UNIVERSIDAD DE LAS CIENCIAS INFORMATICAS

Facultad 5

“Técnicas de corte en mallas triangulares superficiales.”

Trabajo de Diploma para optar por el Título de

Ingeniero en Ciencias Informáticas

Autora: Yaima Soto Avello

Tutores: Ing. Raissel Ramírez Orozco

Ing. Lester O. Rodríguez González

Ciudad de la Habana

15 de Junio 2009

Datos de Contacto

Nombre y Apellidos: Raissel Ramírez Orozco

Edad: 25 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas

Título: Ingeniero en Ciencias Informáticas.

Categoría Docente: Profesor Adiestrado

E-mail: rramirez@uci.cu

Graduado en la Universidad de las Ciencias Informáticas, 4 años de experiencia en proyectos de Realidad Virtual.

Nombre y Apellidos: Lester Oscar Rodríguez González

Edad: 24 años

Ciudadanía: cubano

Institución: Universidad de las Ciencias Informáticas

Título: Ingeniero en Ciencias Informáticas.

Categoría Docente: Profesor Adiestrado

E-mail: lorodriguez@uci.cu

Graduado en la Universidad de las Ciencias Informáticas.

Dedicatoria

… a mi mami, a nanita, las amo.

Yaima

Agradecimientos

A una persona muy especial, por todo su sacrificio, su constante apoyo y preocupación, su amor infinito, eres la luz de mis ojos. Gracias un millón. Te amo: Mami.

A nanita porque te adoro tal y como eres y sé que tú también a mi.

A mi abuelita Angela por siempre estar al pendiente, por tanto cariño.

A tata por estar ahí todos estos años.

A tatico (Osley) por su guía, sus consejos, su constante apoyo y preocupación, su optimismo, su entrega y su amor. Gracias por estos dos años felices y los que vendrán.

A mis tíos, tías políticas, primos gracias por todo.

A los nuevos miembros de mi familia: mis suegros, mi cuñadita, por acogerme con tanto cariño, por preocuparse por mí.

A mis amistades, a todos mis compañeros del grupo anterior, a los que hoy ya no están en la UCI, a los de este nuevo grupo, muy especialmente agradecerle a las muchachitas: Papita, Yadira, Lisy, Mirthica, Eli, Diana, Ari por soportarme y apoyarme, muy especial a Yay por todo su cariño. A los varones Yoan, Arcel, por compartir juntos estos cinco años, por ayudarme y apoyarme. A todos y cada uno los llevaré siempre en el corazón.

A mis tutores: Raissel y Lester por su apoyo y su amistad.

A todas las personas que de alguna forma estuvieron implicadas en el desarrollo de este trabajo y en el cursar de mi carrera.

Resumen

Se vislumbra una crisis en el campo del entrenamiento quirúrgico actual, según aumenta la complejidad de las técnicas, los cirujanos requieren de una prática más extendida y las oportunidades se reducen. La solución parece ser el desarrollo de simuladores virtuales, donde el proceso de corte se presenta como un requerimiento indispensable para lograr un verdadero entrenamiento. El presente trabajo tiene como objetivo brindar dos técnicas de corte para modelos superficiales, específicamente mallas triangulares, una de ellas basa su algoritmo en destruir el elemento colisionante con la herramienta virtual del corte y la otra consiste en separarlo.

Se ha realizado un estudio para conocer los principales trabajos enfocados a las diferentes técnicas de corte a nivel mundial, las cuales presentan tres tendencias fundamentales: la destructiva, la de separación y la de subdivisión.

Por último se propone un módulo, el cual cuenta con las dos técnicas de corte descritas anteriormente y se ha desarrollado desarrollado usando el proceso unificado de software e implementado en c++ estándar.

Palabras claves

Simuladores virtuales, técnicas de corte, modelos superficiales, mallas triangulares.

Tabla de contenido

Introducción

Las técnicas de realidad virtual aparecen ante los ojos del mundo como un medio más de la entrada de la informática en la enseñanza. Dichas técnicas están abriendo paso a nuevas formas de aprendizaje donde el alumno participa activamente, ya sea de forma visual, auditiva, entre otras. Así, para Tiffin y Rajasinghan: "la unión de las tecnologías informáticas y de las telecomunicaciones podrían hacer de la clase virtual el principal lugar de aprendizaje en la sociedad" [1].

En los últimos decenios la medicina se ha vinculado estrechamente con la tecnología, siendo visible en el desarrollo de simuladores virtuales para el aprendizaje y entrenamiento de los estudiantes y médicos del área de cirugía respectivamente, con el fin de mejorar la calidad de vida de sus pacientes. El reto fundamental de estos simuladores consiste en aparentar el mundo real, conservando las propiedades físico - matemáticas de los objetos y permitan de una forma natural interactuar con ellos permitiendo el mayor realismo posible.

El corte es una de las habilidades fundamentales que todo cirujano debe adquirir, es un proceso que se observa constantemente en las cirugías, por tanto, una de las tareas básicas de la simulación quirúrgica lo constituye el fenómeno virtual del corte, este puede ser considerado como la interacción de un modelo virtual deformable y un cuerpo rígido, visto como herramienta de corte. En este proceso, el modelo deformable es cortado y su topología geométrica sufre cambios. Esta habilidad solo es posible desarrollarla con la práctica, pero si se realiza con pacientes el riesgo es considerable, por tal motivo el empleo de simuladores brinda la herramienta para que el cirujano pueda aprender a realizar cortes en los órganos sin herir a nadie, también pueden emplearse animales para esta práctica pero el costo es elevado y la anatomía de estos es diferente.

En la universidad de las Ciencias Informáticas (UCI) se realiza toda una investigación y un proceso de producción como bandera del programa de informatización, que se lleva a cabo en nuestro país. La facultad 5 de la misma, contiene en su perfil investigativo y productivo a la realidad virtual, donde se cuenta con un motor gráfico llamado *Scene Toolkit* (STK), con funcionalidades para producir entornos virtuales en corto tiempo, desarrollada por el grupo "Herramientas de desarrollo para sistemas de realidad virtual".

En esta facultad se está desarrollando un simulador quirúrgico (Kheipros), este posee una de las técnicas de corte existentes, pero esto no resulta totalmente suficiente, por tanto, la **problemática** radica en la necesitad de nuevas técnicas de corte para lograr mayor realismo al interactuar con un simulador.

Partiendo de esta situación el **problema científico** a resolver sería: ¿Cómo desarrollar nuevas técnicas de corte para emplearlas en un simulador quirúrgico, por ejemplo, en Kheipros, el que está desarrollando la facultad 5 de la UCI?

El **objetivo general** del trabajo es desarrollar nuevos métodos de corte, que permitan la posibilidad de seleccionar la técnica de corte idónea en dependencia de los requerimientos del ejercicio a desarrollar, garantizando mejores resultados al ser empleados en un simulador.

La ciencia que estudia este tema es muy amplia por tanto se debe centrar como **objeto de estudio** el comportamiento topológico de los objetos virtuales al someterlos al proceso de corte y como **campo de acción** las técnicas y algoritmos empleados para la simulación de corte.

Al culminar el trabajo se pretende contar con un módulo parcial o totalmente implementado que permita seleccionar la técnica de corte a emplear y que le proporcione al programador una manera viable de representar objetos, que reaccionen ante fuerzas externas en el proceso de corte, obedeciendo parámetros reales, es decir, que un objeto al ser interceptado por la herramienta virtual de corte simule lo más real posible, cambios en su topología, siguiendo la base del método seleccionado para cortar. Esto aporta una de las funcionalidades más importantes en los simuladores.

Para cumplir el objetivo planteado y satisfacer la necesidad ya conocida, es necesario ejecutar las **tareas de investigación** que se relacionan a continuación:

- Analizar la bibliografía sobre el tema en los medios de información disponibles.
- Seleccionar las técnicas a desarrollar.
- Describir los métodos que se implementarán para la solución.
- Identificar los algoritmos para la detección de colisiones.
- Aplicar la metodología de desarrollo de software RUP.
- Analizar las características de la STK para la visualización de la solución.
- Desarrollar parcial o totalmente un módulo para solucionar el problema planteado.

Para una mejor comprensión, el documento está dividido en capítulos que estructuran el contenido de la siguiente forma:

En el capítulo uno “Fundamentación Teórica”, se hace un exhaustivo análisis de las técnicas usadas para la simulación de corte en objetos deformables, desde sus inicios hasta nuestros días.

El capítulo dos “Soluciones Técnicas”, establece la solución resultante del análisis realizado en el capítulo anterior y se proponen las técnicas de corte a utilizar.

En el capítulo tres “Descripción de la Solución Propuesta”, se describe el sistema a desarrollar desde la perspectiva de las necesidades del cliente, en función de las dificultades y características del cliente.

El capítulo cuatro “Diseño e Implementación del Sistema”, corresponde a los flujos de trabajo de diseño e implementación de RUP, se describen las clases de diseño, se relacionan mediante el diagrama de clases de análisis y se distribuyen en componentes de software como indica el diagrama de componentes del sistema.

Capítulo 1: Fundamentación teórica.

Introducción

La realización del corte virtual es una tarea de gran complejidad, con el fin de lograr desarrollarla con el mejor realismo posible se realizó una revisión minuciosa de toda la gama de documentación existente correspondiente al tema en cuestión, realizada por diferentes científicos de todo el mundo y que hasta hoy aún continúa en estudio.

En este capítulo se expone el marco teórico conceptual en el cual se encuentra inmerso este trabajo, el mismo se divide en epígrafes donde se tratan temas como la representación geométrica de los objetos virtuales, las principales tendencias que existen para la simulación del corte, los métodos de deformación basados en física y las fases para la detección de colisiones, estos a su vez se subdividen para detallar con mayor profundidad los temas anteriores. Logrando con esto que sea de mejor entendimiento el presente trabajo para todos, incluso aquellos que no estén vinculados al campo de la informática.

Visión general

Es imprescindible que un médico cirujano adquiera soltura y precisión. El principal modo que tiene para obtener la fiabilidad requerida es participar en numerosas operaciones con el consiguiente riesgo para el paciente, en muchas ocasiones verdaderos “conejillos de indias”. Dentro de las tareas comunes que posee un cirujano se encuentra cortar, este es un proceso donde no hay vuelta atrás, por tanto, debe estarse seguro de cuando y como cortar, además el corte es vital tanto en la cirugía abierta como en la mínimamente invasiva.

Son numerosos los avances en el campo de esta cirugía antes mencionada, cítense los recientes adelantos en la cirugía cardiaca empleando procedimientos mínimamente invasivos para la desviación de la arteria coronaria, hay que destacar que el período de recuperación de esta cirugía es menor, por tanto los costos también lo son, así como el dolor y las cicatrices que experimenta el paciente. Sin embargo, para el cirujano, la cirugía mínimamente invasiva presenta dificultades en cuanto a la restricción de visión y movimientos, la dificultad en el manejo del instrumental así como la coordinación entre la mano y el ojo y la ausencia de percepción táctil. A pesar de estas desventajas, este tipo de cirugía se está convirtiendo en la más empleada y popular, lo cual puede apreciarse en la creación de simuladores quirúrgicos endoscópicos por las grandes compañías de realidad virtual en el campo de la medicina, con un enfoque de entrenamiento [2].

Los simuladores parecen ser la solución a los problemas que enfrenta el área de cirugía en cuanto al entrenamiento sumamente necesario, estos les permiten a los cirujanos practicar procedimientos particularmente difíciles bajo el control del ordenador. El surgimiento de los simuladores es muy joven aún y su mayor dificultad radica en lograr el mayor realismo posible, fundamentalmente en el sistema de sentido táctil, aunque en los últimos años se aprecian progresos en este aspecto. Cabe destacar que se emplean instrumentos quirúrgicos exteriormente idénticos, gran ventaja para sentir el ambiente más real [1].

Antecedentes

El acercamiento al corte en la simulación quirúrgica mayoritariamente va acoplado con la deformación y visualización de una malla. Con algunas excepciones, las mallas están compuestas por sus unidades mínimas en un triángulo, en el caso de 2 dimensiones o un tetraedro, en 3 dimensiones. La idea principal del primer algoritmo que surge en este campo es eliminar los elementos que tengan contacto con el instrumento de corte, planteado por Bro-Nielsen [3] y S. Cotin [4]. Este método es retomado por Forest y otros [5], quienes tratan el refinamiento local de la malla con un algoritmo más eficiente.

Fig. 1: Ejemplo de un tetraedro eliminado por colisión con la herramienta de corte.

Más tarde surgen los llamados métodos de subdivisión, los cuales predefinen los estados posibles del corte y luego los ejecutan de acuerdo el correspondiente en cada caso. Se introducen en el contexto médico a través del trabajo correspondiente a Mazura y Seifert [6], donde plantean el corte de una malla tetraédrica con planos predefinidos. Este algoritmo fue refinado más tarde por Mor y T. Kanade [7], dando grandes pasos de avances pues se tratan temas como el corte progresivo y la incisión parcial. Bielser en [8] plantea una máquina de estados.

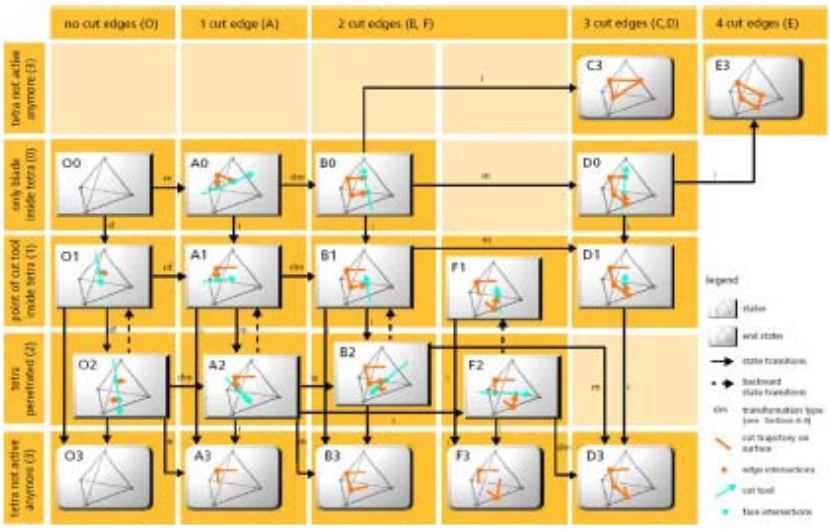


Fig. 2: Diagrama de una máquina de estados.

Un aspecto sumamente negativo es la inclusión de nuevos elementos, los cuales se han reportado entre 5 y 17 por tetraedro interceptado en mallas volumétricas, esto provoca que se extiendan los cálculos en el modelo. Además según reportes realizados en el trabajo de Mor y T. Kanade [7] se reduce el tamaño de los tetraedros, lo que implica problemas en la estabilidad de la deformación.

Otra idea acerca de los procedimientos de corte surge con Nienhuys [9], para simular el corte moviéndose al vértice más cercano al punto donde es interceptado el modelo por la herramienta de corte, Mendoza y Laugier [10] generalizan este método para desplazamientos largos de la herramienta virtual.

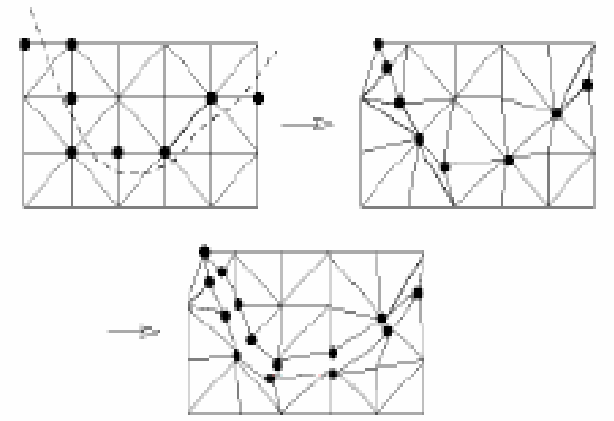


Fig. 3: Moviéndose al vértice más cercano.

En la adaptación realizada por Servy y otros [11] fueron reducidas algunas de estas dificultades, pero desafortunadamente surge otro problema, que es la degeneración de los elementos cuando el instrumento de corte atraviesa la malla cerca de un nodo. Posteriormente emergen las ideas de Bruyns y Montgomery [12], donde plantean una solución que permite el trabajo con múltiples capas.

A pesar de esta amplia realización de estudios en este campo de la cirugía, específicamente en el proceso de corte, aún no se cuenta con un trabajo que reúna dos de las técnicas tratadas por varios autores mencionados anteriormente y es ese el objetivo de esta investigación, proponer dos métodos de corte sobre mallas superficiales.

1.3 Representación geométrica

Existen diferentes objetos que conforman una escena virtual para proporcionar una mejor visualización de los elementos reales, estos conforman los distintos tipos de modelos de representación geométrica, dentro de los que podemos citar los modelos alámbricos, de superficie, volumétricos, sólidos, etc.

1.3.1 Modelos de superficie

Dentro de esta clasificación encontramos los modelos poligonales y los modelos de superficies de curvas, así como la representación por mallas, la cual es una de las técnicas de modelado más populares hoy día. Se encuentran además los modelos basados en puntos, los cuales han ganado una creciente atención como una alternativa de la representación de superficies en el mundo.

1.3.1.1 Mallas triangulares

Una malla conceptualmente es una colección de vértices, aristas y polígonos conectados de tal forma que cada arista es compartida como máximo por dos polígonos. Las mallas pueden clasificarse en Estructuradas o No Estructuradas. En el primer caso la conectividad puede ser descrita por algún esquema de indexado; mientras que en el segundo esa relación no existe y se hace necesario una estructura de datos especial para representar la información de dicha conectividad [13].

En gráficos por computadora se emplean ampliamente las representaciones por mallas, principalmente las mallas triangulares por ser las más populares pues se caracterizan por ser fáciles de *renderizar* [14] , sin embargo cabe mencionar otras como las tetraédricas, las poligonales, etc.

"La mayoría de los 3D *engines* (tanto hardware como software) sólo se relacionan con triángulos. Las razones son muchas, entre otras, los triángulos son fáciles de representar en sistemas gráficos, tres puntos definen un plano y muchos de los algoritmos 3D trabajan mejor con mallas triangulares." [15].Las mallas triangulares consisten en un conjunto de puntos con coordenadas 3D, que representan una superficie, y una estructura que describe como estos puntos son conectados por triángulos [16]. Estas mallas almacenan básicamente las coordenadas de los vértices que la componen y los datos de conectividad que los enlazan. Para la mayoría de las mallas, el número de triángulos es aproximadamente el doble del número de vértices [17].

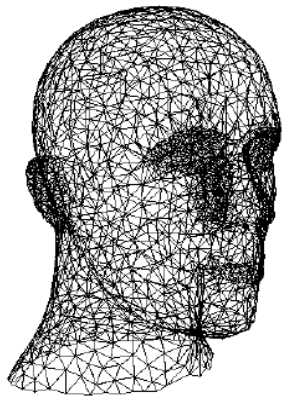


Fig. 4: Representación de un cuerpo mediante mallas triangulares.

1.3.1.2 Modelo basados en puntos

La mayoría de las aplicaciones de simulación tanto 2D como 3D en las que intervienen deformaciones, están basadas en mallas, tanto Sistemas Masa-Resorte, como sistemas motivados por métodos matemáticos como Método de Elementos Finitos (*Finite Element Method FEM*) o Elementos de Frontera (*Boundary Element Method BEM*), conjugados con la teoría de la elasticidad. Sin embargo, las aplicaciones basadas en puntos se han convertido en una popular técnica en el mundo de los gráficos por computadoras. Estas surgen con el objetivo de lograr un bajo costo del sistema en escenas que contienen gran cantidad de triángulos.

Una representación geométrica basada en puntos, puede ser considerada como un muestreo de una superficie continua (como se muestra en la Fig. 5), una colección no necesariamente regular de posiciones 3D, opcionalmente con vectores normales asociados o propiedades auxiliares como color u otras propiedades del material representado [18]. Es un gran reto técnico mantener la consistencia topológica de la malla subyacente cuando se trata de complejos efectos físicos como fracturas [19]. Hoy en día son varias las compañías que producen juegos de consola que utilizan esta técnica en la representación de personajes [20].También está siendo usada en la representación de objetos que se deforman mediante la técnica de *Free-Form Deformation* [21].

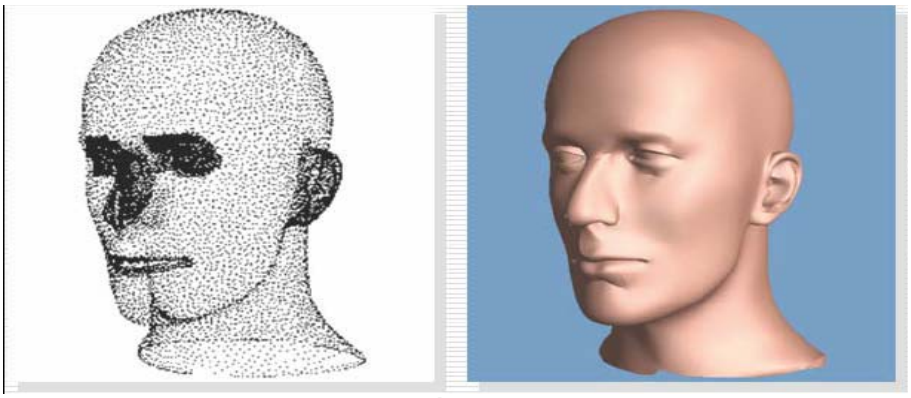


Fig. 5: Rostro representado a través de puntos.

Uno de los trabajos pioneros en este campo es el realizado por Levoy y Whitted en 1985 [22], años más tarde fue tratado por Grossman y Rally [23], en el año1998 y más reciente en el 2003 por Pauly M [24], así también Zwicker M [25] y su equipo en el 2004.

1.3.2 Mallas volumétricas

Los modelos de mallas volumétricas son ampliamente difundidos para la representación de órganos y reconstrucción 3D a partir de imágenes, estos simulan el comportamiento interior de los objetos, conformados por tetraedros, hexaedros o cubos para mencionar los más comunes. Lograr esta característica implica usar algoritmos de enmallado que permitan representar al cuerpo sólido. Los principales algoritmos para este proceso se dividen en algoritmo triángulo/tetraedro y algoritmo cuadrilátero/hexaedro. El primero es el más utilizado, por tanto en él encontramos las técnicas más usadas: Octree, Delaunay, Advancing Front, las cuales se describen a continuación.

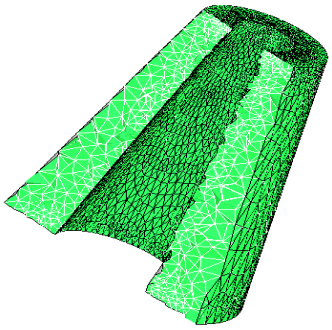


Fig. 6: Malla volumétrica.

Octree: Marck Shepard fue el primero en desarrollarla en 1980. Con este método, los cubos contenidos en el modelo geométrico son subdivididos recursivamente hasta alcanzar la resolución deseada.

Delaunay: Es la técnica más popular para generar mallas triangulares. Plantea que algún vértice de un tetraedro debe estar contenido en la esfera circunscripta, si se traza una esfera por los nodos de un tetraedro y ésta no contiene a todos cumple con el criterio de Delaunay. Así se garantiza una triangulación dado un conjunto finito de puntos.

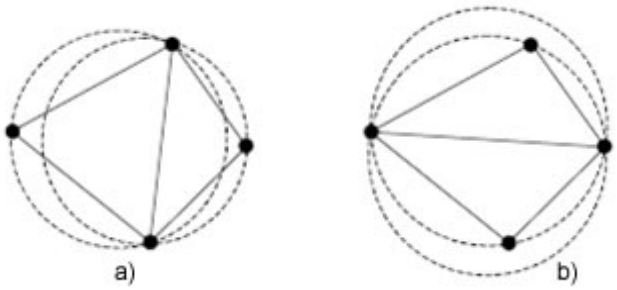


Fig. 7 : Criterio de Delaunay a) cumple condición b) no cumple condición.

Advancing Front: En este método el tetraedro es construido progresivamente hacia dentro desde la superficie triangulada del objeto. La figura 5 muestra un ejemplo en dos dimensiones de Advancing Front, donde los triángulos son formados desde la superficie, el algoritmo avanza desde el frente hasta rellenar todo el área del objeto con triángulos. En tres dimensiones, por cada cara triangular en el frente, es calculada la posición ideal de un cuarto nodo que formaría el tetraedro. El algoritmo selecciona el cuarto nodo creado o de un nodo existente para formar el nuevo tetraedro, basado en cual podría formar el mejor tetraedro. También son necesarios los controles de intersección entre tetraedros para garantizar que no se superpongan al avanzar uno hacia el otro.

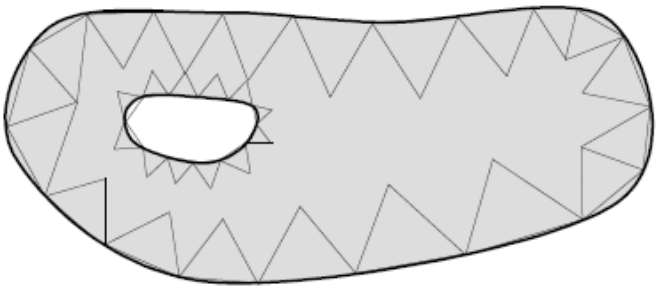


Fig. 8 : Representación de Advancing Front 2D.

1.4 Tendencias de Corte

El fenómeno virtual de corte es una de las tareas básicas fundamentales en la simulación quirúrgica. En este proceso el objeto deformable es cortado y su topología cambia. Los trabajos realizados acerca de este tema pueden clasificarse en tres métodos fundamentales, debido a la técnica de realización de estos las cuales se abordan a continuación:

- El método destructivo.
- El método de separación.
- El método de subdivisión.
- Corte progresivo con generación de ranura interna.

1.4.1 Método destructivo

Es el primer y más antiguo método conocido y empleado, su base consiste en eliminar el elemento que colisione con la herramienta virtual de corte.

Dentro de sus ventajas se encuentra que al eliminar aristas, nodos o tetraedros disminuye el número de elementos y a su vez el tiempo de simulación. Sin embargo, es recomendable emplearlo en aplicaciones donde se destruye realmente el tejido y además viola el principio físico de conservación de la masa. Es una dificultad realizar el corte con este método en el tejido blando, pues se requieren extensos procesos de refinamiento de la malla, sin embargo, en el trabajo realizado por Forest, Delingette Ayache [5] se aborda con profundidad el tema del refinamiento de la malla, específicamente para mallas volumétricas, donde primero remueven los tetraedros y luego refinan la malla alrededor del camino cortado, para que resulte más real las mallas deben refinarse dinámicamente, este tema puede apreciarse en un estudio realizado en el año 2000 [4].

En 1997 Cotin utilizó un modelo Masa-Resorte para la deformación y el corte [26], en este, el corte fue implementado eliminando los elementos que colisionaban con la herramienta virtual de corte. Años más tarde, en el trabajo realizado por S. Cotin, H. Delingette, y N. Ayache [4], retoman este método y describen tres diferentes modelos físicos, dentro de los cuales uno es muy similar al modelo masa-resorte y la operación de corte que se le realiza puede ser simulado en tiempo real. Además C Forest, H.Delingette y N. Ayache en su documento profundizan en este método para mallas volumétricas, describiendo todo el proceso de refinamiento de las mismas, planteando soluciones y dándose a la tarea fundamental de disminuir el tiempo computacional y lograr tetraedros de óptima calidad.

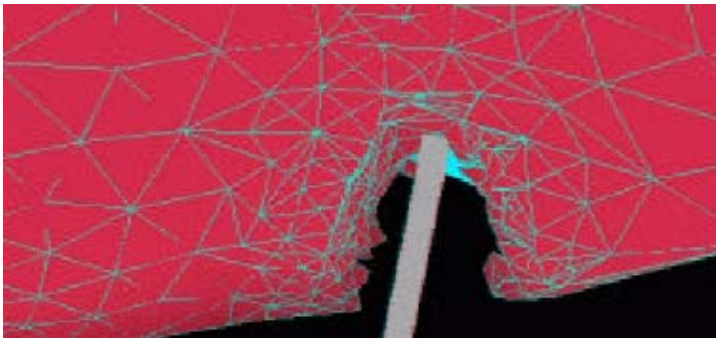


Fig. 9: Modelo para el corte: Se destruyen los tetraedros colisionados por la herramienta de corte.

1.4.2 Método de separación

Este método se basa en el principio de separar las primitivas, en las que son colisionadas por la herramienta virtual, se duplican los nodos significantes para el corte y se procede a la separación apoyándose en estos nodos.

Cumple con las leyes físicas de conservación de la masa y mantiene las propiedades físicas del modelo. Garantiza que no aumente el número de elementos del modelo en el proceso de corte, evitando perjudicar la velocidad de simulación. Dentro de sus desventajas se encuentra que puede ocurrir la degeneración de los elementos cuando el instrumento de corte atraviesa la malla cerca de un nodo.

Nienhuys emplea un modelo linear de elementos finitos y basa su algoritmo de corte en este método [9]. Luego continúa profundizando en el método, pero basándose en un modelo de elementos infinitos y con una malla volumétrica, describe además tres pasos fundamentales en el corte por este método de separación: La selección de la superficie de corte, la ruptura de los nodos y la eliminación de las degeneraciones.

Por su parte Boux de Cason emplea la separación para cortar una malla superficial 2D utilizando el modelo físico masa-resorte, posteriormente aplica este trabajo en un framework de operaciones laparoscópicas.

Otro trabajo importante en 2D, se realiza usando el modelo masa resorte, sobre este método, generalizándolo para largos desplazamientos, requerimiento importante para cortar, creado por Mendoza y Laugier [10], después Nienhuys emplea esta misma idea en 3D, pero sin tener en cuenta los desplazamientos largos.

Mendoza junto a Laugier y B. Cason [28] realizan también un corte en 2D por separación y como resultado lo observamos en una vesícula.

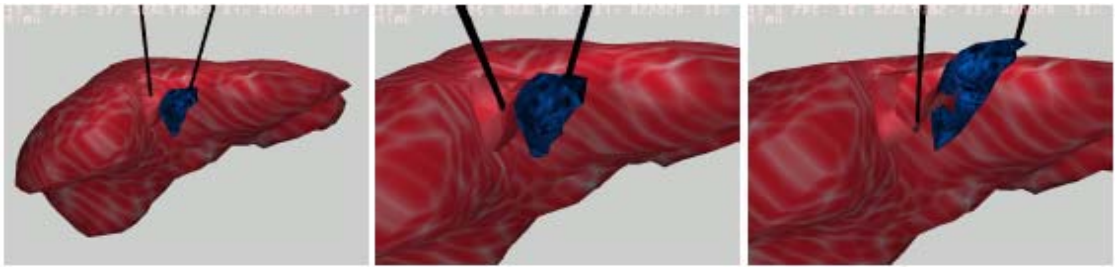


Fig. 10: Cortando el tejido que une la vesícula al hígado.

1.4.3 Método de subdivisión

A finales de los 90, Bielser pone en práctica un nuevo método de corte, el cual basa su principio en la subdivisión de los elementos colisionados por la herramienta virtual de corte, en este caso el bisturí es modelado como instrumento de corte.

Este método cumple con el principio físico de conservación de la masa, no requiere de extensos procesos de refinamiento de la malla, sin embargo, aumenta el número de elementos, lo que provoca lentitud en el ciclo de la simulación y baja calidad en los nuevos elementos. Este algoritmo no es práctico para cortar mallas volumétricas, al menos al realizar una hepatectomía (extirpación de parte o la totalidad del hígado).

Fue introducido por primera vez en el contexto médico cuando Mazura y Seifert [6], plantean el corte de una malla tetraédrica con planos predefinidos. Bielser en su trabajo [29] usa el modelo masa-resorte en una malla tetraédrica para simular el corte y la deformación. Este algoritmo fue refinado más tarde por Mor y T. Kanade [7], abordando temas como la incisión parcial de la malla, emplean un modelo masa-resorte, proponen un corte progresivo y minimizan el número de elementos creados. Posteriormente, D. Bielser, P. Glardon, M. Teschner, M. Gross [8] plantean una máquina de estados, la cual sigue el patrón topológico y controla actualizaciones necesarias de cada tetraedro. Esto da lugar a un algoritmo rápido para la simulación dinámica de la trayectoria volumétrica muy exacta en tiempo real.

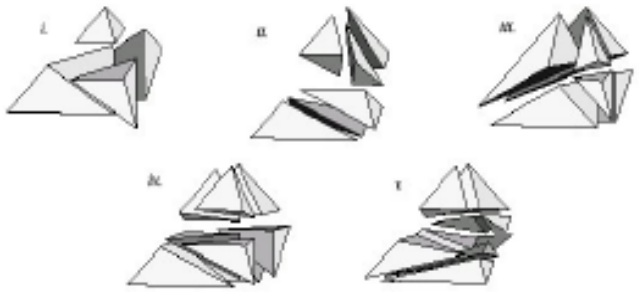


Fig. 11: Proceso de corte mediante el método de subdivisión.

1.4.4 Corte Progresivo en modelo superficial con generación de ranura interna.

En el trabajo realizado por Hui Zhang, Shahram Payandeh y John Dill [30], se propone un modelo masa-resorte, con una base sólida, para simular el funcionamiento de un corte virtual usando un dispositivo de entrada. Se introduce un novedoso y nuevo algoritmo para subdividir la superficie y generar la estructura interna siguiendo el movimiento del dispositivo empleado.
Cuando el instrumento cortante penetra el objeto estamos en presencia de una operación de corte, en el trabajo referenciado se definen dos estados para la misma:

- Estado de contacto: cuando el objeto no ejerce fuerza suficiente para penetrar el objeto. Aquí se deforma el objeto.
- Estado de penetración: cuando es penetrado el objeto, es decir, cuando se logra el corte.

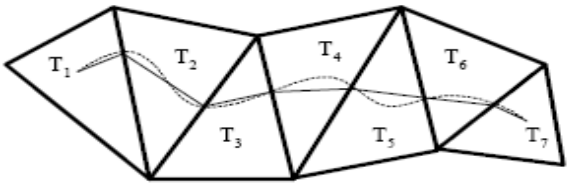


Fig. 12: Ejemplo del corte.

Para generar la ranura, estructura interna de la superficie, se utiliza el principio y el final del corte realizado en el triangulo correspondiente, hallándose así la profundidad.

Se plantean dos tipos de corte progresivo:

- Corte progresivo con subdivisión temporal, donde se genera una ranura interna temporal, en la cual se trata cada triángulo como un estado final del corte, sudvidiéndolo temporalmente según donde pase el instrumento cortante.

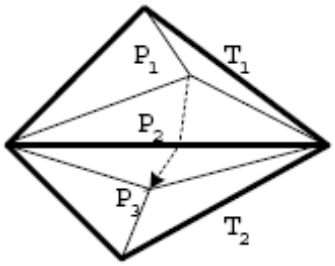


Fig. 13: Corte progresivo con subdivisión temporal.

- La unión de dos cortes, permitiendo realizar un proceso cíclico de corte.

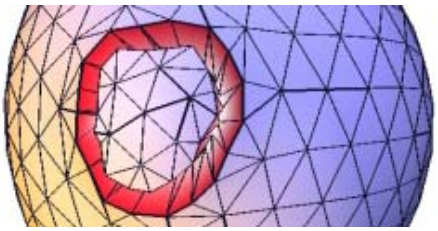


Fig. 14: Proceso cíclico de corte.

El corte progresivo en objetos deformables se ha extendido a malla superficial usando el método de subdivisión, tal que se genera una estructura interna, la cual puede visualizarse. Este es un algoritmo novedoso y eficaz comparado con el modelo volumétrico ampliamente utilizado, según el estudio de Hui Zhang y otros.

1.5 Métodos de deformación

Los modelos deformables basados en física tienen tres décadas en la historia de los gráficos por computadora, en este campo se extienden considerablemente desde el trabajo realizado por Gibson y Mirtich [31] y muchas son las contribuciones alcanzadas, las cuales se observan en diferentes áreas como la animación de ropa, la simulación de fluido estable, etc.

El proceso de corte comienza con la deformación del objeto al ser tocado por la herramienta virtual de corte, luego se modifica la topología de la malla, ya sea separándola o destruyéndola, tendencias en las que se centra este trabajo, sin embargo, para lograr que este proceso de corte sea completo debe acoplarse a un módulo deformable tratado a través de alguno de los métodos de deformación que se describen a continuación, esto podría realizarse en un trabajo futuro.

1.5.1 Sistema masa- resorte

El sistema masa-resorte es un simple modelo físico con una sólida fundamentación matemática. Es computacionalmente ligero y relativamente pequeño [32] y apropiado para aplicaciones interactivas. Se plantea que con la estructura de los sistemas masa–resorte se pueden llevar a cabo largas deformaciones y modificaciones topológicas. Es usado ampliamente para modelar objetos blandos, consiste en la discretización de los objetos como una malla de partículas y muelles, así como en animaciones faciales estáticas y dinámicas [33] [34], también han sido utilizadas para la simulación de ropa, video juegos y películas de animados.

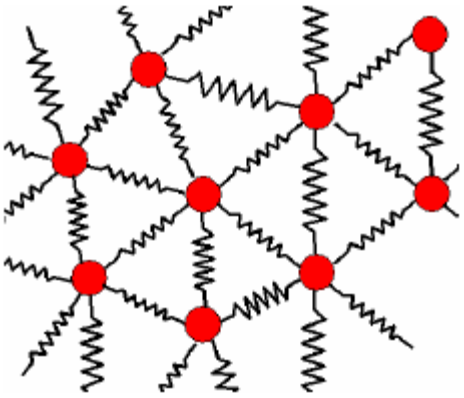


Fig. 15: Sistema Masa- Resorte.

Uno de los primeros trabajos en el uso de este método para soluciones dinámicas fue propuesto por D. Terzopoulos y K. Waters en 1990 ver [34], este tipo de modelación ha sido muy usada hasta hoy debido a que es la forma más intuitiva de imaginar un modelo deformable, es una técnica poco compleja para la implementación y garantiza velocidades superiores a métodos continuos [31]. Sin embargo, no es todo lo exacta que se necesita porque no se soporta en la elasticidad continua, además depende en gran medida de la topología y la resolución de la malla [35].

1.5.2 Método de Elementos Finitos (FEM)

Es uno de los métodos más populares en las ciencias de la computación para resolver ecuaciones diferenciales en rejillas irregulares. El artículo publicado por Turner, Clough, Martin y Topp en 1956 es reconocido como el inicio del actual Método de Elemento Finito [36]. Bro-Nielsen y Cotin trabajan con FEM Linealizado para simulación de cirugía. Ellos lograron un aumento de velocidad simulando solo los nodos visibles de la superficie, similar al método de Elementos de la Frontera (*Boundary Element Method, BEM*). De esta manera se obtienen resultados en tiempo real [37].

Este método transforma la mecánica continua de la deformación en un problema individual que puede ser resuelto usando el análisis numérico. Este método descompone el modelo en pequeños polígonos o poliedros: triángulos en 2D y tetraedros en 3D.

La principal ventaja de FEM comparada con otras aproximaciones, es que puede producir simulaciones más realistas físicamente, debido a que las matrices de masa y rigidez permanecen constantes en el tiempo. Sin embargo, requiere de muchos cálculos, que sólo pueden ser reducidos disminuyendo el número de elementos, lo que atenta contra la exactitud del modelo [38].

1.5.3 Método de Elementos de Frontera

Este método fue propuesto por primera vez para simular objetos deformables por Doug L. James y Dinesh K. Pai en 1999 [39], en el año 2003 se propone otro trabajo agregando posibilidades para el cambio de las condiciones de la frontera, como colisiones órgano – órgano [40].

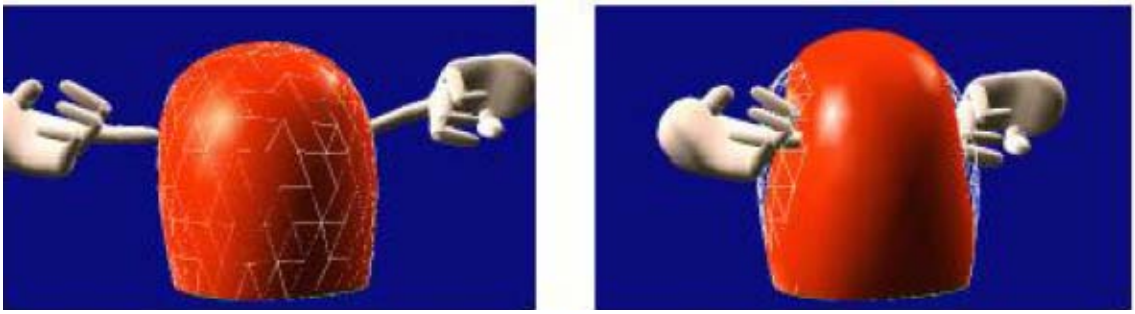


Fig. 16: Ejemplo de la deformación de un objeto usando BEM. 1

El Método de Elementos de Frontera (Boundary Element Method, BEM), es una alternativa interesante al FEM estándar, porque todos los cálculos se realizan en la superficie del cuerpo elástico en lugar de su volumen. BEM garantiza una velocidad sustancial porque el problema tridimensional lo reduce a dos dimensiones. Sin embargo, el método sólo funciona para objetos que en su interior estén compuestos por un material homogéneo. “En los cambios topológicos como corte y fractura son más difíciles de emplear que el método de FEM explícito” [41].

Entre las ventajas que ofrece este método vale resaltar que es más preciso que FEM para calcular fuerzas de contacto y quizás la mejor opción que ofrece BEM es que usa la misma discretización usada para el render, es decir, no se necesita otro enmallado [39].

1.6 Colisiones

La interacción entre los objetos virtuales dentro de un ambiente dinámico es inevitable. A esta interacción, choque o contacto entre dos elementos virtuales se le denomina colisión, la detección de estas colisiones es el primer paso para realizar una interacción realista.

Fases de la Detección de Colisiones

En un ambiente virtual donde están presentes objetos rígidos o deformables la estrategia para detectar colisiones se divide en tres fases.

Fase Amplia: Son seleccionados el par de objetos que probablemente colisionan en esta primera fase de optimización. Los algoritmos empleados en esta fase principalmente están basados en descomposición espacial.

Fase Estrecha: Es otra fase de optimización pero en esta se da una idea preliminar acerca de la región donde los objetos pudieran colisionar, a lo que denominados zona de colisión.

Fase Exacta: En esta se obtiene la obtiene la entidad exacta de colisión. El algoritmo de esta fase chequea la colisión entre dos polígonos. En esta fase el algoritmo de colisión usado determina si al menos dos primitivas se interceptan. Fundamentalmente son considerados estas primitivas: esfera, caja, triángulo.

Consideraciones del capítulo

En este capítulo se abordaron temas fundamentales a tener en cuenta para simular el proceso de corte para simuladores quirúrgicos. Al finalizar se cuenta con una reseña del estado del arte y se plantean las tendencias actuales para obtener un algoritmo de corte, donde se perfilan las ventajas y desventajas que acarrean cada uno de estos, según distintas fuentes bibliográficas.

Con el estudio del marco teórico conceptual realizado se considera que existe la base suficiente para proponer una solución en el siguiente capítulo.

Capítulo 2: Soluciones Técnicas

Introducción

En este capítulo se describen las técnicas que conforman la solución que se propone en el presente trabajo para resolver el problema de obtener mayor realismo en el proceso de corte en un simulador. Se explica además como se maneja la detección de colisiones y el seguimiento de estas durante el proceso de corte. Además se describen las herramientas, la metodología y el lenguaje a emplear para el desarrollo y la implementación de las técnicas de corte seleccionadas.

2.1 Procedimientos de corte.

En el anterior capítulo se describen las tendencias fundamentales del corte en el mundo, el método más empleado, el de subdivisión plantea que donde se intercepten el objeto y la herramienta virtual de corte, se subdivida el modelo creando a su vez nuevos elementos, lo cual es una desventaja significativa pues provoca lentitud en el ciclo de la simulación y baja calidad en los nuevos elementos, mientras que el primer método descrito, el más antiguo, el destructivo se basa en eliminar el elemento interceptado, por consiguiente, disminuyen los elementos y a su vez el tiempo de simulación y el segundo método analizado, el de separación, en las primitivas que son colisionadas por la herramienta virtual se duplican los vértices significantes para el corte y se procede a separarlos apoyándose en estos nodos, de esta forma se garantiza que no aumente el número de elementos y no se perjudica la velocidad de la simulación.

Finalmente se desarrollarán estas dos técnicas de corte, el método destructivo y el método de separación, dando seguimiento a una de las recomendaciones del trabajo de diploma realizado anteriormente por Lester O. Rodríguez y Leonardo R. Fernández acerca de este tema [42].

Se propone la solución a la técnica destructiva basada en lo planteado por Cotin, Delingette y Ayache [4] y a la técnica de separación por el trabajo realizado por Niehyus y Van der Stappen [9], pero en este trabajo estas técnicas se desarrollarán sobre mallas triangulares.

2.1.1 Proceso general de corte a través del método destructivo.

El procedimiento general para cortar, mediante el método eliminar elementos en una malla triangular es el siguiente: Primero se verifica con que triángulo de la malla colisiona el objeto virtual de corte. Después ese triángulo interceptado es eliminado, dando lugar al corte a través del método propuesto, por último se actualizan las estructuras de datos empleadas.

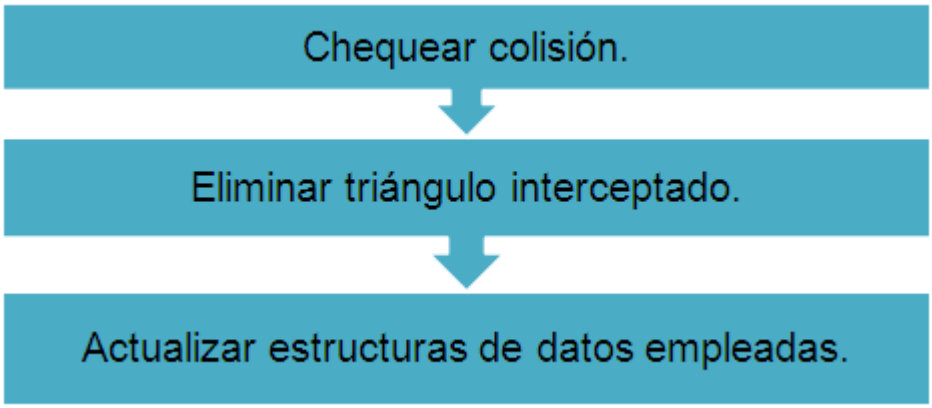


Fig. 17: Proceso general de corte a través del método destructivo.

2.1.2 Proceso general de corte a través del método de separación.

El procedimiento para cortar a través del método de separación es un poco más complicado y extenso, los pasos son los siguientes:

- Primero se verifica con que triángulo de la malla colisiona la herramienta virtual de corte y el punto donde se interceptan.
- Se busca el vértice más cercano al punto de intersección, en el triángulo colisionado.

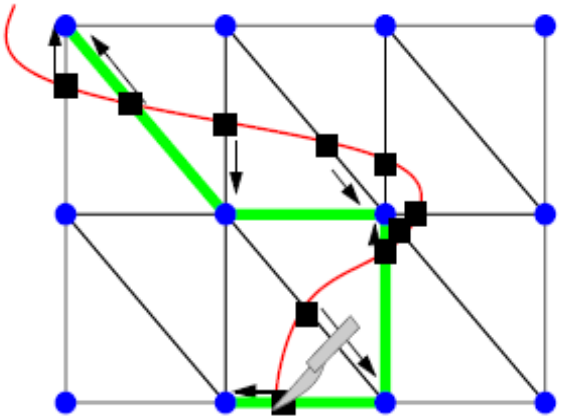


Fig. 18: Trayectoria del instrumento de corte asociándose al vértice más cercano.

- Buscar los triángulos asociados al vértice más cercano, el cual será el que se va a separar.
- Se busca entonces el vértice que está en dirección de la herramienta de corte.
- Se verifica si la arista que forman este vértice y el vértice que se obtuvo como más cercano al punto de intersección pertenece a dos triángulos.
- Luego se pasa a separar el vértice, para ello se:
    - Crean dos nuevos vértices.
    - Se le asigna a cada triángulo afectado por el corte el vértice que le corresponda.
- Por último se actualizan todas las estructuras de datos empleadas.

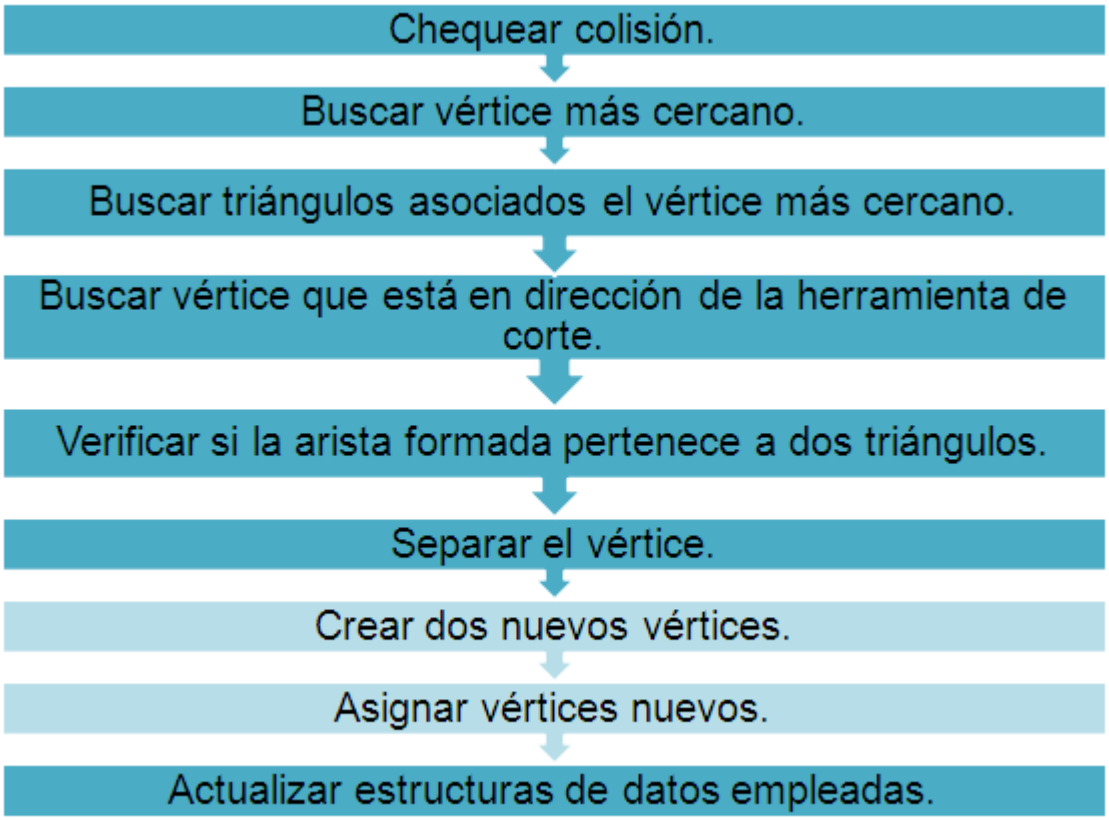


Fig. 19: Proceso general de corte a través del método de separación.

2.1.2.1 Separación y generación de vértices

La fase de separar el vértice que corresponde a más de un triángulo consiste en generar dos nuevos vértices, es decir, duplicar el vértice seleccionado como el más cercano al punto de intersección, realizarlo de acuerdo a la normal de la herramienta de corte, es decir, crear un vértice a favor de la normal y otro en contra, por último asociar a cada triángulo afectado por la trayectoria de corte uno de estos nuevos vértices, tomando el criterio anterior respecto a la normal. Luego se actualizan las estructuras empleadas.

2.2 Detección de colisiones

En el presente trabajo la detección de colisiones se realizará de forma sencilla, solo para dar soporte a una interfaz visual que permita observar el proceso de corte por cualquiera de los métodos desarrollados, pues es este el objetivo del presente trabajo.

Cuando se detecta una colisión entre un triángulo y la herramienta de corte se obtiene el punto donde interactúa la malla con el instrumento de corte. El corte se observa como el segmento cortante del instrumento sobre algún triángulo de la malla, es decir, se ignora todo lo demás y solo se toma en cuenta este segmento y su incisión sobre la malla.

La técnica empleada se basará en el tratamiento de intersecciones segmento-plano, en este caso el segmento será acotado al segmento cortante de la herramienta y el plano a un triángulo de la malla.

Para dar solución se presentan las ecuaciones (1) para el segmento y (2) para el plano.

Sean L1 un segmento y π1 un plano en el espacio:

(1) $L1:(x,y,z)=P+t\bar{v}$

(2) Ecuación cartesiana del plano: $\Pi\,1:\mathrm{a1x}+\mathrm{b1y}+\mathrm{c1z}=\mathrm{d1}$

Para obtener una intersección entre el segmento y el plano se despeja x y z de de la ecuación del segmento y sustituyendo x y z en la ecuación del plano. Luego se resuelve para t, si la solución es única, con este valor de t se obtiene el punto de intersección sustituyendo en la ecuación del segmento.

2.3. Metodologías, Herramientas y Lenguaje Utilizado

2.3.1 Metodología de Ingeniería de Software

El proceso de desarrollo estará guiado por el Proceso Unificado de Desarrollo de Software *(Rational Unified Process, RUP*), el cual constituye una metodología que acumula años de experiencia, además que se ha probado mundialmente que es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software con diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles, diferentes tamaños de proyecto. Esta robusta y poderosa metodología tiene su basamento en tres características fundamentales que la distinguen:

Guiado por casos de uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo del sistema. Conforman el hilo conductor, por el cual se avanza hacia los diferentes flujos de trabajo.

Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo. Incluye los aspectos estáticos y dinámicos más significativos.

Iterativo e Incremental: RUP divide el proyecto en fases de desarrollo, propone además que cada una de ellas se desarrolle en iteraciones, las cuales aportan un incremento en el proceso de desarrollo y terminan con el cumplimiento del punto de control trazado en la fase.

2.3.2 Herramientas de desarrollo

Microsoft Visual Studio.NET 2003

Visual Studio 2003 como entorno integrado de desarrollo (**IDE**), ya que el mismo cuenta con pequeñas herramientas auxiliares de completamiento de códigos como el Visual Assist lo que ayuda mucho a la codificación de la aplicación, y otras como el VTune que le permite al desarrollador ir comprobando el rendimiento en cuanto a memoria y uso del CPU de la aplicación que se está codificando. Posee gran integración de varios lenguajes entre ellos el C++, C#, y Asp.Net. Otra razón por la que se emplea es que es la plataforma de desarrolla de la STK, el motor gráfico que se empleará para visualizar la solución propuesta en este trabajo.

Rational Rose Enterprise Edition 2003

Se emplea para modelar la solución propuesta. Ofrece un total y sencillo soporte al proceso de desarrollo, brinda una interfaz cómoda, amigable para trabajar.

2.3.3 Lenguajes

Lenguaje de programación

Se escogió el lenguaje de programación C ++ por ser un lenguaje estandarizado por la norma ISO/IEC 14882:1998, por ser multiplataforma y además constituir el lenguaje por excelencia para las aplicaciones de Realidad Virtual. Está considerado un lenguaje potente al poder trabajar tanto en alto como en bajo nivel, aunque C++ es de alto nivel. Entre sus principales características se encuentra el soporte para la programación orientada a objetos, la posibilidad de redefinir operadores, conocida como sobrecarga de operadores, la identificación de tipos en tiempo de ejecución y que además presenta una biblioteca estándar.

Lenguaje de modelado

Para la realización del análisis y diseño se utilizará el UML (*Unified Modeling Language*) por constituir un estándar mundial y ser independiente de la plataforma en que se trabaje.
Este lenguaje de modelo garantiza una comunicación de alto nivel para todo el que interactué con el modelo, además presenta características tales como:

- Permite modelar sistemas utilizando técnicas orientadas a objetos (OO).
- Permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, sin ambigüedades y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).

Consideraciones del capítulo

En este capítulo se expone la solución a desarrollar, seleccionadas entre varios algoritmos para dar cumplimiento al objetivo y lograr mayor realismo al interactuar con un simulador. Se describe como se detectarán las colisiones, lo cual es el primer proceso que se necesita desarrollar para realizar un corte. Se especificó las herramientas de desarrollo a emplear, los lenguajes, tanto de modelado como para programar y la metodología de ingeniería de software, la cual es fundamental para desarrollar un proceso.

Capítulo 3: Descripción de la solución propuesta

Introducción

En el presente capítulo se describe la solución propuesta en el capítulo anterior desde el punto de vista del cliente, es decir, desde sus necesidades y perspectivas, a través de la captura de los requisitos, ya sean funcionales o no funcionales. Se presentan las condiciones que debe cumplir el sistema para un correcto funcionamiento conocidas como reglas del negocio. Se define mediante el modelo de dominio los conceptos fundamentales del área de interés. Se presenta además un modelo de casos de uso para resumir el proceso de levantamiento de requisitos, tal como lo define el Proceso Unificado de Software.

3.1 Reglas del negocio

El sistema está basado en mallas triangulares.

El desplazamiento del segmento cortante debe ser menor que el área del triángulo.

3.2 Modelo de dominio

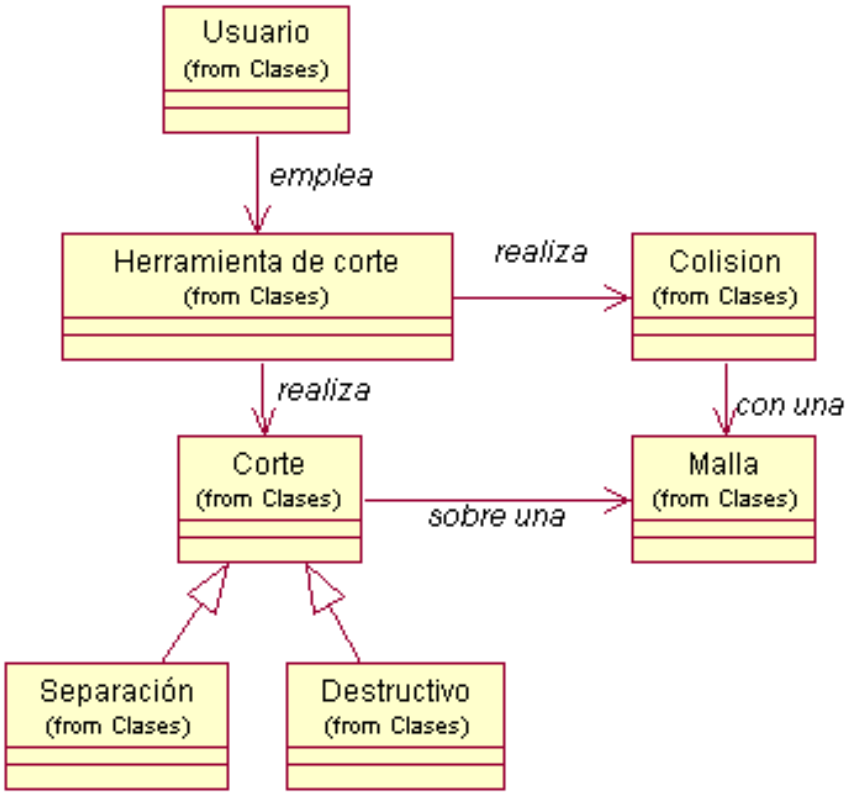


Fig. 20: Modelo de dominio.

3.2.1 Glosario de términos del modelo de dominio.

Usuario: Persona que interactúa con el sistema, con el fin de realizar un corte.

Herramienta de corte: Es una herramienta especial, que permite manipular a un médico desde fuera por su cuerpo auxiliar alargado.

Corte: Proceso que realiza un usuario sobre una malla, el cual provoca una incisión sobre la misma.

Destructivo: Tipo de corte a seleccionar.

Separación: Tipo de corte a seleccionar.

Colisión: Choque, intersección entre el modelo a cortar y la herramienta de corte.

Malla: Representa la malla triangular, modelo que será sometido al corte.

3.3 Captura de requisitos

El flujo de trabajo levantamiento de requisitos define las condiciones o capacidades que el sistema debe cumplir. Estas se dividen en dos clasificaciones, los requisitos funcionales, los cuales constituyen lo que el sistema debe hacer y cómo debe hacerlo y los requisitos no funcionales, estos por su parte conforman las cualidades o propiedades del sistema, las cuales establecen aspectos que regulan el comportamiento del mismo.

3.3.1 Requisitos funcionales

R1. Separar malla

R1.1Duplicar vértices

R2. Destruir triángulo de la malla

R2.1 Asociar vértices

R.3 Gestionar triángulos de la malla

R3.1 Buscar triángulo

R3.2 Eliminar triángulo

R3.3 Almacenar lista de triángulos vecinos

R3.4 Actualizar lista de triángulos vecinos

R3.5 Buscar vértice

R3.6 Separar vértice

R3.7 Almacenar lista de vértices

R3.8 Actualizar lista de vértices

R3.9 Crear nuevos vértices.

R4 Detectar intersecciones

R4.1 Detectar intersección entre la herramienta de corte y la malla.

R4.2 Detectar punto de intersección entre el segmento cortante y el triángulo.

3.3.2 Requisitos no funcionales

Usabilidad: Los futuros usuarios deberán tener conocimientos básicos de programación gráfica y la tecnología a fin.

Rendimiento: Como aplicación en tiempo real debe tener alta velocidad de procesamiento o cálculo, tiempo de respuesta y de recuperación, y disponibilidad.

Soporte: Compatible con la plataforma Windows y Linux.

Hardware: Compatibilidad con tarjetas gráficas de la familia NVIDIA.

Diseño e implementación: Emplear la biblioteca de clases STK la cual tiene definidas las funcionalidades básicas para trabajar con gráficos por ordenador, soporta OpenGL y DirectX. Se harán llamadas a dicha biblioteca desde el lenguaje C/C++. Se regirá por la filosofía de Programación Orientada a Objetos.

3.4 Modelo de Casos de Uso del Sistema

Este modelo describe los procesos de un sistema como casos de uso y su interacción con elementos externos nombrados actores, tales como socios y clientes, es decir, describe las funciones que el negocio pretende realizar y su objetivo básico es describir cómo el sistema es utilizado por sus clientes y socios.

3.4.1 Actor del Sistema

Actores	Justificación
Sistema externo	Es quien interactúa con las funcionalidades del sistema.

Tabla 1: Actor del Sistema.

3.4.2 Casos de Uso del Sistema

CU1	Separar malla
Actor	Sistema externo
Descripción	Modifica la estructura de la malla generando una abertura.
Referencia	R1.1, CU3 (include), CU4 (include).

Tabla 2: CU1 Separar malla.

CU2	Destruir triángulo de la malla
Actor	Sistema externo
Descripción	Modifica la estructura de la malla generando una abertura.
Referencia	R2.1, CU3 (include), CU4 (include).

Tabla 3: CU2 Destruir triángulo de la malla.

CU3	Gestionar triángulos de la malla
Actor	CU1 o CU2.
Descripción	Maneja todo el trabajo con la malla, es decir, buscar, eliminar, almacenar y actualizar triángulos de la malla y vértices de estos.

Referencia	R3.1, R3.2, R3.3, R3.4, R3.5, R3.6, R3.7, R3.8, R3.9.

Tabla 4: CU3 Gestionar triángulos de la malla.

CU4	Detectar intersecciones
Actor	CU1 o CU2
Descripción	Maneja las interacciones entre la herramienta de corte y la malla a cortar.
Referencia	R4.1, R4.2.

Tabla 5: CU4 Detectar intersecciones.

3.4.3 Diagrama de Casos de Uso del Sistema

El siguiente diagrama muestra la interacción entre el actor y los casos de uso del sistema.

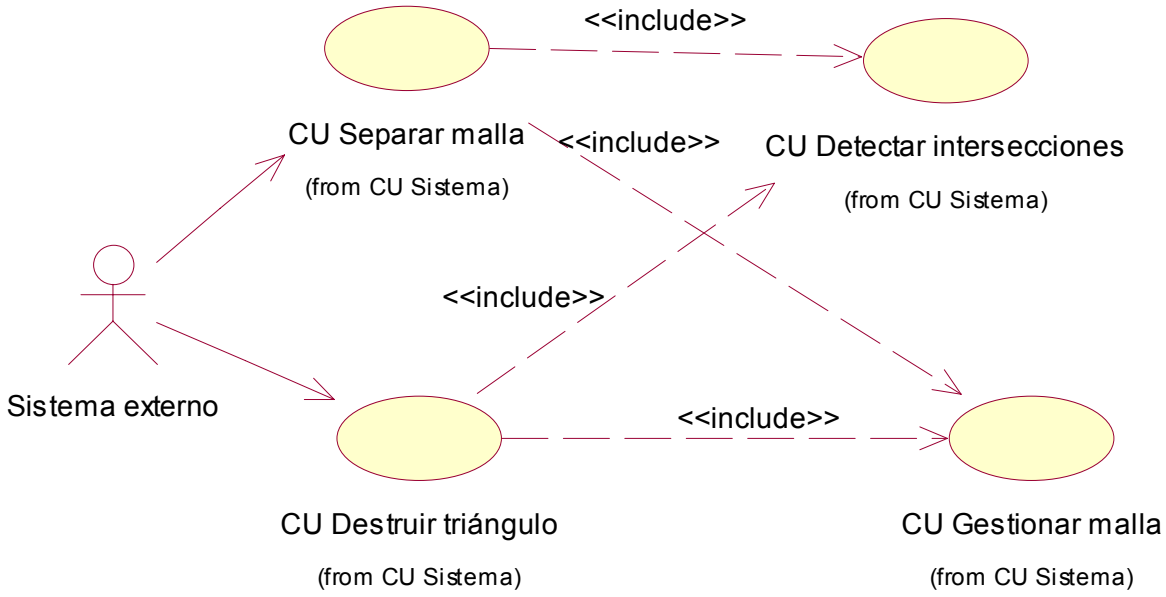


Fig. 21: Diagrama de Casos de Uso del Sistema.

3.4.4 Expansión de Casos de Uso

Caso de uso	
CU1	Separar malla
Propósito	Modifica la estructura de la malla generando una abertura
Actor	Sistema externo
Resumen: El caso de uso se inicia cuando el actor demanda realizar un corte a la malla por el método de separación.	
Referencias	R1.1, CU3 (include), CU4 (include)
Precondiciones	Detectar contacto entre el segmento cortante y un triángulo de la malla
Poscondiciones	La malla debe quedar separada, realizando una apertura.
Curso Normal de los Eventos	
Acción del actor	Respuesta del sistema
1-El simulador desea separar una malla, se realiza movimiento de la herramienta de corte por la malla a separar.	2. Al detectar contacto con un triángulo **ver descripción CU4** se obtiene el triangulo donde hubo el primer contacto y el punto de intersección.
	3. Se busca el vértice más cercano al punto de intersección.
	4. Chequear que la arista que conforman estos dos vértices pertenezca a más de un triángulo.
	5. Duplicar vértice, es decir, crear nuevos vértices, **ver CU3, sección "Separar vértices".**
	6. Asociar nuevos vértices a cada triángulo afectado, ver CU3, sección **"Actualizar lista de vértices".**
Curso alternativo	

	4. Almacenar vértice.
Prioridad	Crítico

Tabla 6: Expansión CU1 Separar malla.

Caso de uso	
CU2	Destruir triángulo de la malla
Propósito	Modifica la estructura de la malla generando una abertura.
Actor	Sistema externo
Resumen: El caso de uso se inicia cuando el actor demanda realizar un corte a la malla por el método destructivo.	
Referencias	R2.1, CU3 (include), CU4 (include)
Precondiciones	Detectar contacto entre el segmento cortante y un triángulo de la malla
Poscondiciones	El triángulo debe ser eliminado y debe quedar una apertura en la malla.
Curso Normal de los Eventos	
Acción del actor	Respuesta del sistema
1-El simulador desea destruir un triángulo, se realiza movimiento de la herramienta de corte por el triángulo a eliminar.	2. Al detectar contacto con un triángulo **ver descripción CU4** se obtiene el punto donde hubo el primer contacto.
	3. Se busca el triángulo correspondiente.
	4. Se elimina el triángulo, ver CU3, sección “Eliminar triángulo”.
	5. Asocian vértices a los triángulos restantes.
	6. Actualiza la lista de triángulos.
Prioridad	Crítico

Tabla 7: Expansión CU2 Destruir triángulo de la malla.

Caso de uso		
CU3	Gestionar triángulos de la malla	
Propósito	Maneja todo el trabajo con la malla, es decir, buscar, eliminar, almacenar y actualizar triángulos de la malla y vértices de estos.	
Actor	CU1 o CU2.	
Resumen**:** El caso de uso se inicia cuando el CU1 o el CU2 demandan alguna acción sobre la malla.		
Referencias	R3.1, R3.2, R3.3, R3.4, R3.5, R3.6, R3.7, R3.8, R3.9.	
Precondiciones	Tener una estructura de datos para gestionar los triángulos	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema
1. CU1 o CU2 seleccionan realizar alguna acción sobre la malla, triángulos de esta o sus vértices.		2. Permite realizar las siguientes acciones. a) Si decide eliminar triángulos ver sección "Eliminar triángulo". b) Si decide separar vértice ver sección "Separar vértices".
Sección "Eliminar triángulo"		
Acción del actor		Respuesta del sistema
3. Selecciona eliminar triángulo.		4. Busca el triángulo, lo encuentra y lo elimina.
		5. Actualizar lista de triángulos.
Sección "Separar vértices"		
6. Selecciona separa vértices.		7. Crear nuevos vértices.
		8. Asociar los nuevos vértices a sus triángulos

	correspondientes.
	9. Actualizar lista de vértices.
	10. Actualizar lista de triángulos.
Prioridad:	Crítico.

Tabla 8: Expansión CU3 Gestionar triángulo.

Caso de uso		
CU4	Detectar interacciones	
Propósito	Maneja las interacciones entre el segmento cortante y el triángulo a cortar.	
Actor		
Resumen: El caso de uso se inicia cuando la herramienta de corte colisiona con la malla.		
Referencias	R4.1, R4.2.	
Precondiciones	Tener la herramienta de corte y la malla a cortar.	
Poscondiciones	Debe obtenerse el triángulo con el que colisiona la herramienta y el punto por donde es interceptado ese triángulo por el segmento cortante.	
Curso Normal de los Eventos		
Acción del actor		Respuesta del sistema
		1. Detectar triángulo de intersección entre la herramienta de corte y la malla.
		2. Detectar punto de intersección entre el segmento cortante y el triángulo de la malla.
		3. El CU termina cuando no exista ninguna interacción.
Prioridad:	Crítico.	

Tabla 9: Expansión CU4 Detectar interacciones.

Consideraciones del capítulo

Al concluir este capítulo queda expuesto un modelo del dominio, con el objetivo de exponer mediante conceptos el contexto del sistema; además se realiza la captura de requisitos tanto funcionales como no funcionales, con el fin de plasmar lo que debe hacer el sistema y lo que necesita, desde el punto de vista del cliente; se describen los casos de uso, a partir del levantamiento de requisitos realizado anteriormente y se muestra el diagrama de estos casos de uso del sistema para lograr una mejor comprensión del mismo.

Estos resultados son el punto de partida necesario para el diseño e implementación del sistema, lo cual se desarrollará a continuación.

Capítulo 4: Diseño e Implementación del Sistema

Introducción

Después de realizado el levantamiento de requisitos, rigiéndose por la filosofía de RUP, es necesario definir la estructura del sistema, mediante el desarrollo de los flujos de trabajo diseño e implementación, estos se llevará a cabo en el presente capítulo a través de un diagrama de clases general, mediante los diagramas de secuencia de cada caso de uso, descritos el capítulo anterior y basados en los requisitos del cliente y por último se expondrán los diagramas de componente y la relaciones entre los paquetes que conforman el sistema. Además se describen en tablas las clases del diseño que serán implementadas.

4.1 Scene Toolkit (STK)

Para una mejor comprensión del sistema será necesario citar algunas clases de la STK Este módulo no está acoplado a esta biblioteca de clases, solo utiliza la malla triangular que la STK genera, desde su capa Engine, en el paquete Geometry y mediante la clase CTriMesh, con el fin de visualizar las técnicas de corte implementadas. La STK por su parte también puede emplear este módulo de corte desde su capa de aplicación.

4.2 Diseño del sistema

El sistema que se desarrolla está dividido en tres paquetes: el paquete del manejo y trabajo con la malla, así como con el corte nombrado Cut, el paquete del trabajo con colisiones y un paquete para el trabajo matemático requerido. A continuación se presenta el modelo lógico del sistema y los diagramas de clases de cada paquete.

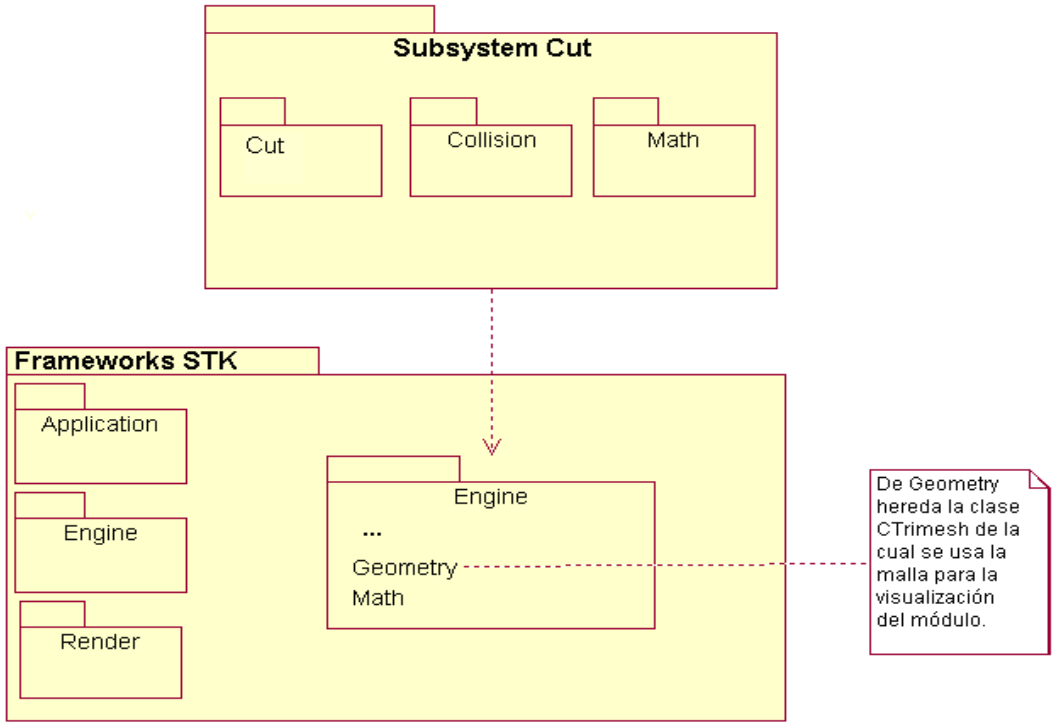


Fig. 22: Modelo lógico del sistema.

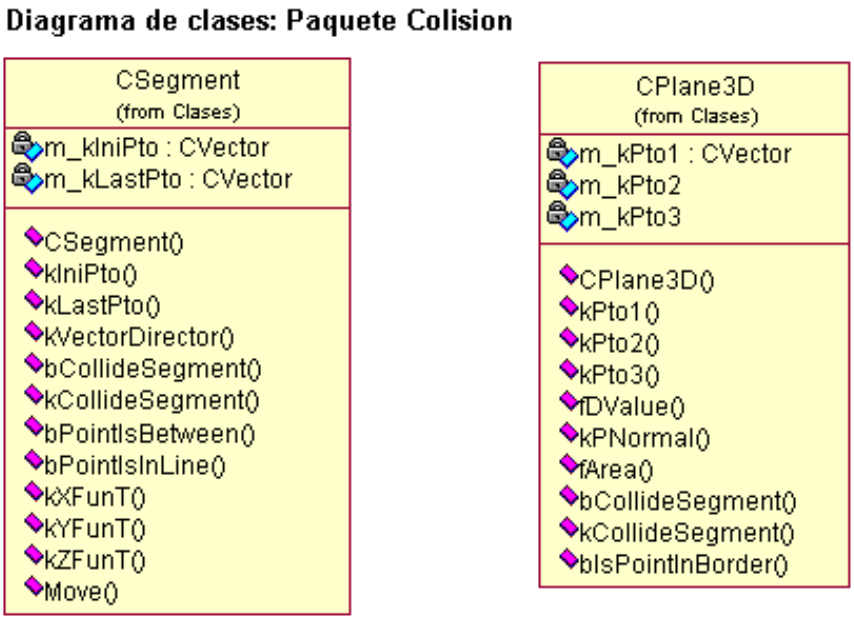


Fig. 23: Diagrama de clases: Paquete Colision.

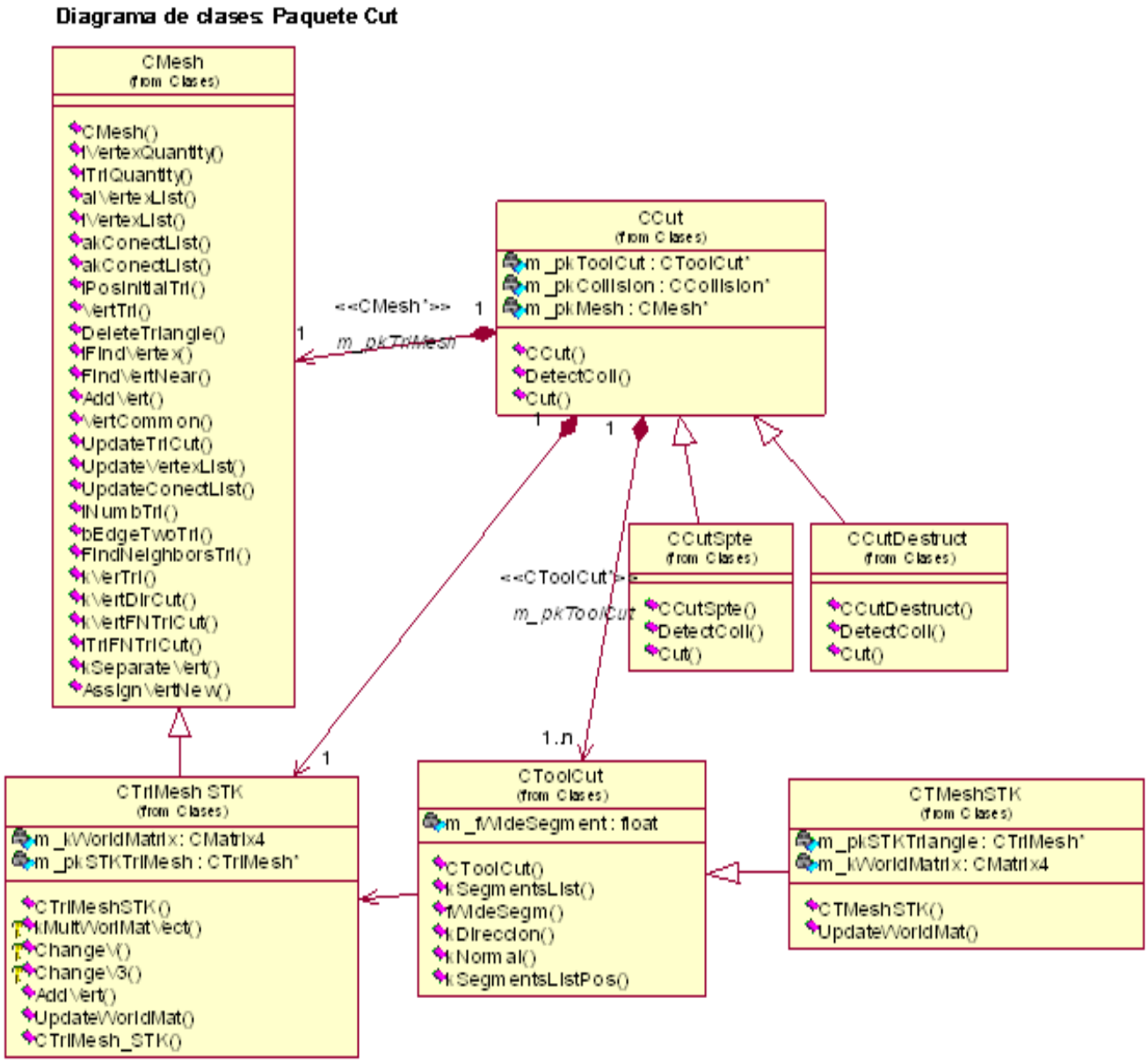


Fig. 24: Diagrama de clases: Paquete Cut.

Diagrama de clases: Paquete Math.

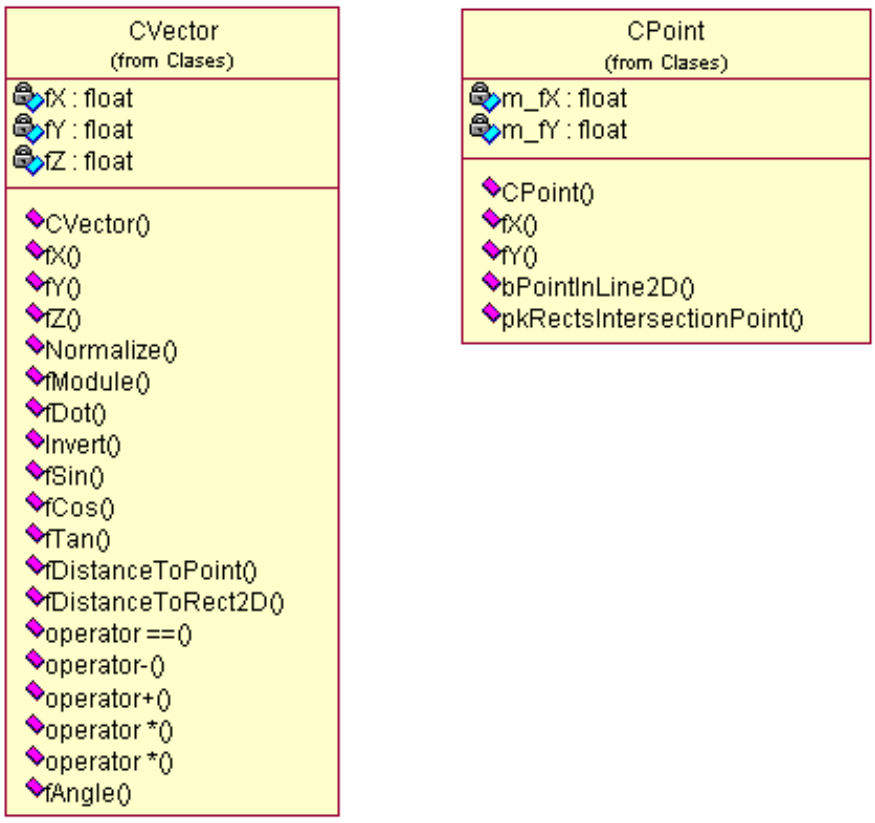


Fig. 25: Diagrama de clases: Paquete Math.

4.2.1 Diagrama de clases del diseño

En el diagrama que a continuación se presenta se muestran las diferentes clases del diseño y sus relaciones entre ellas, es necesario incluir las clases CTriMesh implementada ya en la STK para lograr un mayor entendimiento de la arquitectura del sistema.

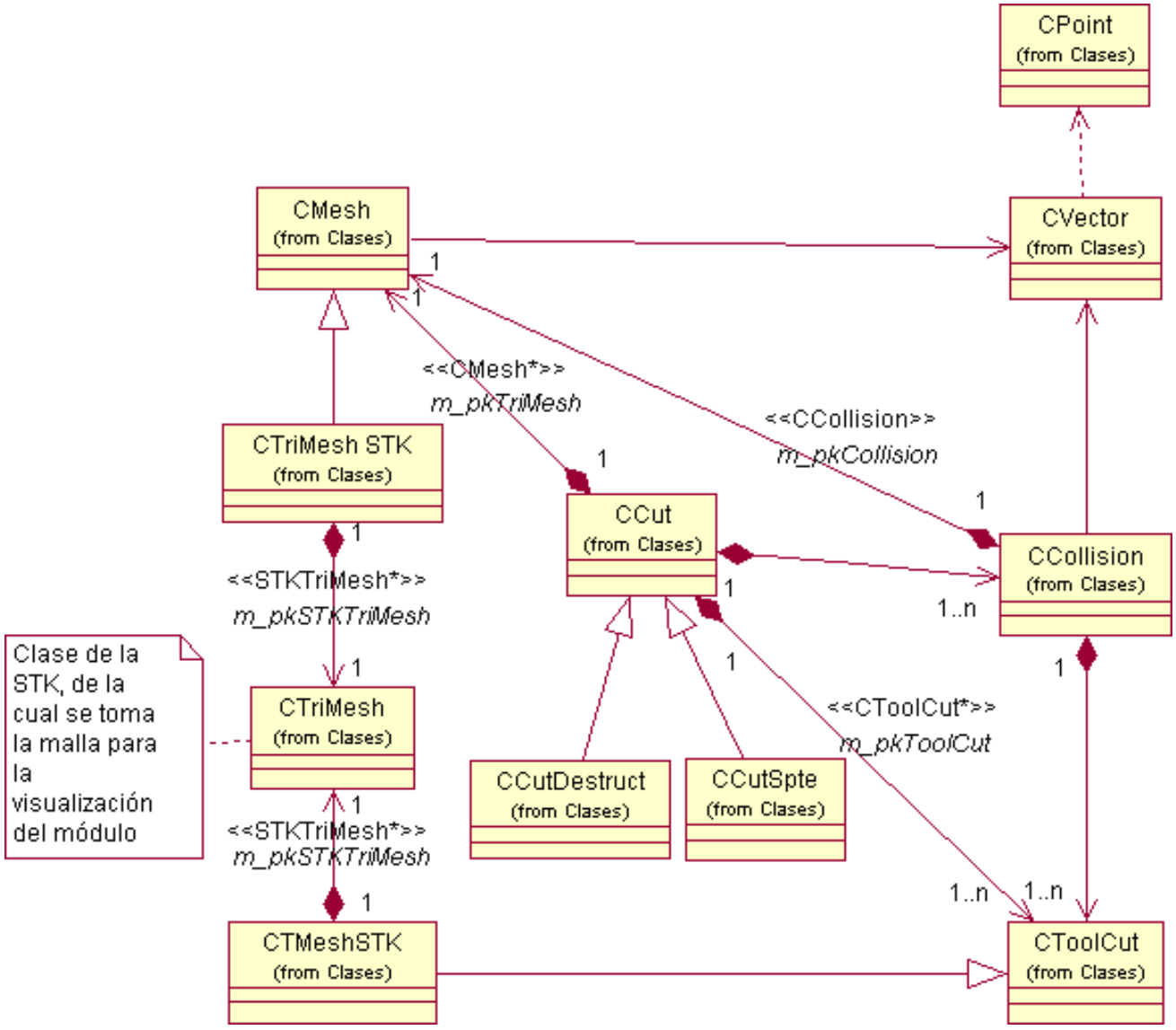


Fig. 26: Diagrama de clases del diseño.

4.3 Descripción de las clases del diseño

Nombre: CMesh	
Tipo de clase: Entidad	
Para cada responsabilidad	
Nombre:	CMesh ()
Descripción:	Constructor de la clase.
Nombre:	unsigned int uiVertexQuantity()
Descripción:	Devuelve la cantidad de vértices de la malla.
Nombre:	unsigned int uiTriQuantity()
Descripción:	Devuelve la cantidad de triángulos de la malla.
Nombre:	int* aiConectList ()
Descripción:	Retorna la lista de conectividad de vértices.
Nombre:	CVector* aiVertexList()
Descripción:	Retorna la lista de vértices de la malla.
Nombre:	CVector iVertexList (int arg_iPos)
Descripción:	Devuelve el valor de la lista de vértices dado una posición.
Nombre:	int iConectList (int arg_iPos)
Descripción:	Devuelve el valor de la lista de conectividad dado una posición.
Nombre:	int iPosInitialTri (int arg_iTriNum)
Descripción:	Devuelve la posición inicial del triángulo en la lista de conectividad, dado el número del triángulo, sería (iTriNum-1) * 3.Se usa para el corte.
Nombre:	bool DeleteTriangle (int arg_iTri)
Descripción:	Elimina un triángulo dado en la lista de conectividad.
Nombre:	CVector* kVertTriang(int arg_iTri)
Descripción:	Devuelve los vértices de un triángulo dado.
Nombre:	int iFindVertex(CVector arg_kVertex)
Descripción:	Busca un vértice determinado en la lista de vértices, devuelve su posición.

Nombre:	CVector FindVertNear(CVector arg_iPto, int arg_iTri)
Descripción:	Busca vértice más cercano al punto de intersección dado por parámetro.
Nombre:	int iNumbTri(int arg_iPos)
Descripción:	Devuelve el número del triángulo, según su posición.
Nombre:	int* aiVertCommon(CVector arg_kVertCom, int* CantTri)
Descripción:	Devuelve los triángulos que tienen en común el vértice dado por parámetro.
Nombre:	bool bEdgeTwoTri(CVector arg_kVert1, CVector arg_kVert2)
Descripción:	Devuelve verdadero si la arista compuesta por los dos vértices dados pertenece a dos triángulos.
Nombre:	CVector* kVerTri(int* arg_aiTriList, int arg_iCanTri, int* CantVert)
Descripción:	Devuelve los vértices de los triángulos que tiene un vértice común.
Nombre:	CVector* kVertDirCut(CVector* arg_akListVert, int* arg_piCantVert, int* arg_piCantVertDir, CVector arg_kDirec, CVector arg_kCommon)
Descripción:	Devuelve el vértice que está en dirección al corte y el que está en contra.
Nombre:	bool bVertFDir(CVector arg_kDir, CVector arg_kVert, CVector arg_kVertCom)
Descripción:	Devuelve verdadero si el vértice está a favor de la dirección del corte.
Nombre:	CVector* kVertFDir(CVector* arg_akListVert, int* arg_piCantVert, int* arg_piCantVertFN, CVector arg_kDir, CVector arg_kVertCom)
Descripción:	Devuelve los vértices que están a favor de la normal de la herramienta de corte.
Nombre:	int* aiTriInDir(CVector* arg_akListVert, int* arg_piCantVert, int* arg_akListTri, int* arg_piCanTri, int* arg_piCanTriInDir, CVector arg_kDir, CVector arg_kVertCom)
Descripción:	Devuelve los triángulos que están a favor de la normal de la herramienta de corte.
Nombre:	CVector* kSeparateVert(CVector arg_kVertCom, CVector arg_kDir, float arg_fDim)
Descripción:	Separar el vértice a cortar, crea dos nuevos vértices uno a favor de la normal, otro en contra.
Nombre:	void AddVert(CVector arg_kVert)
Descripción:	Adiciona un vértice a la lista de vértices.

Nombre:	void AssignVertNew(CVector arg_kVertCom, CVector arg_kDir, float arg_fDim, int* arg_aiLisTri, int* arg_piCanTri)
Descripción:	Asignar los vértices nuevos a cada triángulo afectado por el corte, según si está en contra o a favor de la normal.
Nombre:	int* aiTriCN(int* arg_aiTriList, int* arg_aiTriListFN)
Descripción:	Devuelve los triángulos que están en contra de la normal.
Nombre:	CVector* kVertCN(CVector* arg_akVertList, CVector* arg_akVertListFN)
Descripción:	Devuelve los vértices que están en contra de la normal.
Nombre:	CVector* akDteVertInList(CVector* arg_akVerListIni, int* arg_piCantVertIni, CVector arg_kVert)
Descripción:	Elimina un vértice de la lista de vértices.
Nombre:	CVector* akDteListVertInListVert(CVector* arg_akVerListIni, int* arg_piCantVertIni, CVector* arg_akVerListToDte, int* arg_piCantVertToDte)
Descripción:	Elimina de la lista de vértices una sublista pasada por parámetro.

Tabla 10: Descripción de la clase CMesh.

Nombre: CTriMesh_STK	
Tipo de clase: Interfaz	
Atributo	Tipo
m_pkSTKTriMesh	CTriMesh*
m_kWorldMatrix	CMatrix4
Para cada responsabilidad	
Nombre:	CTriMesh_STK(CTriMesh* arg_pkSTKTriMesh)
Descripción:	Constructor de la clase.
Nombre:	unsigned int uiVertexQuantity()
Descripción:	Devuelve la cantidad de vértices de la malla.
Nombre:	unsigned int uiTriQuantity()

Descripción:	Devuelve la cantidad de triángulos de la malla.
Nombre:	int* aiConectList ()
Descripción:	Retorna la lista de conectividad de vértices.
Nombre:	CVector* aiVertexList()
Descripción:	Retorna la lista de vértices de la malla.
Nombre:	CVector iVertexList (int arg_iPos)
Descripción:	Devuelve el valor de la lista de vértices dado una posición.
Nombre:	int iConectList (int arg_iPos)
Descripción:	Devuelve el valor de la lista de conectividad dado una posición.
Nombre:	int iPosInitialTri (int arg_iTriNum)
Descripción:	Devuelve la posición inicial del triángulo en la lista de conectividad, dado el número del triángulo, sería (iTriNum-1) * 3.Se usa para el corte.
Nombre:	bool DeleteTriangle (int arg_iTri)
Descripción:	Elimina un triángulo dado en la lista de conectividad.
Nombre:	CVector* kVertTriang(int arg_iTri)
Descripción:	Devuelve los vértices de un triángulo dado.
Nombre:	int iFindVertex(CVector arg_kVertex)
Descripción:	Busca un vértice determinado en la lista de vértices, devuelve su posición.
Nombre:	CVector FindVertNear(CVector arg_iPto, int arg_iTri)
Descripción:	Busca vértice más cercano al punto de intersección dado por parámetro.
Nombre:	int iNumbTri(int arg_iPos)
Descripción:	Devuelve el número del triángulo, según su posición.
Nombre:	int* aiVertCommon(CVector arg_kVertCom, int* CantTri)
Descripción:	Devuelve los triángulos que tienen en común el vértice dado por parámetro.
Nombre:	bool bEdgeTwoTri(CVector arg_kVert1, CVector arg_kVert2)
Descripción:	Devuelve verdadero si la arista compuesta por los dos vértices dados pertenece a dos triángulos.
Nombre:	CVector* kVerTri(int* arg_aiTriList, int arg_iCanTri, int* CantVert)

Descripción:	Devuelve los vértices de los triángulos que tiene un vértice común.
Nombre:	CVector* kVertDirCut(CVector* arg_akListVert, int* arg_piCantVert, int* arg_piCantVertDir, CVector arg_kDirec, CVector arg_kCommon)
Descripción:	Devuelve el vértice que está en dirección al corte y el que está en contra.
Nombre:	bool bVertFDir(CVector arg_kDir, CVector arg_kVert, CVector arg_kVertCom)
Descripción:	Devuelve verdadero si el vértice está a favor de la dirección del corte.
Nombre:	CVector* kVertFDir(CVector* arg_akListVert, int* arg_piCantVert, int* arg_piCantVertFN, CVector arg_kDir, CVector arg_kVertCom)
Descripción:	Devuelve los vértices que están a favor de la normal de la herramienta de corte.
Nombre:	int* aiTriInDir(CVector* arg_akListVert, int* arg_piCantVert, int* arg_akListTri, int* arg_piCanTri, int* arg_piCanTriInDir, CVector arg_kDir, CVector arg_kVertCom)
Descripción:	Devuelve los triángulos que están a favor de la normal de la herramienta de corte.
Nombre:	CVector* kSeparateVert(CVector arg_kVertCom, CVector arg_kDir, float arg_fDim)
Descripción:	Separar el vértice a cortar, crea dos nuevos vértices uno a favor de la normal, otro en contra.
Nombre:	void AddVert(CVector arg_kVert)
Descripción:	Adiciona un vértice a la lista de vértices.
Nombre:	void AssignVertNew(CVector arg_kVertCom, CVector arg_kDir, float arg_fDim, int* arg_aiLisTri, int* arg_piCanTri)
Descripción:	Asignar los vértices nuevos a cada triángulo afectado por el corte, según si está en contra o a favor de la normal.
Nombre:	int* aiTriCN(int* arg_aiTriList, int* arg_aiTriListFN)
Descripción:	Devuelve los triángulos que están en contra de la normal.
Nombre:	CVector* kVertCN(CVector* arg_akVertList, CVector* arg_akVertListFN)
Descripción:	Devuelve los vértices que están en contra de la normal.
Nombre:	CVector* akDteVertInList(CVector* arg_akVerListIni, int* arg_piCantVertIni, CVector arg_kVert)

Descripción:	Elimina un vértice de la lista de vértices.
Nombre:	CVector* akDteListVertInListVert(CVector* arg_akVerListIni, int* arg_piCantVertIni, CVector* arg_akVerListToDte, int* arg_piCantVertToDte)
Descripción:	Elimina de la lista de vértices una sublista pasada por parámetro.
Nombre:	CVector kMultWorlMatVect(CVector3 arg_kLocat)
Descripción:	Halla la posición real del objeto.
Nombre:	CVector* akAddVerTriToListVert(CVector* arg_akVerList, CVector* arg_akVerTri, int* arg_piCantVert)
Descripción:	Dado una lista de vértices adiciona los vértices de un triángulo en caso de no existir.
Nombre:	void AddVertToList(CVector* arg_akVerList, CVector arg_kVer, int* arg_piCantVert)
Descripción:	Adiciona un vértice a la lista de vértices.
Nombre:	CVector* akDeleteInPosition(CVector* arg_akVerListIni, int* arg_piCantVertIni, int arg_ipos)
Descripción:	Elimina un vértice de una posición dada.
Nombre:	void ChangeVertex(CVector arg_kVect, int arg_iPos)
Descripción:	Asignar un valor en la lista de vertices original, segun una posicion y un vértice.

Tabla 11: Descripción de la clase CTriMesh_STK.

Nombre: CCut	
Tipo de clase: Controladora	
Atributo	Tipo
m_pkMesh	CMesh*
m_pkToolCut	CToolCut*
m_pkCollision	CCollision
Para cada responsabilidad	
Nombre:	CCut (CMesh * arg_pkTriMesh, CToolCut* arg_pkToolCut, CCollision arg_pkCollision)

Descripción:	Constructor de la clase, se encarga de crear la colisión entre la malla y la o las herramientas de corte.
Nombre:	virtual void Cut (void * arg_vCut)
Descripción:	Maneja el proceso de corte.
Nombre:	virtual void DetectColl()
Descripción:	Identifica el triángulo y/o el punto de intersección donde exista colisión entre la malla y la herramienta de corte.

Tabla 12: Descripción de la clase CCut.

Nombre: CCutDestruct	
Tipo de clase: Controladora	
Para cada responsabilidad	
Nombre:	CDestructCut ()
Descripción:	Constructor de la clase.
Nombre:	void Cut (int arg_iTri)
Descripción:	Maneja el proceso de corte por el método destructivo, elimina el triángulo colisionado por la herramienta de corte.
Nombre:	void DetectColl()
Descripción:	Identifica el triángulo donde colisiona la malla con la herramienta.

Tabla 13: Descripción de la clase CCutDestruct.

Nombre: CCutSpte	
Tipo de clase: Controladora	
Para cada responsabilidad	
Nombre:	CCutSpte()
Descripción:	Constructor de la clase.
Nombre:	void Cut (CVector arg_iPtoIntersec, int arg_iTri)

Descripción:	Maneja el proceso de corte por el método separación.
Nombre:	void DetectColl()
Descripción:	Identifica el triángulo y el punto donde colisiona la malla con la herramienta de corte.

Tabla 14: Descripción de la clase CCutSpte.

Nombre: CCollision	
Tipo de clase: Controladora	
Atributo	Tipo
m_pkMesh	CMesh*
m_pkTool	CToolCut*
Para cada responsabilidad	
Nombre:	CCollision(CMesh* arg_pkMesh, CToolCut* arg_pkTool)
Descripción:	Constructor de la clase.
Nombre:	int iTriCollide ()
Descripción:	Método que devuelve el número del triángulo con el que colisiona la herramienta de corte.
Nombre:	CVector kPointCollide ()
Descripción:	Método que devuelve el punto por donde es interceptado el triángulo colisionado con la herramienta de corte.

Tabla 15: Descripción de la clase CCollision.

Nombre: CToolCut	
Tipo de clase: Controladora	
Atributo	Tipo
m_fWideSegment	float
Para cada responsabilidad	

Nombre:	CToolCut(float arg_fWideSegment)
Descripción:	Construtor de la clase.
Nombre:	CVector* kSegmentsList()
Descripción:	Devuelve la lista de 2 vértices que conforman el segmento de corte.
Nombre:	CVector kSegmentsListPos(int arg_Pos)
Descripción:	Devuelve el valor de un vértice del segmento dado su posición.
Nombre:	float fWideSegm()
Descripción:	Devuelve el ancho del segmento.
Nombre:	CVector kDireccion()
Descripción:	Devuelve la dirección del segmento.
Nombre:	CVector kNormal()
Descripción:	Devuelve la normal del triángulo de corte.

Tabla 16: Descripción de la clase CToolCut.

Nombre: CTMeshSTK	
Tipo de clase:	
Atributo	Tipo
m_pkSTKTriangle	CTriMesh*
m_kWorldMatrix	CMatrix4
Para cada responsabilidad	
Nombre:	CTMeshSTK (CTriMesh* arg_pkSTKTriangle, CMatrix4 arg_kWorldMatrix, float arg_fWideSegment)
Descripción:	Constructor de la clase.
Nombre:	CVector* kSegmentsList()
Descripción:	Devuelve la lista de 2 vértices que conforman el segmento de corte.
Nombre:	CVector kSegmentsListPos(int arg_Pos)
Descripción:	Devuelve el valor de un vértice del segmento dado su posición.

Nombre:	CVector kDireccion()
Descripción:	Devuelve la dirección del segmento.
Nombre:	CVector kNormal()
Descripción:	Devuelve la normal del triángulo de corte.
Nombre:	void UpdateWorldMat(CMatrix3 arg_kWorldMatrix, CVector3 arg_kLocation)
Descripción:	Actualizar la matriz global de un objeto.

Tabla 17: Descripción de la clase CTMeshSTK.

Nombre: CVector	
Tipo de clase: Entidad	
Atributo	Tipo
m_fX	float
m_fY	float
m_fZ	float
Para cada responsabilidad	
Nombre:	CVector (float arg_fX, float arg_fY, float arg_fZ)
Descripción:	Constructor de la clase, inicializa los valores de x,y,z.
Nombre:	void Normalize ()
Descripción:	Realiza la normalización.
Nombre:	float fModule ()
Descripción:	Devuelve el módulo de un vector.
Nombre:	float fDot (CVector arg_kVector3)
Descripción:	Calcula el producto escalar entre dos vectores.
Nombre:	void Invert ()
Descripción:	Invierte el vector.
Nombre:	float fSin(CVector arg_kVector)

Descripción:	Halla el seno entre dos vectores.
Nombre:	float fCos(CVector arg_kVector)
Descripción:	Halla el coseno entre dos vectores.
Nombre:	float fTan(CVector arg_kVector)
Descripción:	Halla la tangente entre dos vectores.
Nombre:	float fDistanceToPoint (CVector arg_kVector3)
Descripción:	Calcula la distancia de un segmento a un punto.
Nombre:	float fDistanceToRect2D(CVector arg_kVector1_3D, CVector arg_kVector2_3D)
Descripción:	Calcula la distancia entre dos rectas.
Nombre:	bool operator == (CVector arg_kVector3)
Descripción:	Operador de asignación.
Nombre:	CVector operator- (const CVector& arg_kVector3)
Descripción:	Operador de diferencia.
Nombre:	CVector operator+ (const CVector& arg_kVector3)
Descripción:	Operador de adición.
Nombre:	CVector operator * (float arg_fFactor)
Descripción:	Operador de producto.
Nombre:	CVector operator * (CVector arg_kVector3)
Descripción:	Producto vectorial.
Nombre:	float fAngle(CVector arg_kVector)
Descripción:	Calcula el angulo en radianes entre dos vectores.
Nombre:	float& fX();
Descripción:	Método de acceso al atributo X, devuelve su valor.
Nombre:	float& fY();
Descripción:	Método de acceso al atributo Y, devuelve su valor.
Nombre:	float& fZ();

Descripción:	Método de acceso al atributo Z, devuelve su valor.

Tabla 18: Descripción de la clase CVector.

Nombre: CPoint	
Tipo de clase: Entidad	
Atributo	Tipo
m_fX	float
m_fY	float
Para cada responsabilidad	
Nombre:	CPoint(float arg_fX, float arg_fY)
Descripción:	Constructor de la clase.
Nombre:	float& fX()
Descripción:	Método de acceso al atributo m_fX.
Nombre:	float& fY()
Descripción:	Método de acceso al atributo m_fY.
Nombre:	bool bPointInLine2D (CPoint arg_kPoint, CPoint arg_kRectPoint1, CPoint arg_kRectPoint2)
Descripción:	Verifica si un punto está en un segmento de recta 2D
Nombre:	CPoint* pkRectsIntersectionPoint (CPoint arg_kRectAPoint1, CPoint arg_kRectAPoint2, CPoint arg_kRectBPoint1, CPoint arg_kRectBPoint2)
Descripción:	Punto de intersección entre rectas 2D

Tabla 19: Descripción de la clase CPoint.

Nombre: CPlane3D	
Tipo de clase: Entidad	
Atributo	Tipo
m_kPto1	CVector

m_kPto2	CVector
m_kPto3	CVector
Para cada responsabilidad	
Nombre:	CPlane3D(CVector arg_kPto1, CVector arg_kPto2, CVector arg_kPto3)
Descripción:	Constructor de clase, inicializa los tres puntos que definen un plano.
Nombre:	CVector kPto1()
Descripción:	Método de acceso al miembro m_kPto1.
Nombre:	CVector kPto2()
Descripción:	Método de acceso al miembro m_kPto2.
Nombre:	CVector kPto3()
Descripción:	Método de acceso al miembro m_kPto3.
Nombre:	float fDValue()
Descripción:	Es el valor D de la ecuación ax + by + cz = D, la cual se utiliza para hallar la intercepción entre un plano y un segmento.
Nombre:	CVector kPNormal()
Descripción:	Devuelve la normal del plano.
Nombre:	float fArea(CVector arg_Pto1,CVector arg_Pto2,CVector arg_Pto3)
Descripción:	Devuelve el área del plano.
Nombre:	bool bCollideSegment(CSegment arg_kSegment,CVector& arg_kInterceptionPoint)
Descripción:	Devuelve verdadero si dos segmentos se interceptan, esta función comprueba además que el punto esté dentro del triángulo–plano interceptado. Utiliza las funciones kCollideSegment3D () la que devuelve el punto exacto de la colisión y fArea() la cual comprueba el sentido de orientación de los triángulos formados con el punto de contacto determinando si está dentro o fuera del triángulo en cuestión.
Nombre:	CVector kCollideSegment(CSegment arg_kSegment)
Descripción:	Determina el punto de intersección entre dos segmentos.
Nombre:	bool bIsPointInBorder(CVector kVector)

Descripción:	Devuelve verdadero si el punto está en el borde del plano.

Tabla 20: Descripción de la clase CPlane3D.

Nombre: CSegment	
Tipo de clase: Entidad	
Atributo	Tipo
m_kIniPto	CVector
m_kLastPto	CVector
Para cada responsabilidad	
Nombre:	CSegment(CVector arg_kIniPto,CVector arg_kLastPto)
Descripción:	Constructor de la clase se encarga de inicializar los puntos extremos de un segmento.
Nombre:	CVector kIniPto()
Descripción:	Método de acceso al miembro m_kIniPto.
Nombre:	CVector kLastPto()
Descripción:	Método de acceso al miembro m_kLastPto.
Nombre:	CVector kVectorDirector()
Descripción:	Devuelve el vector director de un segmento.
Nombre:	bool bCollideSegment(CSegment arg_kSegment)
Descripción:	Devuelve verdadero si dos segmentos colisionan.
Nombre:	CVector kCollideSegment(CSegment arg_kSegment)
Descripción:	Determina el punto exacto de colisión entre dos segmentos.
Nombre:	bool bPointIsBetween(CVector)
Descripción:	Devuelve verdadero si un punto está en un segmento determinado.
Nombre:	bool bPointIsInLine(CVector arg_kVector)
Descripción:	Devuelve verdadero si un punto pertenece a una recta.
Nombre:	CPoint kXFunT()

Descripción:	Devuelve el valor de x despejado en función de t para la ecuación paramétrica de la recta.
Nombre:	CPoint kYFunT()
Descripción:	Devuelve el valor de y despejado en función de t para la ecuación paramétrica de la recta.
Nombre:	CPoint kZFunT()
Descripción:	Devuelve el valor de z despejado en función de t para la ecuación paramétrica de la recta.
Nombre:	void Move(CVector arg_kVDistance)
Descripción:	Método de acceso a miembros donde se cambia los valores de m_kIniPto y m_kLastPto.

Tabla 21: Descripción de la clase CSegment.

4.4 Diagramas de secuencia

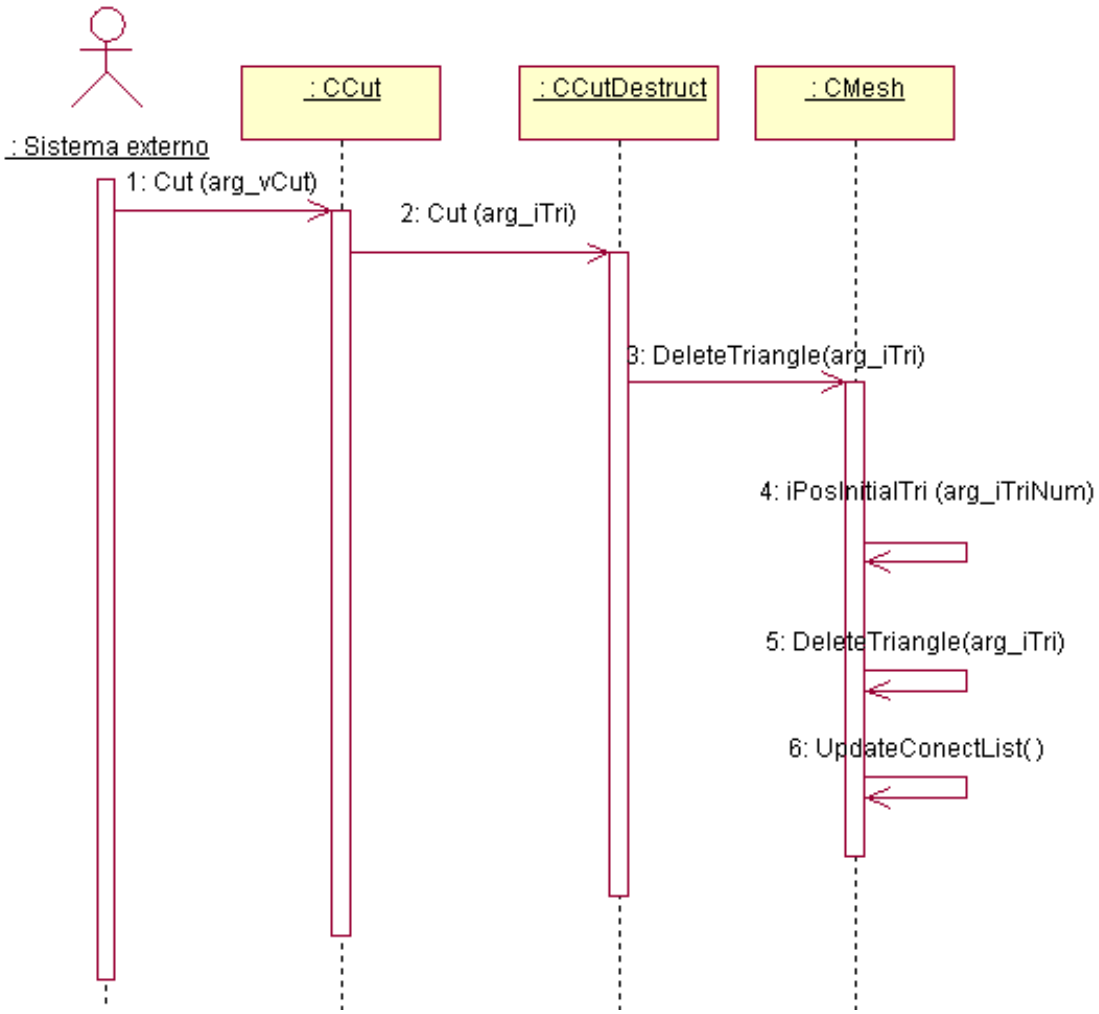
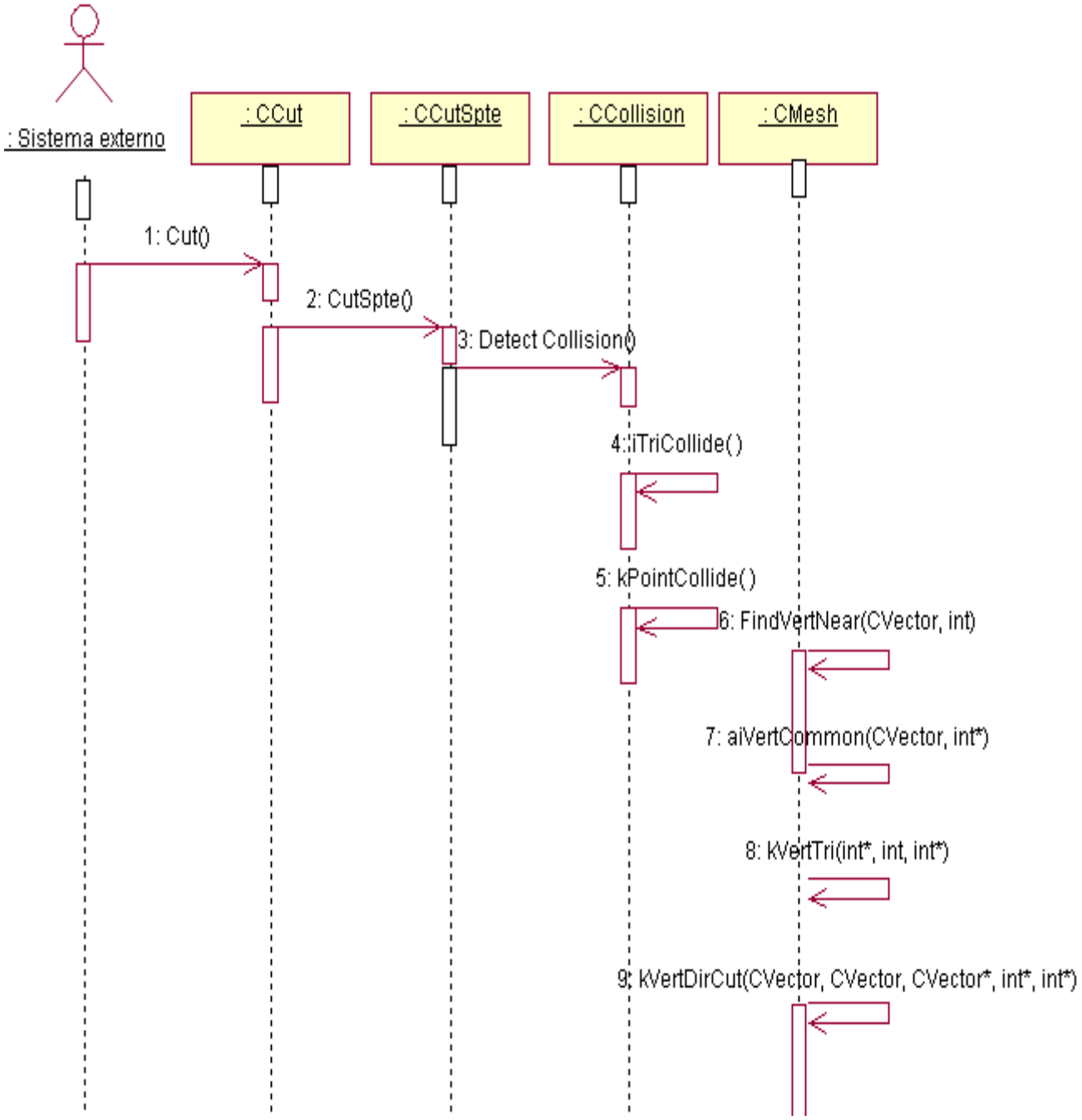


Fig. 27: Diagrama de secuencia CU Destruir triángulo.


: Sistema externo
: CCut
: CCutSpte
: CCollision
: CMesh
1: Cut()
2: CutSpte()
3: Detect Collision()
4: iTriCollide()
5: kPointCollide()
6: FindVertNear(CVector, int)
7: aiVertCommon(CVector, int*)
8: kVertTri(int*, int, int*)
9: kVertDirCut(CVector, CVector, CVector*, int*, int*)

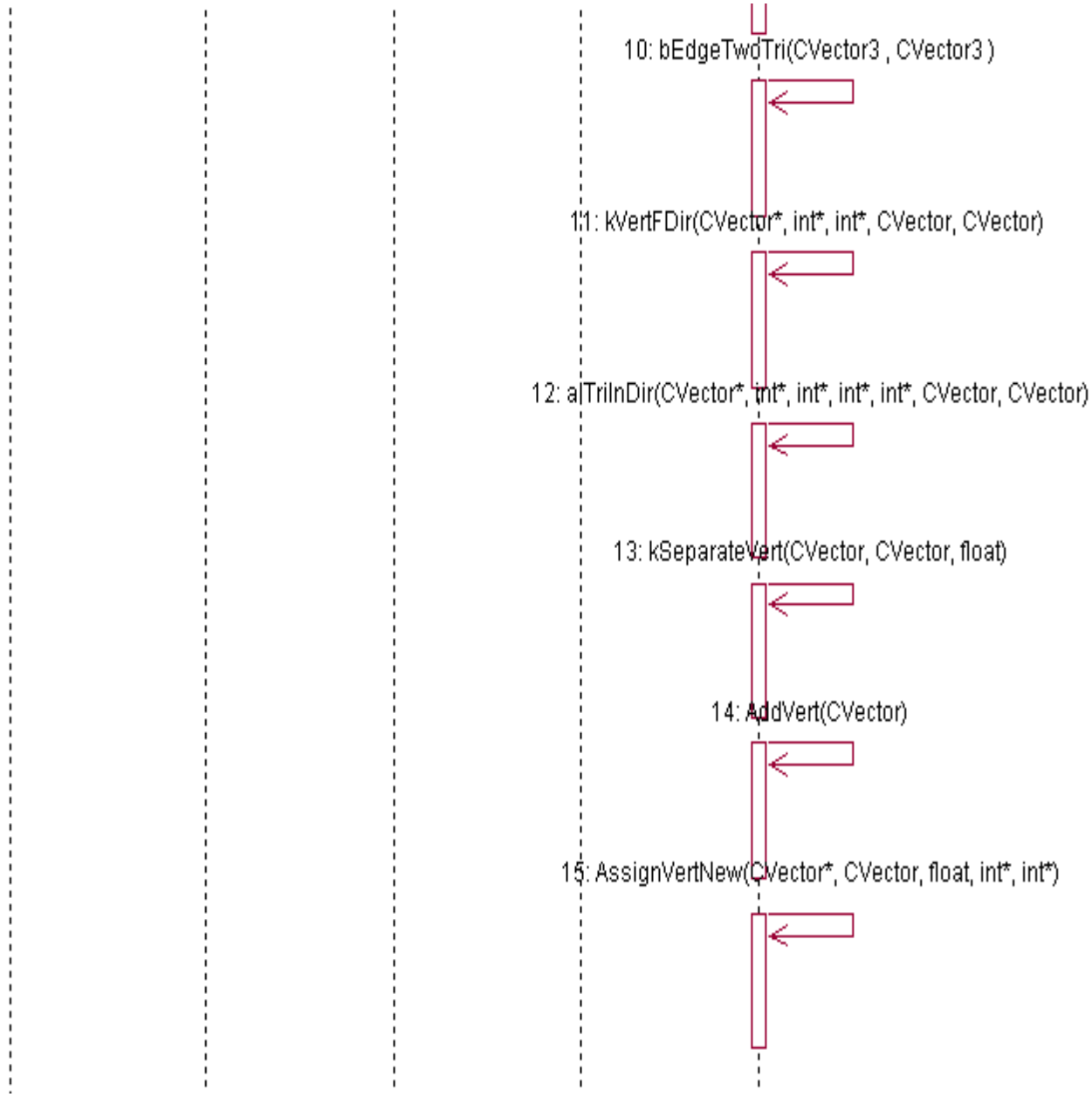


Fig. 28: Diagrama de secuencia CU Separar la malla.

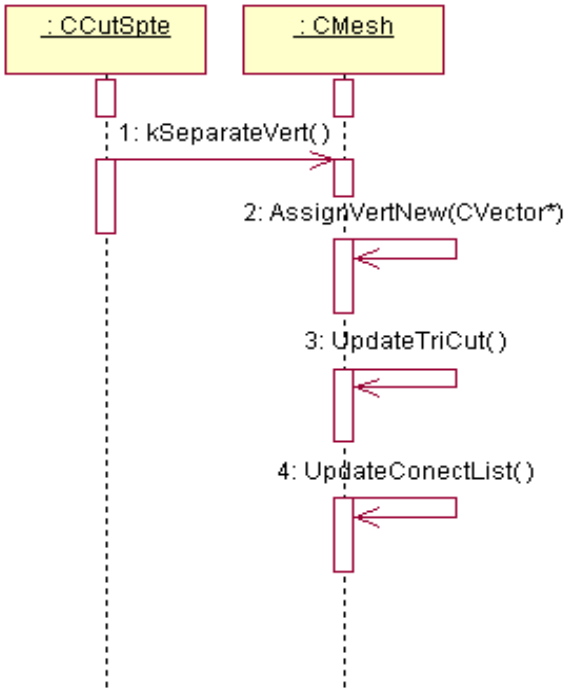


Fig. 29: Diagrama de secuencia CU Gestionar triángulos, sección Separar vértices.

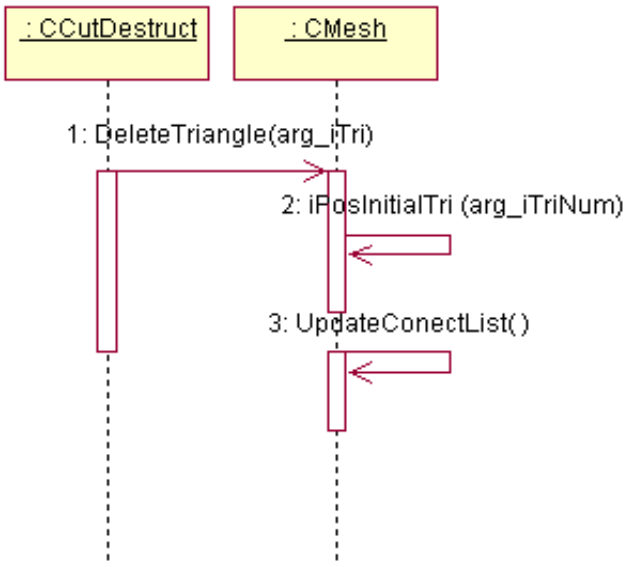


Fig. 30: Diagrama de secuencia CU Gestionar triángulos, sección Eliminar triángulos.

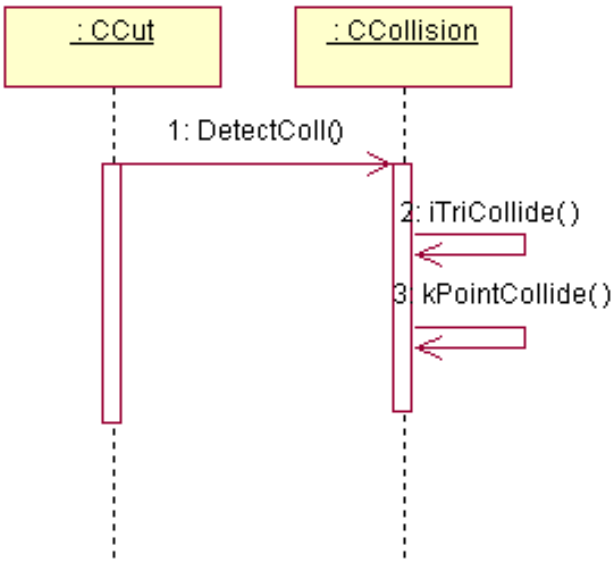


Fig. 31: Diagrama de secuencia CU Detectar intersecciones.

4.5 Diagramas de componentes

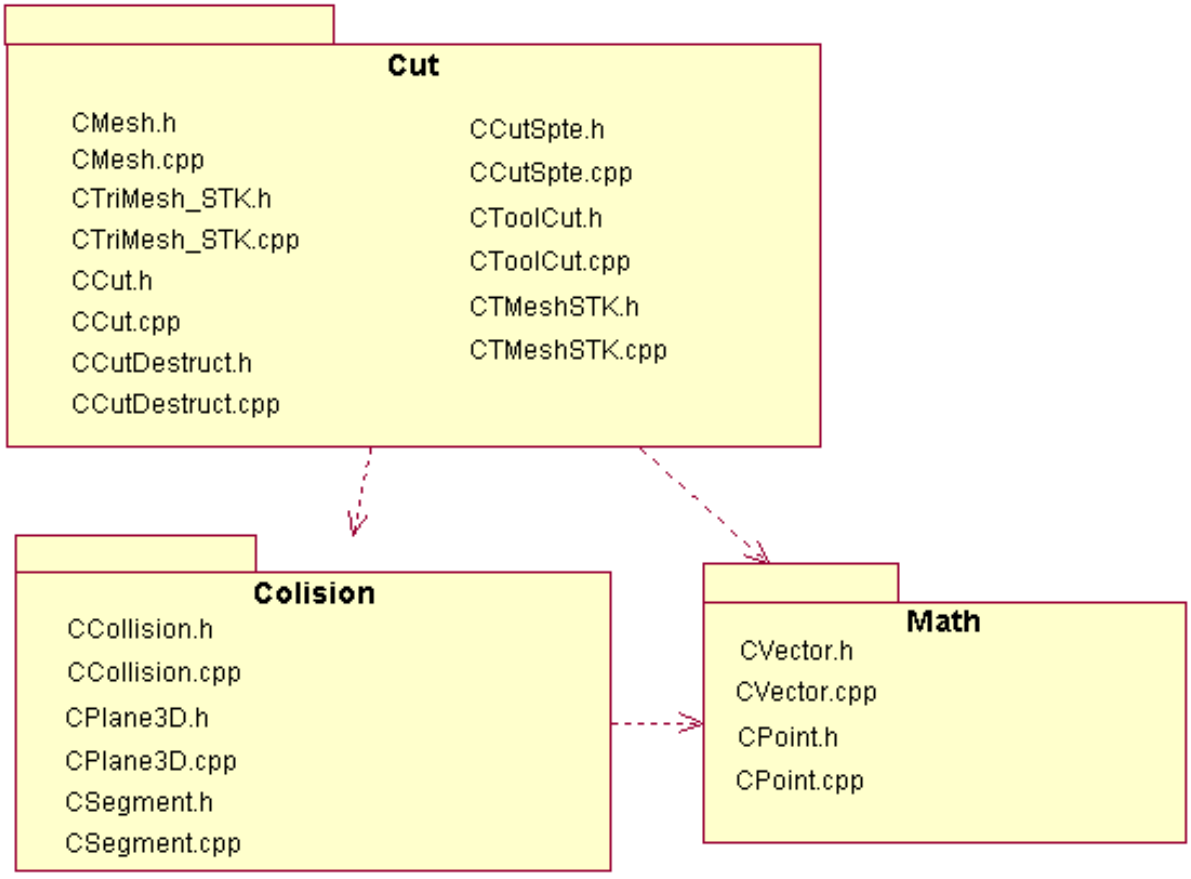


Fig. 32: Diagrama de relación entre paquetes de componentes.

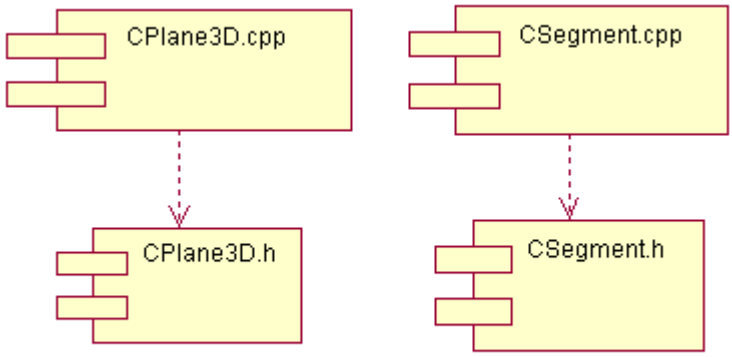


Fig. 33: Diagrama de relación de componentes: Paquete Colision.

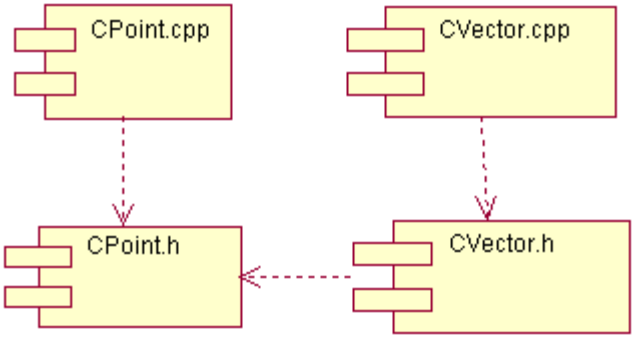


Fig. 34: Diagrama de relación de componentes: Paquete Math.

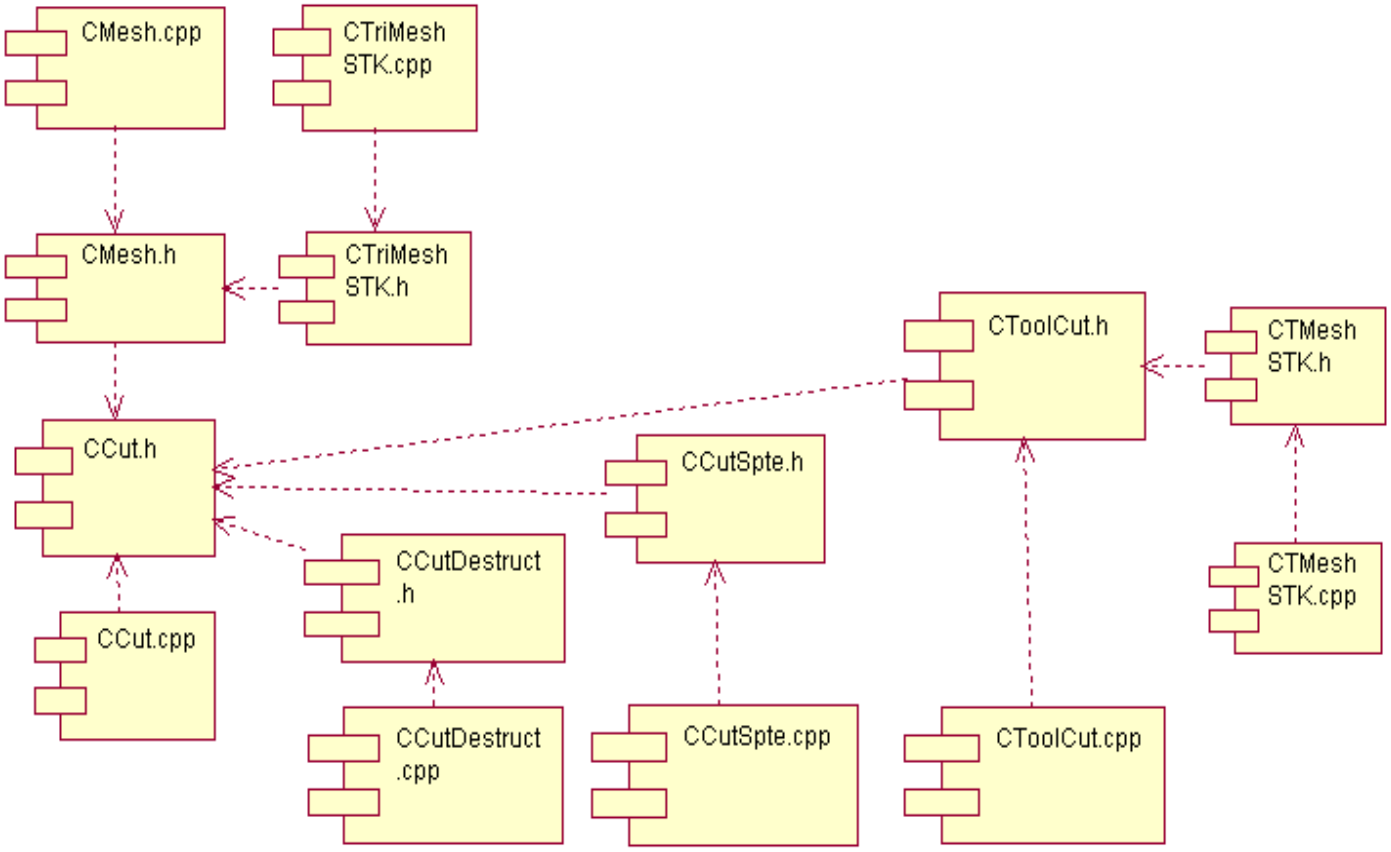


Fig. 35: Diagrama de relación de componentes: Paquete Cut.

Consideraciones del capítulo

En el recién concluido capítulo se realizó todo lo correspondiente al diseño y la implementación para el módulo de corte, a lo largo del mismo se mostraron todos los diagramas y artefactos pertenecientes a estos dos flujos de trabajo de ingeniería de software.

Luego de concebir el diseño del sistema, mediante su diagrama de clases, donde se muestra claramente las relaciones entre ellas, se modeló mediante el diagrama de componentes y las relaciones entre paquetes, la parte física del sistema, consolidando el proceso de desarrollo para pasar a la programación del módulo, a partir de los casos de uso.

Conclusiones generales

A lo largo de este trabajo se realizó un estudio de las principales técnicas de corte para el uso de simuladores, enfatizando en sus ventajas y desventajas, en aras de brindar una solución eficiente al problema planteado, atendiendo a las necesidades del cliente. De aquí se propone la solución que da cumplimiento al objetivo trazado.

Posteriormente se realizó el proceso de Ingeniería de Software donde se describió la solución propuesta en términos de los flujos de trabajo de requerimientos, diseño e implementación, ajustándose a la metodología de desarrollo de software utilizada; que en este caso fue el Proceso Unificado del Software (RUP).

Finalmente se propone un módulo genérico y flexible, capaz de realizar cualquiera de las dos técnicas de corte desarrolladas a lo largo del presente trabajo, la técnica destructiva o la de separación, este se puede acoplar a cualquier motor gráfico con solo adicionarle una clase interfaz, pues todo el trabajo y manejo de las operaciones con la malla y la herramienta de corte lo realiza el propio módulo.

Recomendaciones

Para continuar este trabajo y complementarlo, se recomienda incluir o desarrollar:

- Generar la ranura interna con el objetivo de representar el interior en una incisión.
- Desarrollar un módulo de colisiones con algoritmos más eficientes para su mejoría.
- Tratar el refinamiento de la malla para lograr mayor realismo.
- Acoplar el módulo a uno deformable para lograr mejor visualización y un proceso de corte más real y completo.

Referencias bibliográficas

[1] Levis Diego, “Realidad Virtual y Educación”, 1997.

http://www.diegolevis.com.ar/secciones/Articulos/master_eduvirtual.pdf

[2] Realidad Virtual y Simuladores virtuales quirúrgicos. Instrumentación Biomédica. Doctorado.

[3] M. Bro-Nielsen. Finite element modeling in surgery simulation. Journal of the IEEE, 86(3):490 503, 1998.

[4] S. Cotin, H. Delingette, and N. Ayache. “A hybrid elastic Model allowing real-time cutting, deformations and force-feedback for surgery training and simulation”. The Visual Computer, 16(8):437 452, 2000.

[5] C. Forest, H. Delingette N. Ayache, “Cutting simulation of manifold volumetric mehses”, Proceedings Medical Image Computing and Computer Assisted Intervention (MICCAI), 2002.

[6] A. Mazura and S. Seifert. “Virtual cutting in medical data.” In Proc. of Medicine Meets Virtual Reality, pages 420 429, 1997.

[7] A. Mor and T. Kanade. Modifying soft tissue models: “Progressive cutting with minimal new element creation.” In MICCAI Proc., pages 598 607, 2000.

[8] D. Bielser et al. A state machine for real-time cutting of tetrahedral meshes. In Proc. of Pacific Graphics, pages 377 386, 2003.

[9] Han-Wen Nienhuys, A. Frank van der Stappen, “A surgery simulation supporting cuts and finite element deformation”. MICCAI, 2001.

[10] C. Mendoza - C. Laugier - O. Galizzi - F.Faure, Simulating Cutting in Surgery Applications using Haptics and Finite Element Models, Proceedings MICCAI, 2003.

[11] D.Servy, M. Harders, G. Szekely. A new approach to cutting into finite element models. In MICCAI'01 Proc, páginas 425 - 433, 2001.

[12] C.D. Bruyns, K. Montgomery, Generalized intersections using virtual tools within the spring framework: Probing, piercing, cauterizing and ablating in: Proc. Medicine Meets Virtual Reality 2002, pp 74- 78, 2002.

[13] Shewchuk J, Lectures notes on Delaunay mesh generation, Technical report, Department of Electrical Engineering and Compurter Science, University of California at Berkeley, 1999.

[14] Germán Sánchez Torres, Sandra P. Mateus Santiago y John William Branch. “Construcción de Mallas Triangulares no Estructuradas Aplicado al Ajuste de Superficies de Objetos Tridimensionales".

[15] LaMothe A, “Tricks of the 3D Game Programming Gurus” Indianapolis, SAMS, 2003, 1601.

[16] Campbell, R. y Flynn, P. “A survey of free-form objects representation and recognition techniques”, Comput. Vis. Image Underst. 81(2), 166–210, 2001.

[17] Nachiappan S. Kapoor S. Prem K. “Geometry Based Connectivity Compression of Triangular Meshes” Illinois Institute of Technology, USA 2006.

[18] Kobbelt L., Botsch M.” A Survey of Point-Based Techniques in Computer Graphics” Computer Graphics Group, RWTH Aachen University, Germany 2004.

[19] Guo X, Qin H “Point-Based Dynamic Deformation and Crack Propagation” Stony Brook University, New York.

[20] AnimaTek International, Inc., “Caviar Technology,” Web page: http://www.animatek.com/products_caviar.htm

[21] Mark Pauly, Richard Keiser, Leif P. Kobbelt, Markus Gross. Shape Modeling with Point-Sampled Geometry.

[22] Gross M. “Getting to the Point?” Computer Graphics Laboratory ETH Zürich, Switzerland 2006.

[23] Grossman, J. and Dally, W. “Point Sample Rendering,” Proc. Eurographics Rendering Workshop, 1998.

[24] Pauly M., Keiser R., Kobbelt L. P., GrossM.: Shape modeling with point-sampled geometry. ACM Transactions on Graphics (TOG) 22, 3 (2003), 641–650.

[25]Zwicker M., Rsnen J., Botsch M., Dachsbacher C., Pauly M.: Perspective accurate splatting. In *Proceedings of Graphics Interface* (2004).

[26] S. Cotin, "Modéles anatomiques déformables en temps-réel." Thése de doctorat, INRIA Sophia Antipolis – Université de Nice, Sophia Antipolis, 1997, cited on pages 9, 10, 12, 23, 37, 46,50, 50, 54, 54, 55, 55, 68, 68, 69, 73, 172.

[28] C.Mendoza, C. Laugier y B. Cason, "Virtual Reality Cutting Phenomena using Force Feedback for Surgery Simulations".

[29] D. Bielser, V. A. Maiwald, and M. H. Gross. "Interactive cuts through 3-dimentional soft tissue". In EUROGRAPHICS'99, volume 18, pages C31–C38, 1999.

[30] Hui Zhang, Shahram Payandeh y John Dill, "Simulation of Progressive Cutting on Surface Mesh Model".

[31] Gibson S. F. F., Mirtich B. "A Survey of Deformable Modeling in Computer Graphics" A Mitsubishi Electric Research Lab, Noviembre 1997.

[32] Aulignac, D. (2001). "Modelisation de l'interaction avec des objets deformables en temps-réel pour des simulateurs medicaux". PhD thesis, Institute Nationale Polytechnique de Grenoble, France.

[33] Waters, K. (1987)."A muscle model for animating three dimensional facial expression". In Proceedings of ACM SIGGRAPH.

[34] Terzopoulos D, Waters K "Physically-Based Facial Modeling, Analysis, and Animation" Journal of Visualization and Computer Animation, 1(2):73–80, 1990.

[35] Prietoni N. "Physically Based Deformable Objects in Computer Graphics." Universidad de Geneva, Diciembre 2005.

[36] Turner, M. J., Clough, R. W., Martin, H. C., Topp, L. J. "Stiffness and deflection analysis of complex structures", J. Aero., 1956.

[37] Bro-Nielsen M., Cotin S. “Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation” Technical University of Denmark, Lyngby, Denmark 1996.

[38] Al-khalifah A., Roberts D. “Survey of modeling approaches for medical simulators” Centre for Virtual Environments, the University of Salford, Manchester, UK 2004.

[39] Doug L. J., Dinesh K. P. “Accurate Real Time Deformable Objects” University of British Columbia, Canada, 1999.

[40] Koppel D., Wang Y., Chandrasekaran Sh.” Toward Real-Time, Physically-Correct Soft Tissue Behavior Simulation” University of California, USA 2003.

[41] Nealen A, Müller M, Keiser R, Boxerman E and Carlson M “Physically Based Deformable Models in Computer Graphics” EUROGRAPHICS 2005.

[42] Rodríguez O. Lester, Fernández R. Leonardo, “Técnicas de corte para mallas superficiales”, Universidad de las Ciencias Informáticas, Cuba, 2008, 95.

Anexo 1: Glosario de términos

B

Biblioteca de clases: Clases de programación orientada a objetos suministradas por terceros.

C

Cirugía de Mínimo Acceso: La cirugía laparoscópica, sin ingreso o mínimamente invasiva es una técnica quirúrgica practicada a través de pequeñas incisiones, asistida de una cámara de video que permite cirujano accionar sobre el campo quirúrgico, evitando los grandes cortes de bisturí requeridos por la cirugía abierta o convencional y posibilita un periodo post-operatorio mucho más rápido y confortable.

C++: Lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos (POO). C++ está considerado por muchos como uno de los lenguajes más potentes, debido a que permite trabajar tanto a alto como a bajo nivel.

E

3D Engines: Herramientas que facilitan la visualización de escenas 3D en ordenadores.

H

Hardware: Componentes físicos de una computadora o de una red (a diferencia de los programas o elementos lógicos que los hacen funcionar).

M

Malla Triangular: Conjunto de vértices y aristas que en su composición forman triángulos y estos a su vez la malla.

O

Objeto Deformable: Cuerpo que por sus características físicas están expuestos a cambios en su forma debido a la acción de agentes externos.

R

Realidad Virtual: Simulación generada por computadora de imágenes o ambientes tridimensionales interactivos con cierto grado de realismo físico o visual.

S

Simulación: Intento de recrear el comportamiento real de sistemas a través de modelos aproximados.

Sistemas de Realidad Virtual: Sistema informático interactivo que ofrece una percepción sensorial al usuario de un mundo tridimensional sintético que suplanta al real.

Software: Equipamiento o soporte lógico de un computador digital, que comprende el conjunto de los componentes necesarios para hacer posible la realización de una tarea específica.

T

Tetraedro: Es un poliedro formado por cuatro caras que son triángulos equiláteros, y cuatro vértices en cada uno de los cuales concurren tres caras. Es uno de los cinco poliedros perfectos llamados sólidos platónicos.

Tiempo Real: Termino usado en el mundo de los gráficos por computadora para las aplicaciones interactivas con respuesta en intervalos de tiempo que parecen instantáneos al usuario, usualmente más de 16 fotogramas por segundo.

Topología: Las propiedades de los objetos con independencia de su tamaño o forma.

V

Virtual: Término utilizado para hacer referencia a algo que no tiene existencia física o real, sólo aparente.

Anexo 2: Estándares de codificación

Nombre de los ficheros:

Se nombrarán los ficheros .h y .cpp de la siguiente manera:

NameofUnits.h

Constantes:

Las constantes se nombrarán con mayúsculas:

MAX =

Tipos de datos:

Los tipos se nombrarán siguiendo el siguiente patrón:

Clases: class CClassName;

Indicando con “C” que es una clase.

Anexo 3: Declaración de variables

Los nombres de las variables comenzarán con un identificado del tipo de dato al que correspondan, como se muestra a continuación. En el caso de que sean variables miembros de una clase, se le antepondrá el identificador “” (en minúscula) y en caso de ser argumentos de algún método, se les antepondrá el prefijo “arg_”.

Tipos simples:

bool bVarName;

int iName;

unsigned int uiName;

float fName;

bool m_bMemberVarName; //variable miembro

float arg_fName; //variable argumento

Indice de figuras

Indice de tablas