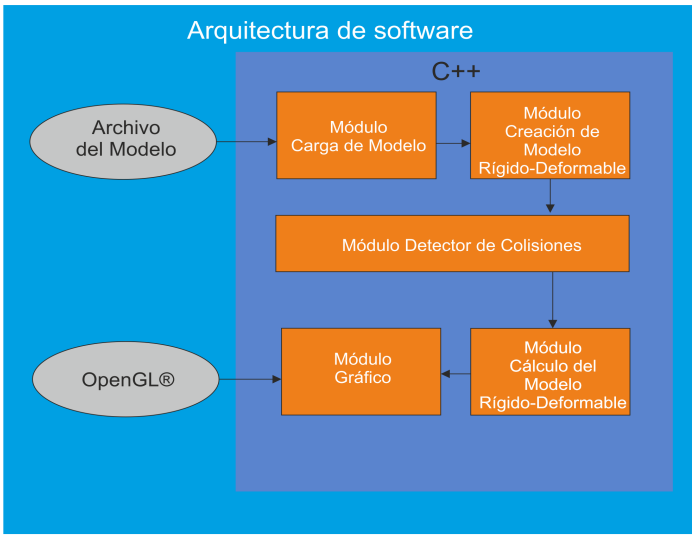

Figura 4.20: Arquitectura de software etapa 3.

Durante esta etapa se implementa un módulo de detección de colisiones en la arquitectura de software. En este módulo se determina cuáles son las piezas que se encuentran colisionando y proporciona cuales son las caras en las cuales ocurre dicha colisión. A partir de la información obtenida durante este módulo se es capaz de representar las deformaciones de las piezas cuando existe un contacto con otra pieza dentro del entorno virtual (Figura 4.20).

4.3.2. Sistema detector de colisiones

Debido a que ningunos de los motores de física actuales provee algún sistema de detección de colisiones que permita representar cuando ocurre algún tipo de interacción entre este tipo de modelos, fue necesario crear uno que se adecue a las necesidades de estos.

Al desarrollar un sistema de detección de colisiones personalizado fue necesario establecer qué tipo de BV utilizar para contener a los objetos que se encuentran en el entorno virtual. En este trabajo se optó por utilizar esferas como BV, debido a que son simples de representar y por ser rotacionalmente invariables, aunque es cierto que el ajuste que tienen sobre los objetos es pobre, en esta esta etapa del trabajo se necesita un sistema de detección de colisiones que permita interactuar a estos modelos, y no uno robusto que permita distintas configuraciones de BVs.

4.3.3. Determinar los atributos del bounding volume

Para determinar cuáles son los valores de los atributos de cada esfera que contiene a los objetos dentro del entorno virtual, fue necesario determinar el centro de un objeto y encontrar la masa a mayor distancia de este. Para encontrar el centro de un objeto, se utilizó una técnica simple que consiste en encontrar los valores más grandes y más pequeños para cada uno de los ejes en el plano, con esta información se obtienen los vectores de coordenadas ($\vec{Min}$) y ($\vec{Max}$) con los cuales es necesario obtener el punto medio entre ambas coordenadas (ver Ecuaciones 4.1,4.2).

$$\vec{Offset} = \frac{\vec{Min} - \vec{Max}}{2} = \frac{\begin{bmatrix} Min_x \\ Min_y \\ Min_z \end{bmatrix} - \begin{bmatrix} Max_x \\ Max_y \\ Max_z \end{bmatrix}}{2} \tag{4.1}$$

$$\vec{Centro} = \begin{bmatrix} Min_x \\ Min_y \\ Min_z \end{bmatrix} + \begin{bmatrix} |Offset_x| \\ |Offset_y| \\ |Offset_z| \end{bmatrix} = \begin{bmatrix} Max_x \\ Max_y \\ Max_z \end{bmatrix} - \begin{bmatrix} |Offset_x| \\ |Offset_y| \\ |Offset_z| \end{bmatrix} \tag{4.2}$$

Una vez obtenido el centro del objeto, es necesario obtener la masa a mayor distancia de este, para realizar esto es necesario determinar la distancia entre el punto central del objeto con cada una de las masas que lo componen utilizando la Ecuación 2.10, y una vez encontrada la mayor distancia se estableció como el radio de la esfera (Figura 4.21).

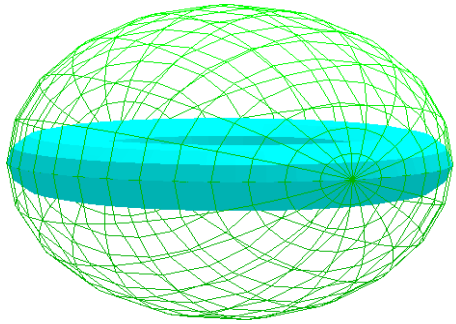

Figura 4.21: Bounding volume de un torus.

4.3.4. Conformación del BVH del modelo

A partir de obtener el BV que conforma a cada una de las figuras en el entorno virtual, es necesario conformar el BVH del entorno virtual. Se utilizó la técnica de creación de BVH top-down donde el nodo raíz es un BV (en este caso una esfera) lo suficientemente grande como para abarcar todos los objetos que se encuentran en el entorno virtual (Figura 4.22).

Los nodos hijos del nodo raíz son creador a partir del eje con mayor longitud (Figura 4.23), es decir, si el eje con mayor longitud es x entonces se dividirá el espacio en dos partes a partir del centro del BV que lo ajusta con respecto al eje x, por lo que todos los BV de los objetos cuya coordenada de su centro sea mayor a la coordenada en

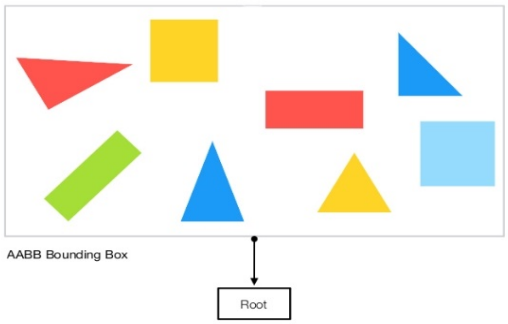

Figura 4.22: Conformación del nodo raíz de un BVH.

x del nodo raíz conformaran al hijo derecho de este, y por consiguiente los que sean menores conformaran a su hijo izquierdo. Lo mismo ocurre cuando el eje de mayor longitud es y o z.

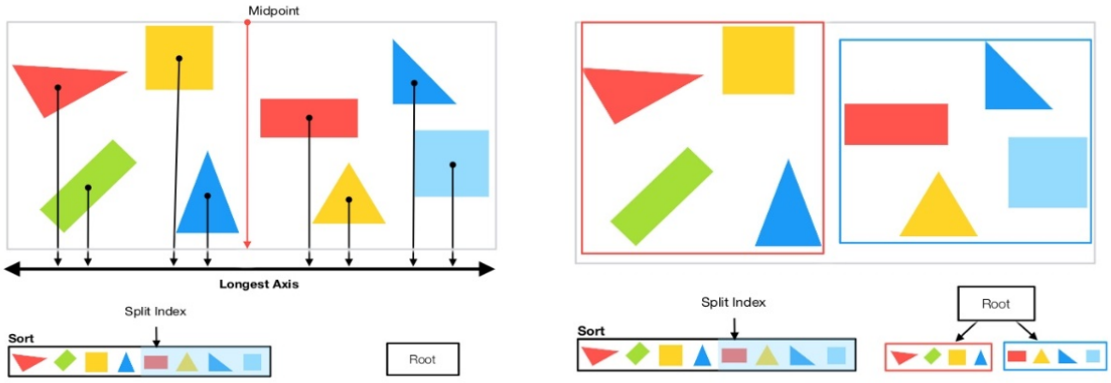

Figura 4.23: Conformación de los nodos hijo del nodo raíz del BVH.

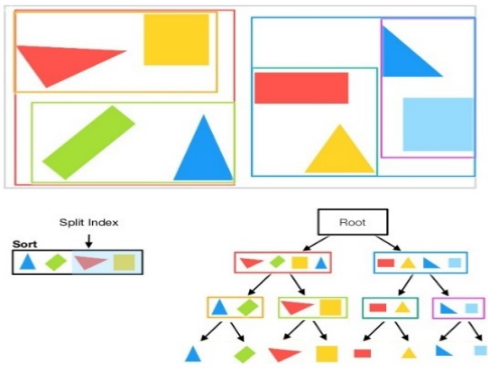

Figura 4.24: Bounding volume hierarchy de la escena virtual.

Realizado este procedimiento, se recalcula el BV que ajuste a todos los objetos que se encuentran en cada uno de los nuevos nodos creados, y se repite este proceso recursivamente para la generación de los nuevos nodos hijos de mayor nivel. Esta tarea termina una vez que solo existe un objeto en el nodo, lo que querrá decir que este se convertirá en un nodo hoja de la jerarquía (Figura 4.24).

Una vez conformado el BVH del entorno virtual, se procede con el desarrollo de la fase amplia del sistema de detección de colisiones. Dicha fase consiste en determinar cuáles son los pares de objetos que posiblemente se encuentren colisionando, almacenarlos y posteriormente realizar una prueba que nos permita determinar con exactitud si ocurre o uno una colisión.

Para determinar cuáles son los objetos que posiblemente se encuentren colisionando con otros es necesario comparar cada uno de los BVs con el BVH del entorno virtual. Esta búsqueda comienza comparando los BV del objeto con el nodo raíz del BVH y aplicando la prueba presentada en la Figura 2.16, y recorriendo el árbol como si de una búsqueda en profundidad se tratase (Figura 2.23). Mientras las pruebas realizadas resulten positivas sobre nodos rama, se seguirá descendiendo en el árbol hasta que estas se efectúen sobre un nodo hoja. Si la prueba realizada en el nodo hoja dan un resultado positivo, el par de modelos es almacenado en un contenedor que posteriormente analizara cuales son los puntos específicos donde una intersección ocurre.

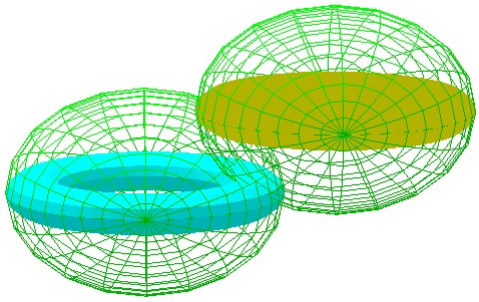

Figura 4.25: Bounding volumes de ambos torus colisionando.

La fase amplia del sistema de detección de colisiones elaborado no presenta ningún cambio en comparación a otros sistemas ya utilizados, debido a que solo se trata de filtrar a aquellos posibles objetos que se encuentran colisionando sin necesidad de desperdiciar poder computacional.

Una vez encontrados todos los pares de objetos que posiblemente se encuentren colisionando, es necesario realizar pruebas a mayor nivel de detalle que permitan saber cuáles son los puntos exactos donde ocurre una o varias intersecciones entre el par de objetos. Dicha fase es conocida como fase estrecha.

Tal cual como se determinó cuáles son los objetos que posiblemente se encuentren colisionando durante la fase amplia, durante esta fase se procura obtener cuales son las caras que posiblemente se encuentren colisionando entre ambos modelos. Esto es posible lograrlo de la misma manera que en la fase anterior, sin embargo, en esta fase no se necesitan la BVH del entorno virtual, sino que es necesario crear una BVH de los objetos y hacer una comparación entre ambas.

4.3.5. Definición del baricentro de los triángulos

Para la creación del BVH de cada objeto es necesario seguir el mismo procedimiento con el cual se creó el BVH del entorno virtual. Este BVH se conformara de manera local dividiendo el espacio en el que se encuentra el objeto y los BVs del mismo se

conformaran a partir del baricentro de cada una de sus caras. El baricentro G es el punto donde concurren las tres medianas (m_a, m_b, m_c) del triángulo, dichas medianas son los segmentos que unen uno de sus vértices con el centro del lado opuesto (Figura 4.26).

A partir de obtener el baricentro de cada una de las caras del objeto es posible crear su BV. El BV será conformado por una esfera cuyo centro será el baricentro del triángulo y su radio será definido por la mayor distancia del baricentro a cualquiera de sus vértices (Figura 4.27).

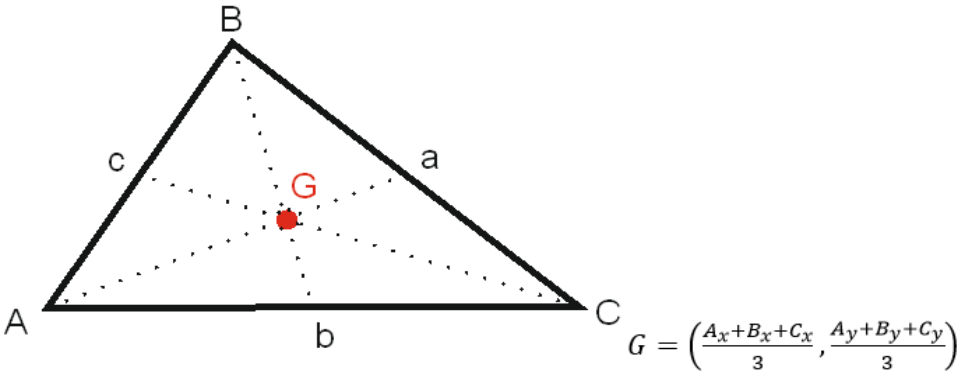

Figura 4.26: a) Modelo de un cubo donde se observa que el BV de cada una de sus caras es demasiado grande comparación a estas b) Modelo de un torus con caras pequeñas y BV más finos.

Es importante mencionar que mientras menor sea el tamaño del triángulo a contener, menor es el espacio desperdiciado por las esferas, en la Figura 4.27 se puede observar los tamaños de los BVs de cada una de las caras, mientras que en el cubo los triángulos que lo componen son grandes y con un BV que abarca gran espacio, en las caras del torus que son pequeñas los BVs de estas no abarcan demasiado espacio.

Un problema que se presenta al realizar una prueba de intersección entre modelos cuyos triángulos son muy grandes y modelos con triángulos pequeños, es que el BV de los triángulos grandes a abarcar gran cantidad de caras del otro modelo, lo que provoca un realizar un gran número de pruebas para determinar si existe una intersección o no, que en la mayoría de los casos resultara falsa (Figura 4.28).

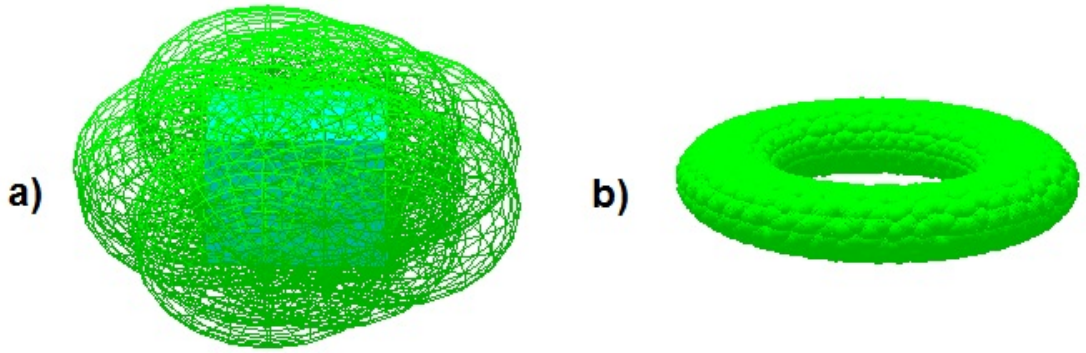

Figura 4.27: a) Modelo de un cubo donde se observa que el BV de cada una de sus caras es demasiado grande comparación a estas b) Modelo de un torus con caras pequeñas y BV más finos.

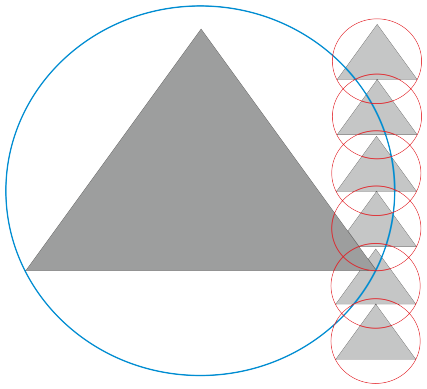

Figura 4.28: El BV azul del triángulo con mayor tamaño es lo suficientemente grande para cruzarse con los BVs de los triángulos pequeños de la derecha, aunque solo existe una colisión con dos triángulos es necesario realizar la prueba en cada uno de ellos.

Al conocer los BVs de todas las caras del objeto resta crear su BVH cuyo nodo raíz es el BV del objeto, y para determinar los nodos hijos se sigue el mismo procedimiento ya observado a partir de la Figura 4.25, solo que en lugar de buscar todos los objetos con valores mayores o menores a partir de un punto de un determinado eje, esta vez se buscaran todos los baricentros de la caras. Una vez terminado el proceso de creación del BVH se obtuvo un árbol cuya raíz en el BV del objeto y todos sus nodos hoja representan el BV de cada una de sus caras (Figura 4.29).

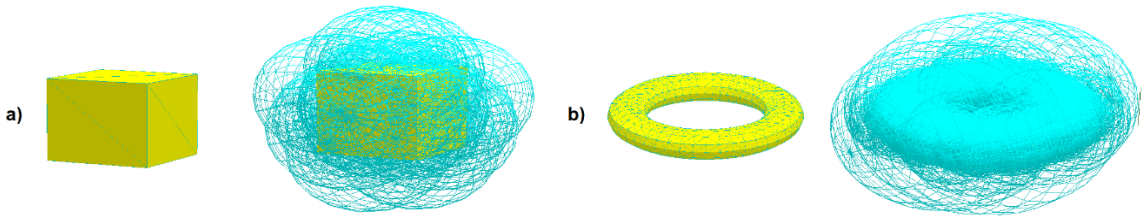

Figura 4.29: a) BVH de un cubo, cuyos BVs de cada nodo de la jerarquía es una esfera b) BVH de un torus, cuyos BVs de cada nodo de la jerarquía es una esfera.

4.3.6. Comparación entre los BVH de dos modelos

Una vez que obtenidos los BVH de los objetos del entorno virtual y los pares de objetos que posiblemente se encuentren colisionando durante la fase amplia del sistema detector de colisiones, resta comparar los BVH de cada uno de los pares. La comparación de los arboles ocurre de la misma manera que durante la fase amplia, incluso la prueba de intersección no cambia debido a que se utilizaron esferas como BV. La diferencia es que en esta ocasión los pares que posiblemente colisionan no son pares de objetos, si no pares de caras que es necesario determinar si estas están en contacto o no.

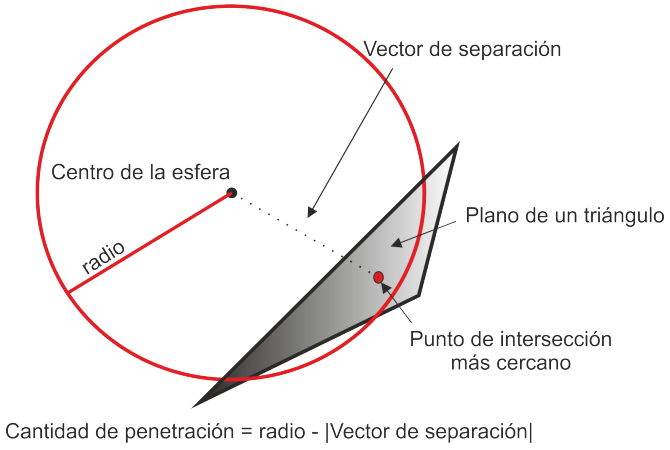

Figura 4.30: Intersección entre una esfera y el plano de un triángulo .

Hasta este punto de la fase estrecha se han detectado las posibles caras que intersectan entre un par de modelos, y ahora es necesario determinar si estas intersectan.

Anteriormente se observó que el ajuste de las esferas no es lo suficientemente bueno, así que determinar que existe una intersección entre dos caras a partir de sus BV resulta poco acertado. Una solución a esto es el realizar una prueba a partir de la geometría de sus caras, es decir una prueba de intersección triangulo-triangulo, que resulta ser mucho más complicada y costosa computacionalmente, además resulta difícil aplicar una deformación debido a la forma compleja de estas figuras. Para solventar esto se decidió realizar una prueba de intersección esfera-triangulo (Figura 4.30), donde uno de los modelos actuara como un modelo rígido-deformable aplicando deformaciones en las áreas de contacto y el otro como un modelo rígido que no sufrirá deformaciones y podrá deformar al modelo rígido-deformable.

Dicha prueba de intersección consta en determinar la penetración de la esfera de una cara del modelo rígido dentro del triángulo del modelo rígido-deformable y la dirección en la que la deformación se aplicara. Para esto se utiliza el algoritmo presentado en [10] el cual con el cual se determina en que coordenadas del triángulo se encuentra el punto más cercano al centro de la esfera. Conociendo esta información es posible aplicar una deformación al modelo rígido-deformable solo en aquellas caras que presenten una intersección, y la cantidad de deformación que se aplicara en las caras será de acuerdo a la distancia de penetración y a su vector de dirección. Al aplicar una deformación en la cara del modelo rígido-deformable es necesario recalcular de posición las 3 masas que componen a la cara.

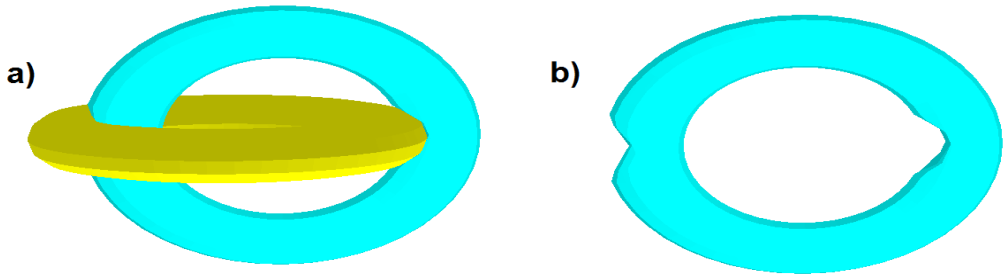

Figura 4.31: Simulación de deformación con el sistema detector de colisiones para modelos rígido-deformables. Torus amarillo es un modelo rígido, y el Torus azul es rígido-deformable. a) Torus amarillo deformando al Torus azul. b) Torus amarillo invisible, donde se muestra la deformacion del Torus azul.

4.4. Tareas de ensamble con modelos rígido-deformables

Durante esta etapa se detalla la técnica implementada para llevar a cabo el ensamble de dos piezas. Se toma como inspiración la técnica snap-fitting mejorado que representa mediante vectores a dos piezas a ensamblar (pieza primaria y pieza receptora o secundaria), y se cambia la representación vectorial de la pieza secundaria por un área de ensamble. Cuando el vector que representa a la pieza primaria se encuentra inmerso dentro del área de ensamble definida, la pieza primaria lleva a cabo una serie de transformaciones geométricas que la llevan a su posición de ensamble final.

4.4.1. Arquitectura de software

Durante esta etapa se implementa un módulo de detección de colisiones en la arquitectura de software. En este módulo se determina cuáles son las piezas que se encuentran colisionando y proporciona cuales son las caras en las cuales ocurre dicha colisión. A partir de la información obtenida durante este módulo se es capaz de representar las deformaciones de las piezas cuando existe un contacto con otra pieza dentro del entorno virtual (Figura 4.32).

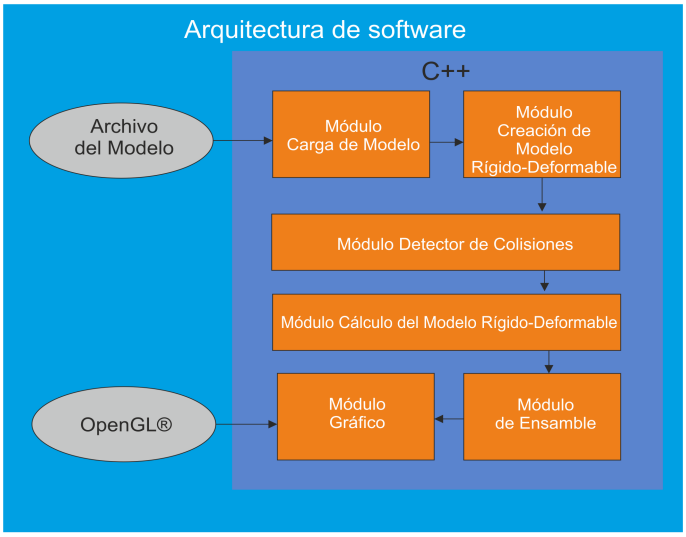

Figura 4.32: Arquitectura de software etapa 4.

4.4.2. Tareas de ensamble

En esta etapa se utilizan los modelos rígido-deformables para analizar su factibilidad en el apoyo a las tareas de ensamble virtual y se compara su comportamiento en comparación de los modelos rígidos. La tarea de ensamble con la que se realizaron los experimentos consiste en atravesar un torus con una esfera, la idea detrás de este experimento es simular que las esferas son pelotas y que el torus es la boca de un contenedor que almacenara a las pelotas tal y como se muestra en la Figura 4.33.

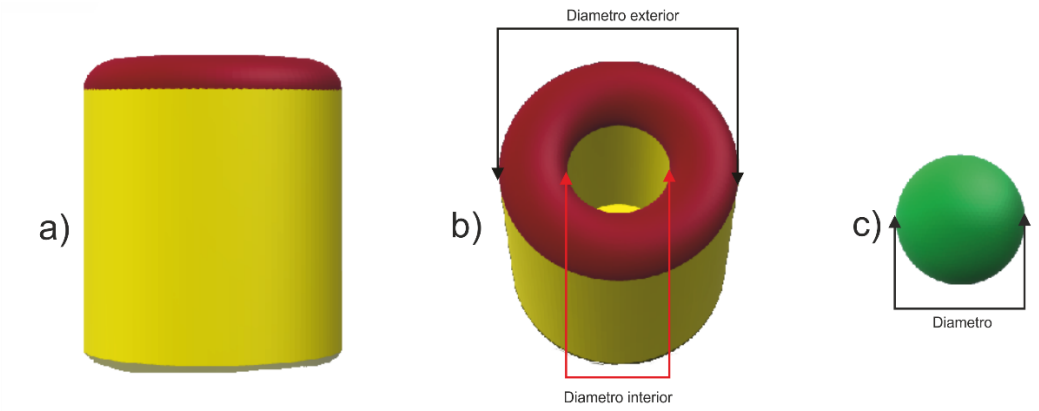

Figura 4.33: a) Vista lateral del contenedor b) Vista de la parte superior del contenedor c) Vista de la pelota.

Los elementos a tomar en cuenta de cada modelo, son el diámetro interno del torus y el diámetro de la esfera que se introducirá. Si el diámetro de la esfera es menor al diámetro interno del torus, la esfera es capaz de atravesarlo.

Otro caso posible es que el diámetro de la esfera sea igual o mayor al diámetro interno del torus, en esta situación si los objetos son rígidos, la esfera difícilmente puede atravesar el centro del mismo. Para comprobar esto, se realizó una serie de experimentos los cuales consisten en utilizar dos modelos rígidos (Un torus y una esfera) con comportamiento dinámico utilizando el motor de física Physx^{TM}, el primero simulando la boca del contenedor y el segundo siendo la pelota que tiene que entrar a este.

En la Tabla 4.5 se muestran las medidas del torus, el diámetro de las esferas y el resultado de los experimentos. La manera en la que se realizaron estos experimentos es la siguiente: en primer lugar se sitúa la esfera a una mayor altura que el torus y alineándola con el centro de este, posteriormente se le aplica una fuerza de gravedad (9.81 m/s) a la esfera, lo cual ocasiona que esta simule caer al suelo y por ultimo resta confirmar si la esfera atravesó al torus o no la atravesó.

Torus		Esfera	¿Logro atravesar?
Díametro externo	Díametro interno	Díametro	
15	25	<15.1 >15.1	NO SI

Tabla 4.5: Experimentos con un modelo de torus con comportamiento dinámico utilizando PhysX.

Cuando el diámetro de la esfera utilizada en el experimento oscila entre los valores 15 15.1, en teoría no debería poder atravesar el centro del torus, sin embargo el sistema detector de colisiones que utiliza Physx^{TM} para estos objetos provoca que esta clase de comportamientos ocurra si se aplica la fuerza suficiente. Incluso si el tamaño de la esfera es tan grande como el diámetro externo del torus este puede ser atravesado si se aplica la fuerza suficiente. En el caso presentado en la Figura 4.34 se aplicó una fuerza de 2500 unidades en dirección del vector (0,-1,0), lo que provoco que una esfera más grande que el diámetro externo del torus pudiera atravesarlo.

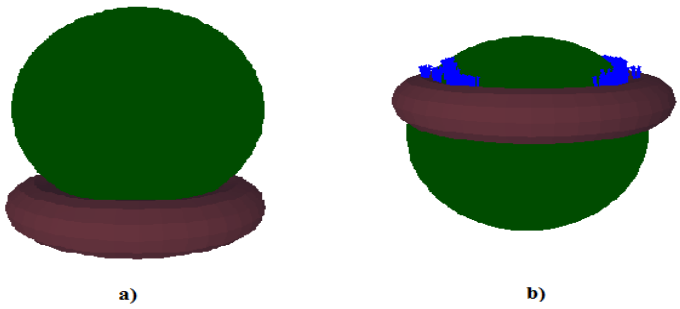

Figura 4.34: A la izquierda las piezas a ensamblar utilizando la técnica snap-fitting mejorado, a la derecha las piezas ensambladas una vez aplicada la técnica.

4.4.3. Snap-fitting por zonas

Para establecer el ensamble entre dos piezas, se utiliza una técnica inspirada en el snap-fitting mejorado utilizado en [49], en dicha técnica el ensamble está dado por dos piezas (pieza primaria y pieza receptora o secundaria), ambas piezas poseen un vector de ensamble con su respectiva posición de origen y de extremo, cuando las posiciones de los orígenes y de los extremos de ambos vector satisfacen una determinada restricción, estas se ensamblan automáticamente (Figura 4.35).

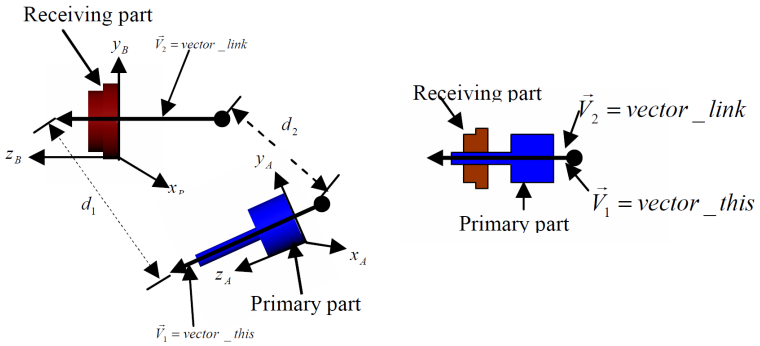

Figura 4.35: A la izquierda las piezas a ensamblar utilizando la técnica snap-fitting mejorado, a la derecha las piezas ensambladas una vez aplicada la técnica. Tomado de [49]

La diferencia en la técnica que se implementa, es que esta establece un área en la cual el vector de ensamble debe estar inmerso, si el vector de ensamble no se encuentra completamente inmerso en el área de ensamble, esta no se llevara a cabo. Esta técnica es implementada para no tomar en cuenta la orientación de las esferas utilizadas, sin embargo no se descarta que pueda ser utilizada en otro tipo de experimentos.

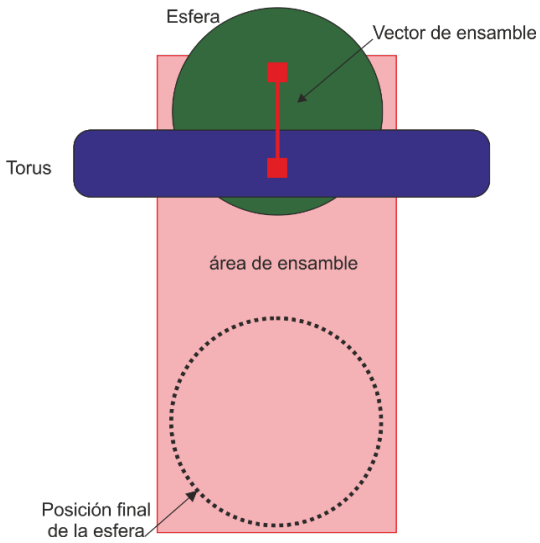

Figura 4.36: Proceso de ensamble en donde el vector de ensamble está completamente inmerso en el área de ensamble, por lo que se situara automáticamente en la posición final de ensamble.

Un problema que ocurre con los objetos rígidos utilizando la técnica snap-fitting mejorado o la versión modificada que se utiliza, es que en ocasiones el ensamble se puede llevar a cabo de manera poco realista como se muestra en la Figura 4.36, debido a que el vector de ensamble se encuentra totalmente inmerso en el área de ensamble, este sufrirá las transformaciones geométricas necesarias para llevarlo a su posición final.

La idea de utilizar los modelos rígido-deformables para esta tarea, se centra en que la simulación del ensamble ocurra con mayor realismo, y al contrario de los objetos rígidos, donde comúnmente las transformación se dan desde una posición muy distinta a la posición final de ensamble, la pieza receptora sea capaz de sufrir deformaciones que ayuden a que el objeto penetre a mayor profundidad y así reducir el área de ensamble y situarla en una posición que permita otorgar mayor realismo al momento de realizar el ensamble (Figura 4.37).

A partir de esto se realizaron distintos experimentos con distintos diámetros de esferas, desde diámetros menores al diámetro interno del torus, hasta diámetros mayores a este. En todos los casos el modelo logra atravesar el torus sin problemas, sin em-

bargo, se decidió utilizar esferas lo suficientemente grandes (hasta un diámetro de 17 unidades) para que la deformación de este fuera lo más realista posible (Figura 4.38).

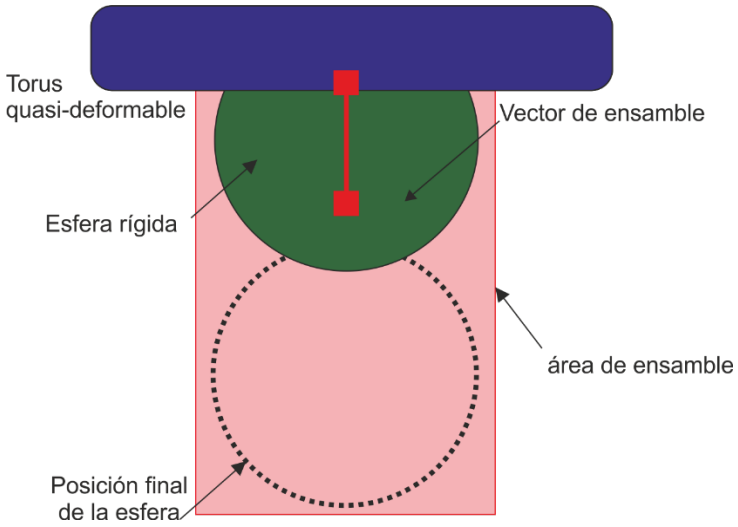

Figura 4.37: Proceso de ensamble en donde el vector de ensamble está completamente inmerso en el área de ensamble, por lo que se situara automáticamente en la posición final de ensamble, pero por ser el interior del contenedor estos movimientos no serán percibidos por el usuario.

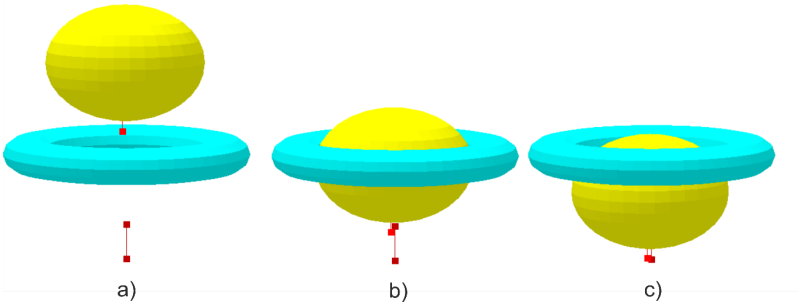

Figura 4.38: Etapas durante el proceso de ensamble. La esfera deforma al diámetro interno del torus para lograr pasar a través de el.

4.4.4. Conclusiones de la cuarta etapa

Una de las ventajas de utilizar modelos rígido-deformables para el apoyo de ensamble virtual es que se pueden mejorar técnicas como el "snap-fitting mejorado" debido a que los modelos pueden sufrir deformaciones, por lo que las restricciones impuestas en los vectores pueden verse reducidas, o incluso en tamaño de los vectores puede cambiar para evitar ensambles poco creíbles. Además de esto, si los objetos modelados también pueden sufrir deformaciones en la vida real, esto otorgaría mayor realismo durante el proceso.

Algo que afecta el uso de estos modelos, es el rendimiento de la aplicación en un momento donde existan demasiados contactos entre las partes a ensamblar. Aunque en este experimento no se centra en medir el rendimiento de las aplicaciones, es necesario realizar un análisis sobre cuanto disminuyen los FPS de la aplicación utilizando modelos rígido-deformables con respecto a los modelos rígidos. Debido a que la técnica que se utiliza en el detector de colisiones y en el motor de física Physx™ son distintas, debido a que una se enfoca en detectar colisiones en modelos rígidos y la nuestra en modelo con regiones deformables es injusto realizar una comparación en el rendimiento de la aplicación, es por esto que los experimentos se centran en determinar si la esfera atraviesa o no al torus.

Debido al ajuste que otorgan las esferas, nuestra aplicación es indudable que exista un mayor número de comparaciones, por lo que el rendimiento de esta se ve afectado. Es necesario considerar el uso de otro tipo de BV con mayor ajuste y observar el rendimiento de la aplicación.

Capítulo 5

Conclusiones generales y trabajo futuro

Se ha propuesto un modelo que permite representar deformaciones en una o varias zonas de los objeto aumentando el rendimiento en comparación con el uso de un modelo completamente deformable. El rendimiento que presenta del modelo propuesto cuando no existe interacción alguna con él es similar al de un modelo rígido, por lo que su implementación en ambientes donde no habrá interacción alguna con los modelos puede representar una opción viable, esto debido a que el rendimiento será prácticamente el mismo. Cuando es necesario representar deformaciones en los objetos, se demostró que este tipo de modelo permite un rendimiento hasta 2.5 veces mayor cuando las zonas de deformación son pequeñas, por lo que esto representa una alternativa a los modelos deformables convencionales.

Cuando se necesita realizar simulaciones que precisen de un alto grado de realismo, el modelo propuesto no tendrá un mejor desempeño en comparación a los modelos deformables, esto por distintas razones:

1. Cuando existen los suficientes puntos de contacto, y a su vez la profundidad de deformación de estos en el modelo propuesto es lo suficientemente grande para que todos los resortes estén activos, el rendimiento del modelo es inferior

al modelo MRA.

2. Debido a que las zonas donde los resortes se encuentran inactivos no ejercen ningún cambio a sus masas asociadas, debido a que en las simulaciones del modelo deformable existen cambios pequeños en las masas distantes a las zonas de deformación, en el modelo propuesto estos cambios se descartaran.

3. Debido a que el modelo propuesto ignora algunos cambios en las masas del mismo, las simulaciones que precisen de un alto nivel de detalle en las deformaciones descartaran el uso de este tipo de modelo.

Durante la primera etapa del desarrollo del modelo propuesto, se obtiene la región y la cantidad de resortes que ejerzan una fuerza lo suficientemente grande para considerar que estos representan cambios representativos en el modelo. En este caso la deformación aplicada al modelo constaba en aumentar la velocidad de una masa en dirección del eje x, lo que provocaba que esta cambiara de posición. Debido a que la aceleración aplicada era lo suficientemente grande para que el modelo sufriera deformaciones notorias, fue posible observar una gran cantidad de resortes cuya longitud superaba el umbral de deformación que se estableció, por lo que el reducir la aceleración aplicada resultaba en cambios tan pequeños en el modelo que solo una pequeña cantidad de resortes pasaban este umbral (los más cercanos a la masa). Esto nos dio un respaldo aun mayor para establecer zonas de deformación y distintos niveles de profundidad de deformación.

Uno de los puntos clave en los modelos deformables es que estos recuperen su forma. Para esto existen diversas técnicas, donde la mayoría de estas constan en utilizar una mayor cantidad de masas y resortes en estructuras con forma de cubos que utilizan resortes internos para garantizar que estas no pierdan su forma. Sin embargo este tipo de método es utilizado especialmente en modelos volumétricos donde existe una gran cantidad de masas y resortes que el usuario no observa a simple vista, a comparación de los modelos que se implementan en este trabajo, los cuales son representaciones de la parte exterior de un objeto, y que por dentro estos son huecos. Debido a que estas técnicas para preservar la forma son sumamente costosas por la gran cantidad de resortes que son necesarios de calcular, el método basado en restricciones representa

una alternativa cuyo costo es mínimo en comparación a la antes mencionada, esto se debe a que no se implementan nuevos resortes y que solo es necesario realizar un cálculo extra en los resortes ya existentes siempre y cuando la restricción se cumpla.

Durante la fase del diseño del sistema detector de colisiones para los modelos propuestos es posible realizar mejoras, entre las cuales se encuentra:

- Uso de un BV con mayor ajuste
- Permitir el uso de cualquier malla poligonal, sin importar el tamaño de los triángulos que la compongan.

El uso de esferas como BV fue de gran utilidad debido a la rapidez de las pruebas de intersección de estas, sobre todo cuando los objetos se encuentran lo suficientemente separados entre sí, resulta casi inmediato determinar que no existe colisión. Sin embargo cuando los objetos se encontraban lo suficientemente cerca, la cantidad de pruebas aumentaba considerablemente, esto empeoraba si algún modelo contaba con caras cuyo tamaño fuera muy grande, debido a que el BV de esta alcanzaba a abarcar a una gran cantidad de triángulos que no colisionaban con este, y que de cualquier manera se realizaba la prueba de intersección.

Se considera que el uso de un BV con mayor ajuste resolverá este problema, el cual no representa un mal funcionamiento del sistema detector de colisiones, si no que representa una disminución en el rendimiento de la aplicaciones debido a las características del BV.

Por ultimo durante la experimentación del ensamble de dos piezas, la técnica implementada en conjunto con el modelo propuesto permite aumentar la calidad visual durante la fase final del proceso de ensamble, esto debido a que las piezas primarias pueden estar lo suficientemente cerca de su posición final de ensamble que no es necesario realizar transformaciones de gran magnitud sobre esta, logrando un mejor resultado visual durante este proceso.

Debido a que los resultados obtenidos con el modelo propuesto han sido satisfactorios

incluso si los cálculos estese hacen completamente utilizando el CPU. A partir de los resultados obtenidos y al observar otros trabajos que utilizan GPUs en conjunto con técnicas de paralelización se considera que estos puede incrementar el rendimiento de la aplicación notoriamente, por lo que realizar esta tarea es uno de los planes para continuar con este trabajo.

Otro punto importante es el mejorar el rendimiento durante las colisiones, aunque de momento se utilizan BHV con esferas, es posible que el utilizar otro tipo de división del entorno virtual como quadtrees u octrees aumenten el rendimiento de esta, sumando a esto la elección de un mejor BV. Por lo que realizar esta investigación es una de la tareas que se pretenden llevar a cabo para mejorar el trabajo presentado.

Capítulo 6

Anexos

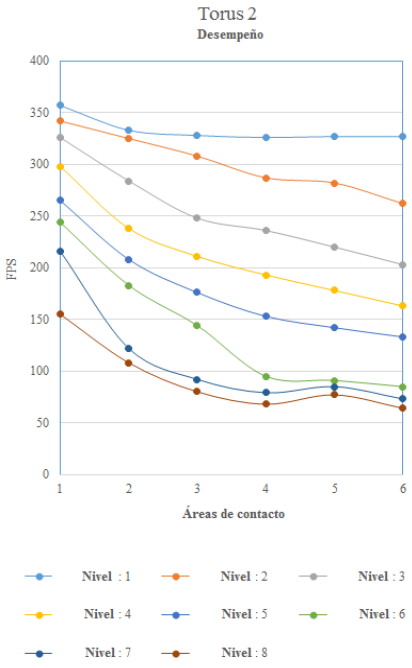

Figura 6.1: Desempeño del modelo torus 2 con distintos puntos de contacto y niveles de profundidad de deformación .

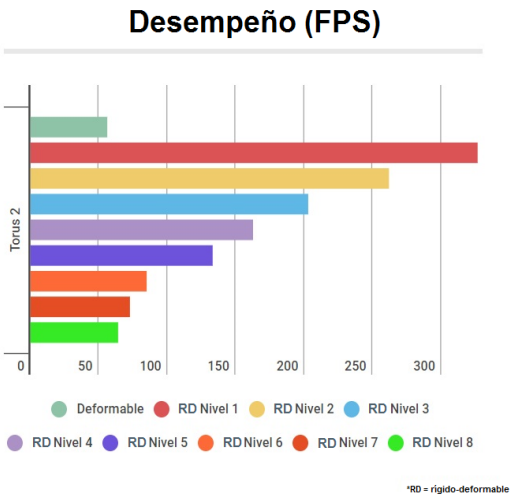

Figura 6.2: Rendimiento del modelo (torus 2).

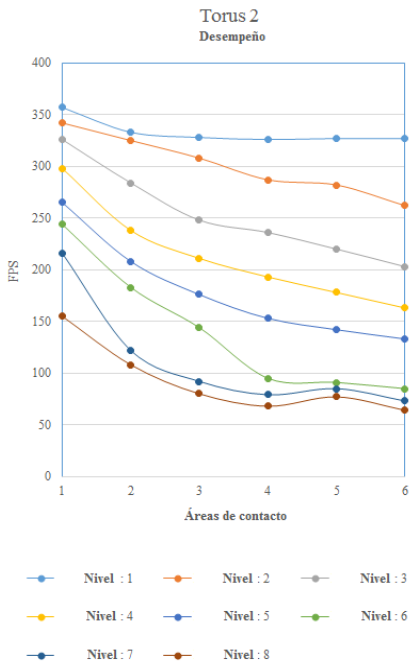

Figura 6.3: Desempeño del modelo Corazón con distintos puntos de contacto y niveles de profundidad de deformación .

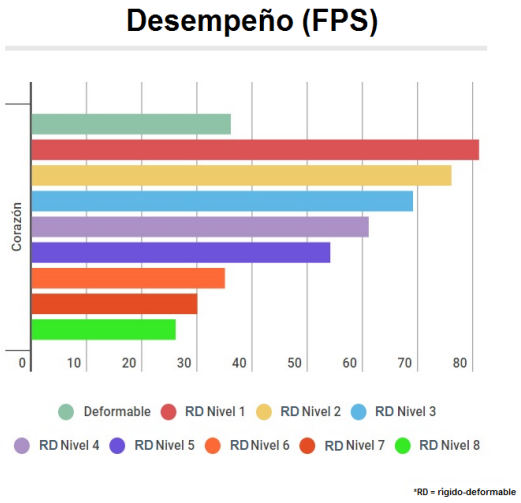

Figura 6.4: Rendimiento del modelo (Corazón).

Bibliografía

[1] Dan Albocher, Uzi Sarel, Yi-King Choi, Gershon Elber, y Wenping Wang. Efficient continuous collision detection for bounding boxes under rational motion. En *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, págs. 3017–3022. IEEE, 2006.

[2] Srikanth Bandi y Daniel Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. En *Computer Graphics Forum*, tomo 14, págs. 259–270. Wiley Online Library, 1995.

[3] A Bond. Havok fx: Gpu-accelerated physics for pc games. En *Proceedings of Game Developers Conference 2006*. 2006.

[4] David Bourguignon y Marie-Paule Cani. Controlling anisotropy in mass-spring systems. En *Computer animation and simulation*, tomo 2000, págs. 113–123. Springer, 2000.

[5] Jonathan D Cohen, Ming C Lin, Dinesh Manocha, y Madhav Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. En *Proceedings of the 1995 symposium on Interactive 3D graphics*, págs. 189–ff. ACM, 1995.

[6] Lee Cooper. Preventing collapse within mass-spring-damper models of deformable objects. 1997.

[7] Antonino Gomes De Sa y Gabriel Zachmann. Virtual reality as a tool for verification of assembly and maintenance processes. *Computers & Graphics*, 23(3):389–403, 1999.

[8] Alain Delchambre. A pragmatic approach to computer-aided assembly planning. En *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, págs. 1600–1605. IEEE, 1990.

[9] Richard G Dewar, Ian D Carpenter, James M Ritchie, y John EL Simmons. Assembly planning in a virtual environment. En *Innovation in Technology Management-The Key to Global Leadership. PICMET'97: Portland International Conference on Management and Technology*, págs. 664–667. IEEE, 1997.

[10] Christer Ericson. *Real-time collision detection*. CRC Press, 2004.

[11] Emil Eriksson. Simulation of biological tissue using mass-spring-damper models. 2013.

[12] Charbel Fares y Yskandar Hamam. Collision detection for rigid bodies: A state of the art review. *GraphiCon 2005*, 2005.

[13] Philippe Fuchs, Guillaume Moreau, y Pascal Guitton. *Virtual reality: concepts and technologies*. CRC Press, 2011.

[14] Samir Garbaya y Ulises Zaldivar-Colado. Modeling dynamic behavior of parts in virtual assembly environment. *interface*, 8:9, 2009.

[15] Sarah FF Gibson y Brian Mirtich. A survey of deformable modeling in computer graphics. Inf. téc., Technical Report, Mitsubishi Electric Research Laboratories, 1997.

[16] Germanico Gonzalez-Badillo, Hugo I Medellin-Castillo, y Theodore Lim. Development of a haptic virtual reality system for assembly planning and evaluation. *Procedia Technology*, 7:265–272, 2013.

[17] Stefan Gottschalk, Ming C Lin, y Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. En *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, págs. 171–180. ACM, 1996.

[18] Naga K Govindaraju, Stephane Redon, Ming C Lin, y Dinesh Manocha. Cullide: Interactive collision detection between complex models in large environments using graphics hardware. En *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, págs. 25–32. Eurographics Association, 2003.

[19] H Hees. 3d computer graphics. 2007.

[20] Philip M Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics (TOG)*, 15(3):179–210, 1996.

[21] Hayley Iben, Mark Meyer, Lena Petrovic, Olivier Soares, John Anderson, y Andrew Witkin. Artistic simulation of curly hair. En *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, págs. 63–71. ACM, 2013.

[22] Sankar Jayaram, Uma Jayaram, Yong Wang, Hrishikesh Tirumali, Kevin Lyons, y Peter Hart. Vade: A virtual assembly design environment. *IEEE Computer Graphics and Applications*, 19(6):44–50, 1999.

[23] Eric Larsen, Stefan Gottschalk, Ming C Lin, y Dinesh Manocha. Fast proximity queries with swept sphere volumes. Inf. téc., Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999.

[24] Achim Loock, Elmar Schömer, y Im Stadtwald. A virtual environment for interactive assembly simulation: From rigid bodies to deformable cables. En *5th World Multiconference on Systemics, Cybernetics and Informatics (SCI'01)*, tomo 3, págs. 325–332. 2001.

[25] Tim McInerney y Demetri Terzopoulos. Deformable models in medical image analysis: a survey. *Medical image analysis*, 1(2):91–108, 1996.

[26] William A McNeely, Kevin D Puterbaugh, y James J Troy. Voxel-based 6-dof haptic rendering improvements. 2006.

[27] Brian Mirtich. Efficient algorithms for two-phase collision detection. *Practical motion planning in robotics: current approaches and future directions*, págs. 203–223, 1997.

[28] Arlindo N Montagnoli, José B Rubert, Flavio Y Watanabe, Osmar Ogashawara, y JC Pereira. Computational simulations in mass-spring dynamic system to the development of vocal folds tissues models. En *Brazilian Conference on Dynamics, Control and their Applications*, págs. 1–6. 2010.

[29] Matthias Müller, Bruno Heidelberger, Marcus Hennix, y John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[30] Bruce F Naylor. Interactive solid geometry via partitioning trees. En *Proc. Graphics Interface*, tomo 92, págs. 11–18. 1992.

[31] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, y Mark Carlson. Physically based deformable models in computer graphics. En *Computer graphics forum*, tomo 25, págs. 809–836. Wiley Online Library, 2006.

[32] James L Nevins y Daniel E Whitney. Assembly research. *Automatica*, 16(6):595–613, 1980.

[33] PhysX NVIDIA. Physics simulation for developers. 2009.

[34] Mike Oren, Patrick Carlson, Stephen Gilbert, y Judy M Vance. Puzzle assembly training: Real world vs. virtual environment. En *Virtual Reality Short Papers and Posters (VRW), 2012 IEEE*, págs. 27–30. IEEE, 2012.

[35] Mark Pauly, Dinesh K Pai, y Leonidas J Guibas. Quasi-rigid objects in contact. En *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, págs. 109–119. Eurographics Association, 2004.

[36] Madhav K Ponamgi, Dinesh Manocha, y Ming C Lin. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):51–64, 1997.

[37] Michael J Pratt. Virtual prototypes and product models in mechanical engineering. En *Virtual Prototyping*, págs. 113–128. Springer, 1995.

[38] Xavier Provot et al. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. En *Graphics interface*, págs. 147–147. Canadian Information Processing Society, 1995.

[39] Hong Qin y Demetri Terzopoulos. D-nurbs: A physics-based geometric design framework. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):85–96, 1996.

[40] Ralph Schroeder. Virtual reality in the real world: history, applications and projections. *Futures*, 25(9):963–973, 1993.

[41] Andrew Selle, Michael Lentine, y Ronald Fedkiw. A mass spring model for hair simulation. *ACM Transactions on Graphics (TOG)*, 27(3):64, 2008.

[42] Abhishek Seth, Hai-Jun Su, y Judy M Vance. Sharp: a system for haptic assembly and realistic prototyping. *ASME Paper No. DETC2006/CIE-99476*, 2006.

[43] Demetri Terzopoulos. On matching deformable models to images. En *Topical meeting on machine vision*, tomo 12, págs. 160–167. 1987.

[44] Demetri Terzopoulos y Keith Waters. Physically-based facial modelling, analysis, and animation. *Computer Animation and Virtual Worlds*, 1(2):73–80, 1990.

[45] Hank Weghorst, Gary Hooper, y Donald P Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics (TOG)*, 3(1):52–69, 1984.

[46] René Weller. A brief overview of collision detection. En *New Geometric Data Structures for Collision Detection and Haptics*, págs. 9–46. Springer, 2013.

[47] Mason Woo, Jackie Neider, Tom Davis, y Dave Shreiner. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[48] Gabriel Zachmann. Optimizing the collision detection pipeline. En *Proc. of the First International Game Technology Conference (GTEC)*, tomo 2. 2001.

[49] Ulises Zaldivar-Colado y Samir Garbaya. Virtual assembly environment modelling. En *ASME Conference Proceedings*, págs. 157–163. 2009.

[50] Tian Zhou y Yue Qi. Deformation-aided virtual assembly system for mechanical structure. En *Virtual Reality and Visualization (ICVRV), 2011 International Conference on*, págs. 63–69. IEEE, 2011.