UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Posgrado en Ciencia e Ingeniería de la Computación

Modelado Computacional de Cortes de Tejido Blando

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN INGENIERÍA

(COMPUTACIÓN)

P R E S E N T A:

SERGIO TEODORO VITE

DIRECTOR DE TESIS

Dr. Fernando Arámbula Cosío

México D. F., Enero 2012.

TEODORO VITE, Sergio.

Modelado Computacional de Cortes de Tejido Blando
México, UNAM.
Posgrado en Ciencia e Ingeniería de la Computación, 2012.

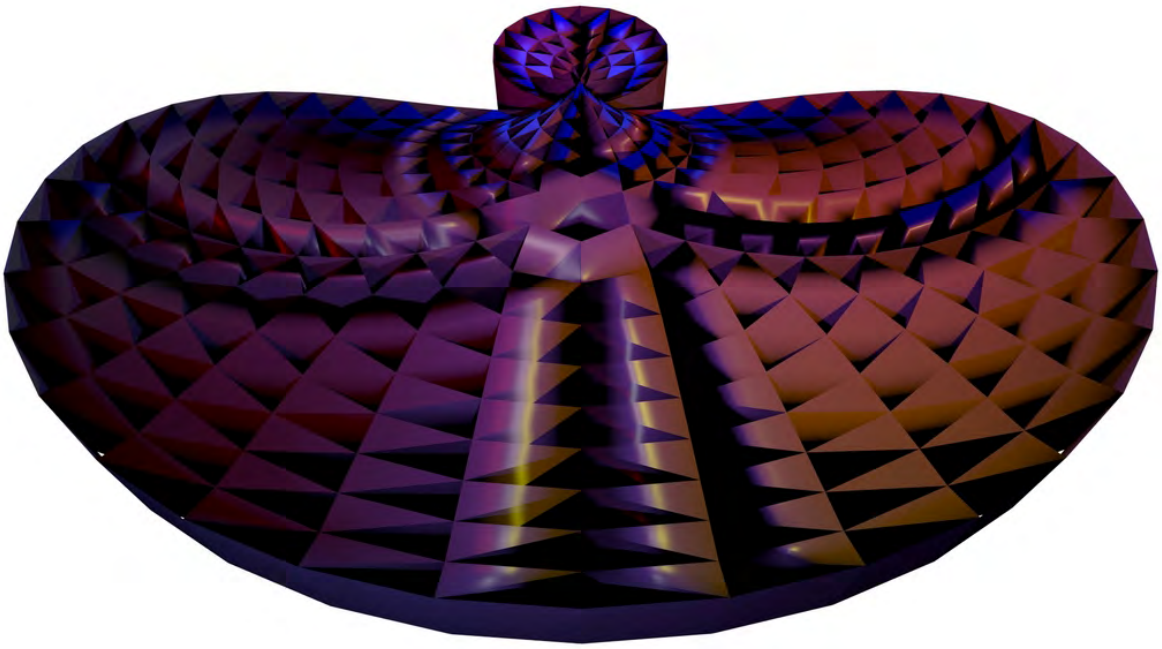
Modelado Computacional de Cortes de Tejido Blando

Universidad Nacional Autónoma de México.
Ciudad Universitaria, 04510, México, D .F.
Edición 2012.

Impreso y Hecho en México.

Modelado
Computacional de Cortes
de Tejido Blando

Sergio Teodoro Vite



Universidad Nacional
Autónoma de
México

UNAM
POSGRADO

CCADET
CENTRO DE CIENCIAS APLICADAS Y
DESARROLLO TECNOLÓGICO

A mis padres

Julia Vite Franco y Emiliano Teodoro Mojica

Resumen

El reto de representar computacionalmente estructuras del mundo real ha cobrado un gran interés en los últimos años, principalmente en áreas como la medicina, las ciencias de la tierra y los videojuegos. Ésto ha provocado el desarrollo de hardware y software especializado para tareas de representación tridimensional, simulación física e interacción hombre-máquina. Particularmente en el ámbito médico ha llamado la atención el desarrollo de simuladores que recrean situaciones quirúrgicas, donde un médico se involucra como usuario en un entorno virtual para llevar a cabo tareas como corte, sutura, exploración y coagulación, poniendo a prueba sus habilidades y conocimientos.

Por esta razón, este trabajo se plantea el estudio de dos elementos presentes en casi todos los procedimientos quirúrgicos: el tejido blando y la acción de corte sobre éste. Ambos reproducibles en una computadora mediante modelos.

Física y matemáticamente, el tejido blando puede modelarse como un objeto deformable con propiedades mecánicas y espaciales. Computacionalmente, usando un algoritmo de solución numérica que permita resolver el modelo fisico-matemático, acompañado de alguna técnica de visualización. Actualmente estos métodos se clasifican en tres tipos: heurísticos, basados en la mecánica de los medios continuos e híbridos. Por otra parte, la acción de corte involucra, además del modelo físico-matemático y computacional, métodos que operen sobre una malla geométrica (que representa el tejido), con el fin de dividirla o remover parte de ella. En tal caso, este trabajo presenta un estudio sobre cinco metodologías actuales: separación de nodos, retracción, eliminación de elementos, deformación plástica y cambio topológico.

El objetivo final es proveer un estudio completo sobre las diferentes modelos computacionales para la simulación de tejido blando, métodos para generación de mallas 3D y métodos de simulación de cortes. Todos ellos para ser usados en simuladores de cirugía con fines de entrenamiento médico.

Lista de Sinodales

Dr. Fernando Arámbula Cosío
Centro de Ciencias Aplicadas y Desarrollo Tecnológico, UNAM.

Dra. María Elena Martínez Pérez
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, UNAM.

Mat. María Concepción Ana Luisa Solis González Cosío
Facultad de Ciencias, UNAM.

Dr. Guillermo de Jesús Hernández García
Instituto de Geofísica, UNAM.

Dr. Sergio Durán Ortíz
Instituto Nacional de Rehabilitación.

Índice de Contenido

CAPÍTULO 1

Introducción

En los últimos años, la creciente necesidad de sistemas de cómputo capaces de simular situaciones del mundo real, ha provocado la inclusión de software y hardware dedicado en una gran variedad de campos del conocimiento. Particularmente en el ámbito de la investigación biomédica ha surgido un área de interés, que involucra tanto la biología de los seres vivos, la medicina, la física, las matemáticas, la robótica y la computación, para desarrollar simuladores capaces de recrear situaciones quirúrgicas, donde un médico se involucra como usuario en un entorno generado por computadora, usando como base de interacción, algún dispositivo de hardware, que comunica ambos espacios: el real y el sintético [53][38]. Además, la necesidad de evaluación, cuantificación y medición de variables relevantes en las ciencias de la salud, ha permitido extender el campo de investigación en nuevas metodologías, partiendo del modelado físico-matemático, para resolver problemas de incertidumbre sobre el comportamiento de cierta estructura biológica, por ejemplo, el tejido [40][88].

Física y matemáticamente, el tejido blando puede modelarse como un objeto deformable con propiedades mecánicas involucrado en un espacio de tres dimensiones [18]; computacionalmente, usando un algoritmo de solución tanto en tiempo de procesamiento (cálculos), como de visualización. En cuanto a resolución del procesamiento, existen reportados una gran cantidad de métodos, actualmente clasificados en tres grupos principales: métodos heurísticos, métodos basados en la mecánica de los medios continuos y métodos híbridos. Meier [43] reporta en 2004, un estudio sobre los métodos de solución para objetos deformables, siendo una de sus grandes aportaciones, la comparación cuantitativa y cualitativa para determinar cuáles de ellos son mejores para implementarse en sistemas de simulación de cirugía en tiempo real.

En sistemas donde se requiere no sólamente la simulación de la deformación del tejido, sino también la remoción de parte de éste, como sucede en una cirugía, se requiere

de un método de simulación para cortes. Para este caso, se pueden mencionar tres tipos de enfoques: la pérdida de elementos que constituyen una malla, la reconstrucción del objeto en tiempo real (cambio topológico) [74][9] y el cambio de forma. La primera considera la eliminación de elementos discretizados del objeto como nodos o tetraedros; la segunda, supone la estabilidad de un objeto al perder elementos conectivos, reconstruyéndose cada vez que son eliminados elementos [14][73]. Este último, por su naturaleza, involucra un alto costo computacional, sobre todo al procesar mallas geométricas muy densas. Un tercer enfoque, menos reportado, se basa en la misma deformación del objeto. En cierta forma, este último método presupone una simulación de la propia simulación, al emular cortes con deformar plásticamente un objeto [76].

Muchos simuladores de cirugía han incorporado estas técnicas. Aquí algunos ejemplos:

- Simulador de resección de la próstata[62][63] (México, 2009).
- Simulador de laparoscopía gástrica[41](Brasil-EUA, 2009).
- Simulador de Paracentesis[23](Italia, 2005).
- Simulador de histeroscopía[45] (USA, 2001).
- Simulador de laparoscopía del conducto Biliar[7](USA, 2001).
- Simulador de entrenamiento de cirugía endoscópica[38](Alemania, 2000).
- Simulador de cirugía abdominal[11](USA, 1998).

Por lo tanto, dado el amplio campo de estudio que ofrecen los simuladores computarizados de cirugía, este trabajo de tesis se concentra en las metodologías para simular tejido blando y las acciones de corte sobre éste.

Sección 1.1

Objetivo, metodología y alcances

El objetivo principal es proponer un modelo general para la simulación computacional de cortes sobre estructuras blandas, poniendo como caso de aplicación directa al tejido. El modelo genérico se basa entonces en tres elementos principales: un objeto deformable, una malla geométrica que lo representa y una interacción (corte) con esta última.

Para lograrlo, se propuso la siguiente metodología:

1. Investigación: Descripción bibliográfica de los métodos de simulación para objetos deformables y la simulación de cortes, reportados en la literatura hasta el momento.

2. Simulación de objetos deformables: Simulación de objetos deformables usando geometrías básicas, tales como un cubo, un cilíndro y una esfera. Para este punto se usaron dos métodos: masas-resortes y uno híbrido.

3. Simulación de cortes sobre el objeto deformable: Simulación de cortes sobre geometrías básicas empleando los métodos investigados.

4. Aplicación de deformación y cortes en un simulador de cirugía para simular tejido blando. La aplicación inmediata fue un simulador de cirugía de próstata, en desarrollo por el Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la UNAM.

El segundo y tercer puntos, implicaron la implementación de software para la simulación de objetos deformables y cortes. Se llevó a cabo para ello, una etapa de pruebas preeliminares, para explorar el software y hardware de desarrollo existente y su posible reutilización. Se trabajó en la plataforma *Windows* con *Visual Studio 2008/2010* como software de desarrollo, *SDK PhysX 2.8.1* de la compañía *nVidia*, como motor de física, *Qt 4.7.3* para agregar interfaces de usuario (GUI) y *OpenGL* para la visualización. Todo programado bajo lenguaje C/C++. Adicionalmente se usaron otras herramientas, principalmente para el modelado de objetos geométricos, como *Blender*, así como algunas bibliotecas propias del grupo de desarrollo del Laboratorio de Análisis de Imágenes y Visualización del CCADET.

Por la motivación principal, que corresponde a la incorporación de este trabajo en simuladores computacionales de cirugía, se definieron los siguientes alcances:

1. Elaborar un trabajo documentado sobre el estado del arte en el desarrollo de metodologías para simulación computacional de objetos deformables, representación geométrica y cortes sobre éstas.

2. Desarrollar un software de simulación de objetos deformables para fines demostrativos; usando geometrías básicas, como un cubo, un cilíndro y una esfera.

3. Desarrollar un software de simulación de cortes de objetos deformables básicos para fines demostrativos y educativos.

4. Adaptar algunas de las metodologías de simulación de tejido blando y cortes al simulador de cirugía de próstata del CCADET [62].

Sección 1.2

Relevancia y Contribución del trabajo

Actualmente en el país existen muy pocos grupos de desarrollo que trabajen en la implementación de simuladores de cirugía, y más aún, con el desarrollo de metodologías de cómputo para su incorporación en éstos. Por tanto, la principal contribución de este trabajo radica en el potencial que ofrecen las metodologías computacionales para ser aplicadas en simuladores para entrenamiento quirúrgico. El impacto social de esto último, es poder ofrecer a los médicos una alternativa de aprendizaje sin riesgo para el paciente. Adicionalmente, este trabajo también busca contribuir con un contenido bibliográfico, en español, sobre objetos deformables, representación tridimensional de estructuras y simulación de corte sobre éstas, desde un punto de vista computacional, que motive su estudio y encuentre más aplicaciones.

Sección 1.3

Organización de la tesis

Con base en los objetivos, el trabajo se encuentra estructurado en tres capítulos metodológicos, un capítulo de experimentos y resultados, y un capítulo de conclusiones. Adicionalmente se presentan tres apéndices, referenciados durante todo el trabajo, que complementan el contenido.

En el Capítulo 2 se aborda el estudio de los modelos biomecánicos, que sirven como base para el planteamiento de metodologías de solución computacional al problema de la deformación de objetos. Y dado que un tejido blando puede ser asociado con un comportamiento elástico, éste puede ser representado en una computadora mediante modelos deformables. Así se presentan los tres enfoques principales para el modelado de este tipo de estructuras: los métodos heurísticos, los métodos basados en la mecánica de los medios continuos y los métodos híbridos. No sin antes estudiar un poco de la biología del tejido y aterrizar la idea de su estructura. Finalmente, se describen los paquetes de software existentes para la simulación computacional y dos casos de estudio, correspondientes a simuladores de cirugía para entrenamiento médico. El primero, un simulador de laparoscopía esofágica, desarrollado por la Universidad Federal de Río Grande del Sur de Brasil y el Instituto Politécnico de Rensselaer en Estados Unidos. El segundo, un simulador para entrenamiento en cirugía de la próstata, en desarrollo por el Centro de Ciencias Aplicadas y Desarrollo Tecnológico de la UNAM.

Dado que todo método de deformación opera sobre un conjunto de elementos geométricos, en el Capítulo 3 se presentan algunas metodologías para la construcción de

geometrías de volumen; es decir, mallas formadas por vértices y tetraedros. Se inicia desde el concepto básico del tetraedro, con la formación de geometrías básicas como un cubo, una caja, una esfera y un cilíndro, hasta la generación de geometrías más complejas, como estructuras anatómicas. Los métodos descritos para tal tarea incluyen: la generación de estructuras tubulares a partir de imágenes, la generación de estructuras complejas a partir de superficies y otros que parten de nubes de puntos. Es importante mencionar que gran parte de este capítulo contiene propuestas originales, propias de este trabajo y que por tanto, son susceptibles a mejoras u optimizaciones. También se propone una especificación para la portabilidad de geometrías de volumen, el formato VOG (*Volume Object Geometry*), de fácil manejo y configuración para definir mallas de tetraedros. Al final del capítulo, se presenta un caso de estudio con la construcción de un modelo de la uretra masculina a partir de una geometría cilíndrica regular y la construcción de una malla de la próstata a partir de imágenes de ultrasonido.

Todo el estudio sobre la geometría de una estructura deformable y sus métodos computacionales relacionados, se concretan en el Capítulo 4, en el cual se emplea una malla que simula un tejido blando para ser operada mediante una acción de corte. Este capítulo se divide en dos partes, en una se describen los conceptos fundamentales de la acción quirúrgica de corte, las técnicas y los instrumentos usados, desde el punto de vista médico. La segunda parte, más importante para este trabajo, se concentra en la descripción de los métodos computacionales para su simulación. Se presentan en total cinco metodologías: separación de nodos o *split*, método de retracción, método de eliminación de elementos, deformación plástica y un método de cambio topológico. Al final del capítulo se presenta una tabla comparativa de los cinco métodos para su consideración en el momento de decidir cuál de ellos usar. La mayoría de las metodologías detalladas en este capítulo se basan en las publicaciones actuales sobre métodos de corte, sin embargo, los algoritmos presentados (a menos que se indique) son propios de este trabajo, por lo que también son susceptibles a mejoras y optimizaciones.

En el Capítulo 5 se presentan algunos desarrollos de software, resultado del trabajo metodológico de los tres capítulos anteriores. Sobre estos desarrollos se prueban los diferentes métodos abordados, tanto para deformaciones, generación de geometrías y simulación de cortes. Al final del capítulo se describe un caso de estudio: el simulador de cirugía de próstata del CCADET de la UNAM; conjuntanto todos los casos de los capítulos anteriores. Esto es, se muestra el resultado de incorporar una técnica de deformación híbrida para deformaciones, una técnica de generación de mallas de volumen a partir de imágenes y la incorporación de simulación de cortes usando el método de eliminación de elementos. Todo este trabajo experimental, y en general de toda la obra, covergen en el Capítulo 6, donde se concluye sobre cada punto importante abordado y se hace una reseña de los posibles trabajos a futuro.

Dado que en la gran mayoría de las metodologías presentes en este trabajo requieren de conceptos físicos, matemáticos, computacionales y médicos, cada capítulo trata de

proporcionar una idea general sobre éstos. Sin embargo, hay conceptos y temas, que por su amplio contenido es mejor tratar por separado. Para ello, se presentan tres apéndices con las siguientes temáticas: Programación del SDK de *NVIDIA PhysX*, Modelo de elasticidad para objetos deformables y una breve reseña de los métodos de integración numérica. El primero trata de exponer los elementos básicos para programar el SDK de *PhysX* en la simulación de propiedades físicas en sistemas de partículas, objetos rígidos y objetos deformables, complementanto los capítulos 2 y 4. Es importante mencionar, que este apéndice no trata de ninguna manera ser un manual para la programación, dado que tan solo el programa de entrenamiento para el SDK que proporciona *nVidia* consta de 12 capítulos y 92 tutoriales, imposibles de abordar completamente en este trabajo, sin embargo, se proporcionan las referencias necesarias para su manejo. El segundo es una recopilación bibliográfica y acorde a la nomenclatura usada en toda la obra, que muestra el planteamiento de un modelo de elasticidad para objetos deformables desde un punto de vista de la mecánica de los medios continuos. El objetivo es demostrar el planteamiento de la ecuación de Navier como modelo de elasticidad, muy usado en los métodos de elemento finito (FEM) para simular deformaciones. En el tercer apéndice se presenta una breve reseña de los tres principales métodos de integración numérica, muy usados en la simulación física: método de Euler-Cauchy, método de Runge-Kutta grado 4 y el método de Verlet.

Finalmente podrá encontrar en las últimas páginas la lista completa de las referencias bibliográficas y electrónicas, que sirvieron como base para la escritura y el desarrollo de este trabajo, todas ellas seleccionadas cuidadosamente y de acuerdo con su impacto actual en el tema.

CAPÍTULO 2

Modelado Biomecánico y Computacional de Tejido Blando

En años recientes, la medicina, en su constante estado de evolución, ha incorporado a la ingeniería de la computación para el desarrollo de nuevas tecnologías de avanzada, que le permitan proveer nuevas formas de efectuar diagnósticos, obtener asistencia durante sus procedimientos y buscar alternativas de recuperación en pacientes que requieren tratamiento especializado. En el camino, se ha fusionado con otras áreas para formar otro campo de investigación de gran impacto actual, la Ingeniería Biomédica, la cual trata de aprovechar al máximo los conceptos de la física, la biología, las matemáticas, las ciencias de la computación, la robótica, la inteligencia artificial, entre otras, para la creación e incorporación de estas nuevas tecnologías.

En la ingeniería de la computación, los campos de mayor actividad y de mayor apoyo en la medicina son el análisis de imágenes, la graficación por computadora, la robótica, la realidad virtual, la inteligencia artificial y la simulación física. Todos éstos reciben como reto inmediato una necesidad de medición, cuantificación y evaluación de procesos y procedimientos que involucran en gran medida al cuerpo humano. Y es precisamente en este punto donde inicia nuestro estudio.

Como lo indica el título de este trabajo, el objeto de estudio es el modelado de cortes sobre una estructura que constituye gran parte del cuerpo: el tejido; más no cualquier tejido, sino específicamente el tejido blando. No tendría caso entonces entrar de lleno al modelado computacional de cortes sin antes conocer la estructura sobre la cual se operará dicha acción principal. Por ello, en este capítulo se describirá de manera general, el modelado biomecánico y computacional que actualmente domina la investigación en estas áreas (la medicina, la computación y la ingeniería biomédica), principalmente para el modelado de tejido como un cuerpo que se deforma al aplicársele una fuerza externa y que con mayor razón, es susceptible

a estirarse, desgarrarse, romperse o separarse. Para introducirnos comenzaremos conociendo esta estructura desde un punto de vista biológico, conoceremos la forma en que puede ser modelada físicamente y matemáticamente como un cuerpo elástico y finalmente modelada computacionalmente como un objeto deformable.

Sección 2.1

El Tejido

Un tejido es un conjunto de células, y matrices extracelulares, que presenta funciones especializadas. Una matriz extracelular está compuesta principalmente por fibras (por ejemplo, proteínas de colágeno y elastina) y por una capa de proteoglicanos. Las proteínas son macromoléculas orgánicas complejas, que contienen carbón, hidrógeno, oxígeno, nitrógeno y usualmente sulfuro; compuestas por una o más cadenas de aminoácidos. Las proteínas son el principal componente de todas las células vivas y por ende, del tejido (*Cowin, Doty, et. al., 2007 [18]*). En resumen, dice Lippert [39], dado que las células son muy pequeñas, una sola no representa mucho. En cada caso deben reunirse numerosas células con idénticas características para desempeñar una función. El conjunto de células del mismo tipo se denomina tejido.

En general, el tejido (animal) pude clasificarse en cuatro grupos [18] [39]:

- **Tejido conectivo:** Como su nombre lo indica, tiene la función de "conectar" y/o sostener órganos y otros tejidos; permitiendo mantener a los órganos en su lugar y garantizar su forma. Está compuesto por pocas células y una gran cantidad de sustancias intercelulares, en su mayoría fibras. Corresponden a este tipo: el cartílago, tendones, ligamentos, la matriz de huesos y el tejido adiposo, así como también la piel, la sangre y la linfa (sustancia que circula por los vasos linfáticos). Puede ser deformable (tejido conectivo, cartílago) o rígido (hueso, marfil, dentina). Al primero de ellos se le conoce como tejido conectivo elástico, ya que es capaz de distenderse y volver posteriormente a su posición original sin necesidad de imprimir sobre éste un nuevo esfuerzo.

- **Tejido epitelial:** También llamado tejido de revestimiento, se compone de formaciones planas de células que delimitan la superficie exterior (piel) e interior del cuerpo (mucosas), produciendo una superficie cerrada. Sus funciones principales son la comunicación con el exterior mediante intercambio de sustancias, así como la protección de los órganos y otros tejidos internos. Corresponden a este tipo: el recubrimiento celular de los pulmones, estómago, intestinos, vasos sanguíneos y la piel. Las células epiteliales también producen cabello y las uñas de manos y pies. En otros animales puden dar lugar a la generación de plumas y carcasas.

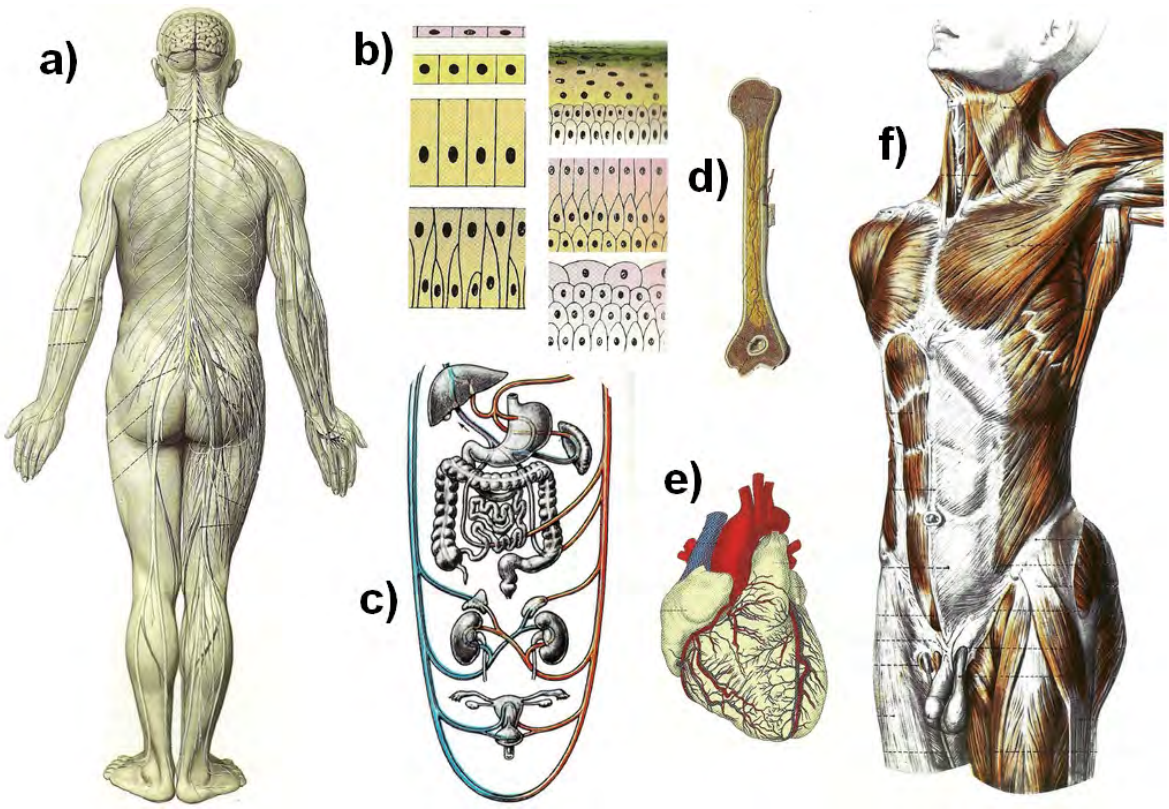


Figura 2.1. *Tipos de tejido presentes en la anatomía humana. a) Tejido nervioso presente sobre el sistema nervioso. b) Estructuras que forma el tejido epitelial de revestimiento. c) Órganos y vasos internos constituidos principalmente por tejido muscular liso. d) Hueso del húmero derecho, constituido por tejido conectivo y tejido muscular estriado. e) El corazón, formado por tejido cardiaco. f) Tejido muscular en los músculos externos. (Imágenes tomadas de: "Anatomía, estructura y morfología del cuerpo humano" de Lippert, 2003 [39].)*

- **Tejido muscular:** Se encarga de consumir el combustible del cuerpo (ejemplos, azúcares y grasas) para crear fuerzas por contracción y por tanto, el trabajo mecánico. Hay tres tipos de tejido muscular: esquelético, liso y cardiaco. Los músculos estriados cubren los huesos, los cuales crean una palanca para generar trabajo. El músculo liso, es aquél que forma una capa muscular de los órganos internos y alrededor de los vasos sanguíneos; se encuentra en: tracto digestivo (excepto faringe y la parte superior del esófago), vejiga, próstata, arterias y venas; activado por el sistema nervioso autónomo. Un fenómeno interesante ocurre en este tipo de tejido, y es la contracción involuntaria, dando lugar a los esfínteres. Finalmente, el músculo cardiaco se encarga de la acción de bombeo del corazón. El tejido cardiaco es el único que forma estructuras complejas que permiten la transmisión de impulsos eléctricos sobre el propio músculo para crear contracciones rítmicas.
- **Tejido nervioso:** Se compone de las células nerviosas, o neuronas; especializadas para transmitir información de una parte del cuerpo a otra. El cerebro y la espina dorsal son tejidos compuestos por redes neuronales.

Dada la anterior clasificación del tejido, nos enfocaremos en el estudio de lo que se denomina tejido blando, el cual se entiende como todo aquel tejido que por sus propiedades elásticas puede ser considerado deformable. De acuerdo con esto y las características anteriores, varios de los tipos de tejido pueden considerarse blandos, sin embargo, en este trabajo abordaremos únicamente dos: el tejido conectivo elástico (piel) y el tejido muscular liso, también llamado suave; que por sus propiedades físicas pueden ser modeladas tanto biomecánicamente, como computacionalmente.

Sección 2.2

Modelos Biomecánicos

Sin duda, uno de los tópicos más complicados de abordar cuando se trata de caracterizar una estructura, es el que se refiere al modelado, pues es muy difícil saber con exactitud la cantidad de variables involucradas; y en muchas ocasiones no se requiere de conocer toda la información, sino únicamente llevar a cabo una tarea de "simulación". Por ejemplo, supongamos un objeto hipotético, el cual está sometido a un conjunto de leyes o principios fundamentales, como de conservación de masa, conservación del momento lineal, conservación del momento angular y conservación de la energía, y se pretende simular sobre éste, comportamientos tales como: flujo sanguíneo, contracciones musculares, respiración, contracciones y relajación, distención, cortes o desprendimientos (en tejidos, por ejemplo), entre muchos otros. Desde el punto de vista físico y matemático, se requiere plantear un modelo que represente la transformación del objeto real a un espacio conceptual, generalmente Euleriano; y dada la gran cantidad de datos y variables involucrados muchas

veces se vuelve bastante complejo de entender. Asimismo, si el problema se plantea en términos de un modelo numérico que puede resolverse en una computadora, donde con ayuda de técnicas de procesamiento y despliegue de datos, un usuario podría visualizar los resultados del fenómeno, aún en este caso la cantidad de variables involucradas harían que terminemos hundidos en un mar de datos. Por ello, el planteamiento de un modelo tiene como primer objetivo dar un primer símil del objeto o fenómeno en estudio, de ahí el término "simular".

De tal modo que, podemos definir a un modelo como una representación idealizada de alguna o algunas de las propiedades de una estructura, fenómeno o sistema, que simulen su comportamiento y rasgos característicos, para definir de manera física y/o simbólica al conjunto de entes que tienen las mismas o semejantes características.

En la Biomecánica, ciencia encargada del estudio del movimiento de las estructuras vivas, se definen seis tipos de modelos. Estos son: partículas, objetos rígidos, objetos deformables, parámetros concentrados, estadísticos y autómatas celulares. En las siguientes secciones abordaremos los tres primeros, que son de interés para este trabajo [1], ya que como veremos más adelante, un tejido blando puede modelarse como un objeto deformable. Al mismo tiempo, no podemos entender un objeto deformable sin conocer antes cómo modelar partículas u objetos rígidos.

2.2.1. Modelo de Partículas

Desde el punto de vista de la mecánica clásica, el modelo de partículas es el más simple y toma como referencia a un objeto de masa como una partícula. La partícula tiene asociada una posición, localizada en el centro de masa del objeto. Así, la única operación de transformación que se efectúa sobre la partícula es la traslación, debida a su movimiento. La representación de estos objetos materiales se transportan a un espacio euclidiano como un modelo de punto (centro de masa). Este modelo fue creado por Isaac Newton (1642-1727) y usado para explicar la ley de gravitación universal, en la cual tomó al sol y los planetas como partículas. Asimismo, la segunda ley de Newton sirve de base para definir la transformación de posición que sufre un cuerpo al aplicársele una fuerza externa.

Así, sea un objeto con centro de masa en $\underline{x} = x_1\widehat{e_i} + x_2\widehat{e_j} + x_3\widehat{e_k}$, donde $\widehat{e_i}$, $\widehat{e_j}$ y $\widehat{e_k}$ son los vectores unitarios base en el sistema Cartesiano, y $x_i = x_i(t)$ $(i = 1, 2, 3)$ funciones del tiempo continuas y diferenciables, que se mueve en un espacio Euclidiano, tiene asociada una velocidad y aceleración lineales, dadas por:

$$\underline{v}(t) = \frac{dx_1}{dt}\widehat{e_i} + \frac{dx_2}{dt}\widehat{e_j} + \frac{dx_3}{dt}\widehat{e_k} \tag{2.1}$$

[1] Para una referencia más amplia de los modelos de parámetros concentrados, estadísticos y autómatas celulares, puede encontrarse mayor información en Cowin y Cotie [18], *Tissue Mechanics* (págs. 58-85)

$$\underline{a}(t) = \frac{d^2x_1}{dt^2}\widehat{e_i} + \frac{d^2x_2}{dt^2}\widehat{e_j} + \frac{d^2x_3}{dt^2}\widehat{e_k} \tag{2.2}$$

Si denotamos la masa total del objeto con m y la suma de fuerzas que actúan sobre él con $\underline{F}$, una expresión de la segunda Ley de Newton es:

$$\underline{F} = m\underline{a}(t) \tag{2.3}$$

Así, dada la masa y las fuerzas actuantes sobre un objeto, podemos calcular su aceleración lineal, velocidad lineal y posición en un tiempo t de la siguiente forma:

$$\underline{a}(t) = \frac{1}{m}\underline{F} \tag{2.4}$$

$$\underline{v}(t) = \int \underline{a}(t)dt \tag{2.5}$$

$$\underline{x}(t) = \int \underline{v}(t)dt \tag{2.6}$$

2.2.2. Modelo de objetos rígidos

Este modelo considera una característica importante de un objeto, su capacidad de movimiento rotacional, además del traslacional. Es decir, un objeto que se desplaza por el medio, sufre un giro asociado a su inercia; lo que implica que el objeto deja de ser un simple punto en el espacio, para convertirse en un conjunto de elementos de volumen con masa, que además no cambia de forma, de ahí el término de "rígido".

En el caso del movimiento traslacional del objeto, se cumple el mismo modelo de partículas; sin embargo, para el movimiento rotacional se hace necesario el uso de las ecuaciones de Euler (1707-1783) para el movimiento rotacional, las cuales corresponden a una forma proveniente de las expresiones usadas para definir una de las leyes constitutivas en el modelado físico matemático: la conservación del momento angular. Éstas se expresan en téminos de un sistema de coordenadas de referencia situado en el centro de masa del objeto rígido (u otro punto fijo de rotación), y coincidente con los ejes principales de inercia.

Así, sean I_{11}, I_{22} e I_{33}, los momentos principales de inercia, pertenecientes al tensor del momento de inercia $\underline{\underline{I}}$, simétrico:

$$\underline{\underline{I}} = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{12} & I_{22} & I_{23} \\ I_{13} & I_{23} & I_{33} \end{bmatrix}$$

donde cada componente está dada por:

$$I_{11} = \int_{\Omega(t)} (x_2^2 + x_3^2)\rho(\underline{x}, t)dv \qquad I_{22} = \int_{\Omega(t)} (x_1^2 + x_3^2)\rho(\underline{x}, t)dv$$

$$I_{33} = \int_{\Omega(t)} (x_2^2 + x_1^2)\rho(\underline{x}, t)dv \qquad I_{12} = -\int_{\Omega(t)} (x_1 x_2)\rho(\underline{x}, t)dv$$

$$I_{13} = -\int_{\Omega(t)} (x_1 x_3)\rho(\underline{x}, t)dv \qquad I_{23} = -\int_{\Omega(t)} (x_2 x_3)\rho(\underline{x}, t)dv$$

donde $\Omega(t)$ representa el dominio espacial que ocupa el objeto, $\underline{x}$ es un vector de posición de un elemento de volumen (dv) o de su masa con respecto al origen; $\rho(\underline{x}, t)$ es la densidad del elemento de masa, y x_1, x_2 y x_3, son las componentes de dicho vector de posición.

Entonces, las ecuaciones de Euler para el momento de fuerza, asociados a cada uno de los ejes de referencia, pueden escribirse como:

$$M_1 = I_{11}\frac{d\omega_1}{dt} + \omega_2\omega_3(I_{33} - I_{22}) \tag{2.7}$$

$$M_2 = I_{22}\frac{d\omega_2}{dt} + \omega_3\omega_1(I_{11} - I_{33}) \tag{2.8}$$

$$M_3 = I_{33}\frac{d\omega_3}{dt} + \omega_1\omega_2(I_{22} - I_{11}) \tag{2.9}$$

donde ω_1, ω_2 y ω_3, son las componentes de la velocidad angular $\underline{\omega}$, alrededor de su respectivo eje coordenado. Cuando hay una sola componente diferente de cero de la velocidad angular; esto es, $\omega_i \neq 0$ y $\forall\omega_j = 0(i \neq j, \{i, j\} = \{1, 2, 3\})$:

$$M_i = I_{ii}\alpha_i \tag{2.10}$$

donde α_i es la aceleración angular.

También, por conservación del momento angular $\underline{L}$, se cumple que:

$$\underline{\omega} = \underline{\underline{I}}^{-1}\underline{L} \tag{2.11}$$

donde $\underline{L} = \underline{r} \times m\underline{v}$, siendo $\underline{r}$ la distancia entre el centro de masa del objeto y una masa puntual m, que se desplaza con velocidad lineal $\underline{v}$.

2.2.3. Modelo de objetos deformables

Este modelo agrega la característica principal del movimiento relativo entre puntos regionalizados de un objeto, lo cual puede observarse en un cambio de la forma de éste. La forma se refiere a la estructura física y espacial (geométrica) de un objeto, compuesta de

materia (*Hess, et. al.[30]*). Por lo general, la forma se identifica por el sentido visual y el tacto; así, podemos identificar si un objeto está "formado" por órganos, huesos, músculos, piel; o bien, de manera general, por tejido. Adicionalmente, la forma geométrica define la frontera entre el material interno del objeto y su entorno.

Las deformaciones son ampliamente utilizadas en la simulación de materiales sólidos elásticos y fluidos, por lo tanto, los modelos deformables son la base para el modelado de estructuras biológicas como tejidos. En el caso del tejido blando, éste se comporta como un material visco-elástico, como se abordará más adelante.

Un objeto deformable tiene la característica de incluir tres tipos de transformaciones: traslación, rotación y deformación. En la siguiente sección abordaremos con más detalle este tipo de objetos, que son de gran importancia para este trabajo, al poder considerar el tejido como un elemento elástico y que por tanto, se encuentra bajo la influencia de leyes físicas.

Así, en la Figura 2.2 se muestra un resumen de los diferentes modelos biomecánicos abordados hasta el momento.

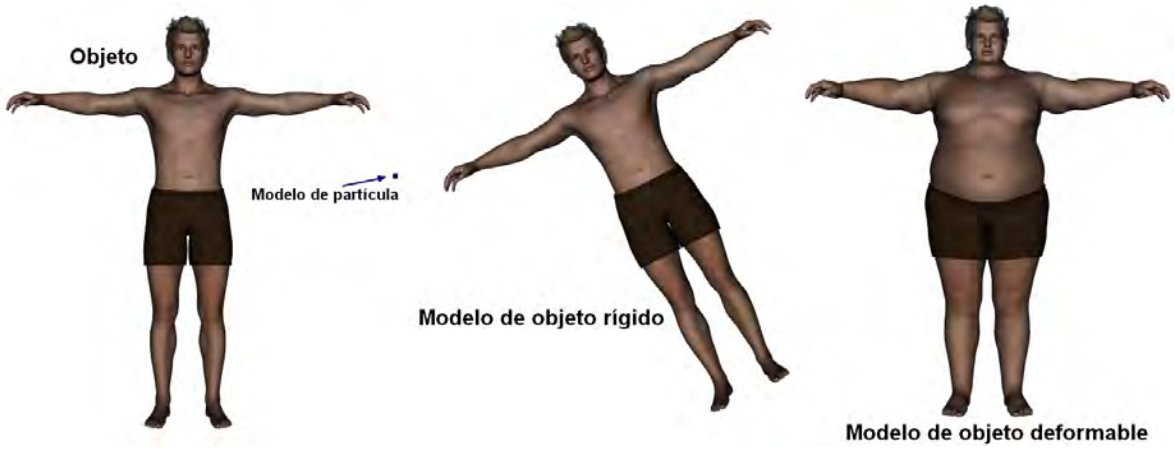


Figura 2.2. *Modelos Biomecánicos. El modelo de partículas representa al objeto como un punto, situado en el centro de masa. El modelo de objetos rígidos considera la geometría del objeto completo, cuyas transformaciones son traslación y rotación. El modelo de objetos deformables, además de considerar las transformaciones básicas, considera un cambio de forma.*

Sección 2.3

Modelado Computacional de objetos deformables

Desde el punto de vista computacional, la tarea de proponer un modelo para objetos deformables, tiene el objetivo de generar una metodología capaz de resolverse en una computadora en un tiempo que depende de la aplicación. Los sistemas que requieren la interacción directa con el usuario en forma sincronizada, generalmente se les denomina *sistemas en tiempo real*; esto es, el sistema debe ser capaz de garantizar el cálculo de parámetros que involucra un modelo (generalmente numérico o de visualización) y la fluida participación del usuario, al mismo tiempo que hay una retroalimentación de información entre el usuario y la máquina. Por ejemplo, si hablamos de un ambiente virtual en tiempo real, uno de los elementos críticos es la visualización, que debe llevarse a cabo en al menos 25 cuadros por segundo. Nealen et. al. [54] aseveran que para tal reto, la ingeniería de la computación hace uso de la dinámica Newtoniana, la mecánica del medio continuo, el cómputo numérico, la geometría diferencial, el cálculo vectorial, la teoría de la simulación y el cómputo gráfico.

Actualmente, las técnicas del cómputo gráfico, han permitido la representación visual de objetos tridimensionales, tanto rígidos como deformables. Inicialmente, éstas se basaban en la modificación de la geometría usando modelos matemáticos, como las superficies *spline*; recientemente, se han desarrollado un conjunto de métodos computacionales y modelos matemáticos basados en la física de los materiales, cada uno de ellos con ventajas y desventajas, con el fin de aplicarse en la simulación de objetos dinámicos, que incluye el cambio que sufren en el tiempo debido a factores externos.

Los modelos deformables desde el punto de vista computacional, pueden definirse dependiendo del espacio en el cual estén involucrados. Así, en una dimensión encontramos a la familia de las curvas, en dos dimensiones a las superficies y en tres dimensiones a los sólidos. En cada espacio podemos definir también elementos básicos: líneas (una dimensión), triángulos (dos dimensiones) o tetraedros (tres dimensiones). En el capítulo siguiente abordaremos más a detalle estos elementos.

Podemos encontrar la aplicación de estos modelos en tres campos de investigación (*Meier, 2004 [43]*):

1. Animación por computadora [15][16][31]: En esta área se emplean para generar modelos deformados a partir de un modelo base (estático), al cual se le aplica un modificador predefinido. En estas aplicaciones no es importante la representación en tiempo real, dado que el objetivo es obtener un objeto igualmente estático pero deformado en un instante de tiempo.

2. Segmentación de imágenes [17]: En este campo se emplean para obtener regiones

en imágenes que siguen patrones en común. En el caso de imágenes médicas, encontramos aplicaciones en la localización de órganos, partes del cuerpo, tejidos, tumores, entre otros. En el caso de la reconstrucción de modelos tridimensionales, una buena segmentación tiene una relevancia muy importante.

3. Simulación física interactiva [43][62]: En este campo se busca emular el comportamiento conocido de ciertos materiales, principalmente elásticos, como el tejido. Así, se pueden encontrar aplicaciones como la planeación quirúrgica, cirugía asistida en tiempo real y sistemas de entrenamiento.

En la medicina, como en otros campos, los sistemas de realidad virtual están siendo empleados con mayor frecuencia. Esto principalmente por el potencial beneficio que parecen introducir, desde un punto de vista psicológico, tecnológico y económico. El primero se ve beneficiado, por ejemplo, en la adquisición de habilidades médicas de un residente usando un sistema de entrenamiento; el segundo, en las implicaciones de desarrollo y generación de nuevos sistemas de diagnóstico automatizados; y el tercero, en la disminuación de costos asociado con el uso de instrumental médico.

Uno de los grandes retos de los sistemas de realidad virtual aplicados en medicina, es el nivel de realismo alcanzado, para el cual influyen muchos aspectos y que dependen totalmente de la percepción visual del usuario, quien evalúa y eventualmente decide si el sistema es réplica de la realidad o simplemente una mera aproximación. Es precisamente en este punto donde los modelos computacionales entran a escena, pues en sistemas que involucran la interacción con tejidos, éstos contribuyen a adicionar un mayor realismo. El principal obstáculo: el tiempo de procesamiento.

Sección 2.4

Métodos de solución al problema de los objetos deformables

Desde su incorporación al campo de investigación (*Terzopoulos, 1987 [77]*), se han desarrollado modelos computacionales para representar a los objetos deformables; divididos en tres grupos: métodos heurísticos, métodos basados en la mecánica del medio continuo y métodos híbridos [43]. El primero toma como base la eficiencia y facilidad de resolución, desde un punto de vista matemático y computacional, en comparación con otros métodos. El segundo tiene su base en la teoría de la mecánica del medio continuo, en la cual se toma en cuenta la dinámica del objeto desde una perspectiva más formal, obteniéndose aproximaciones más realistas; sin embargo, aún hoy en día son muy costosos computacionalmente hablando. Finalmente, el tercer grupo engloba a aquellos métodos que mezclan los dos primeros enfoques, y que de manera separada, procesan una parte del

problema de la deformación. La Figura 2.3 muestra un diagrama de la clasificación de los principales enfoques aplicados en la simulación computacional de objetos deformables.

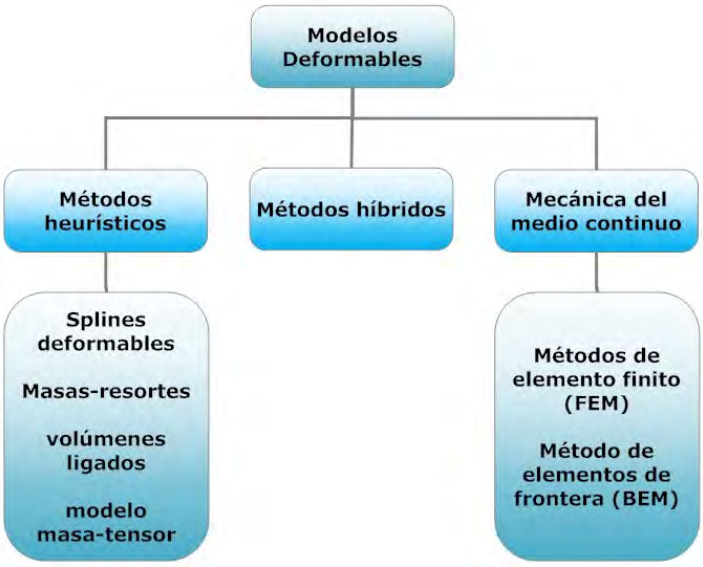


Figura 2.3. *Modelos deformables para simulación de cirugía (Meier, 2004 [43]).*

2.4.1. Métodos heurísticos

Como su nombre lo indica, estos métodos son aproximaciones "creadas" a partir del análisis geométrico de un objeto y cuya deformación depende de la relación espacial o funcional que guardan cada uno de sus elementos. En este grupo encontramos a los *splines* deformables, los modelos masa-resorte, el modelo de volúmenes ligados y el modelo masa-tensor.

Splines deformables

El principio básico del método, es la definición de puntos llamados *de control*, los cuales al ser manipulados de acuerdo a un criterio de desplazamiento y a una función de energía potencial (proporcional al grado de deformación elástica), readaptan la geometría de un objeto, suavizándolo (Figura 2.4). El objeto geométrico puede ser una curva, una superficie o un volumen. Los *splines* deformables también son conocidos como *contornos activos* y fueron los primeros adoptados en el campo de la simulación quirúrgica.

Las principales desventajas de los métodos basados en *splines*, están en la arbitrariedad de sus parámetros y la dificultad de determinarlos empíricamente. Además, los algoritmos de *rendering* gráfico actuales no emplean como estándar la representación geométrica

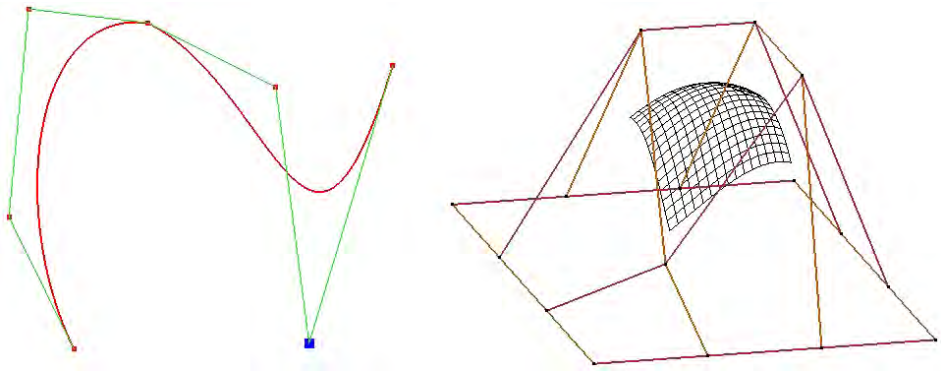


Figura 2.4. *Curva y superficie deformables mediante splines. A la izquierda, curva spline con OpenGL [60]. A la derecha, superficie NURBS con Blender [10].*

de curvas suavizadas, sino que emplean representaciones poligonales, lo cual eleva el costo computacional.

Modelo de masas y resortes

Este método introduce, a diferencia de los modelos basados en funciones *spline*, una mayor aproximación al problema real. Se definen dos espacios principales, el geométrico y el físico. En el espacio geométrico encontramos un objeto discretizado en elementos puntuales, llamados *vértices* y elementos superficiales, llamados *caras*. En el espacio físico, cada vértice en el espacio geométrico representa un elemento de masa y a su vez cada línea que los une (arista), representa un resorte con propiedades elásticas.

El modelo dinámico que describe el equilibrio de cada elemento i de masa en una malla deformable, está dado por la siguiente ecuación:

$$m_i \frac{d^2 \underline{x}_i}{dt^2} + \gamma_i \frac{d\underline{x}_i}{dt} + \underline{f}^i_{int}(t, \underline{x}_i) = \underline{f}^i_{ext}(t, \underline{x}_i) \tag{2.12}$$

donde m_i representa la masa del nodo i, de posición $\underline{x}_i$ que depende del tiempo t; γ_i es el coeficiente de fricción viscosa de los resortes adyacentes; y, $\underline{f}^i_{int}$ y $\underline{f}^i_{ext}$ son las intensidades correspondientes a las fuerzas internas y externas que actúan sobre el nodo i. En el Apéndice B podrá encontrar una definición completa de las fuerzas internas, externas y de frontera.

Las fuerzas internas, opuestas a las fuerzas externas que provocan la deformación, están dadas por el grado de deformación de los resortes; esto es,

$$\underline{f}^i_{int}(t, \underline{x}_i) = \sum_{j \in N_i} K^i_j \frac{\|\underline{x}_j - \underline{x}_i\| - \|\underline{x}_j - \underline{x}_i\|^0}{\|\underline{x}_j - \underline{x}_i\|} (\underline{x}_j - \underline{x}_i) \tag{2.13}$$

donde N_i es el conjunto de resortes adyacentes al nodo i, que lo unen con cada nodo j; K_j^i es la rigidez del resorte que une el nodo i con el nodo j; y, $\|\underline{x}_j - \underline{x}_i\|$ y $\|\underline{x}_j - \underline{x}_i\|^0$ son las longitudes final e inicial de los resortes.

Este modelo suele resolverse usando métodos numéricos de integración, como el de diferencias finitas de Euler sobre un intervalo Δt de tiempo. Otros métodos más sofisticados de resolución, como el de *Runge-Kutta* permiten usar intervalos de tiempo más grandes para alcanzar la convergencia; sin embargo, esto representa también un incremento del doble de procesamiento con respecto al método básico de Euler. El Apéndice C aborda éstos y otros métodos de integración muy útiles en la simulación computacional de objetos deformables.

En cuanto al método de deformación, el modelo de masas y resortes ofrece una ventaja sustancial para objetos dinámicos que requieren visualizarse en tiempo real. Y como puede observarse, la desventaja principal radica en que es totalmente local, y la deformación únicamente depende de los nodos adyacentes, lo que limita el realismo visual.

En la simulación de tejido, Teodoro[76] y Padilla[63] presentan la implementación de este modelo para superficies y volúmenes, respectivamente, en la simulación de comportamiento elasto-dinámico de una próstata virtual, como se muestra en la Figura 2.5.

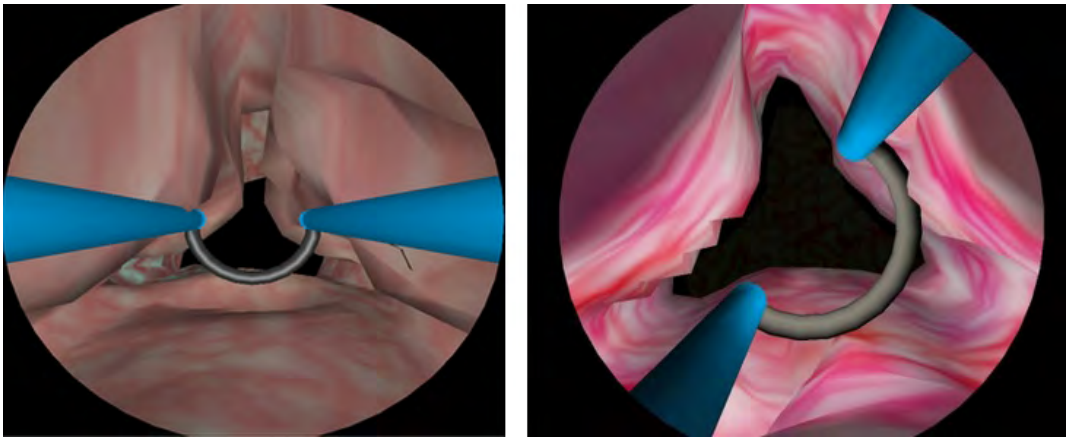


Figura 2.5. *Deformaciones de una próstata virtual usando el método de masa-resorte. A la izquierda, deformación sobre un modelo de superficie triangular. A la derecha, deformación sobre un modelo de volumen de tetraedros.*

Volúmenes ligados

El método consiste en la discretización de un objeto en pequeños elementos de volumen, como cubos, los cuales se encuentran interconectados bajo un criterio de vecindad. La diferencia con el método de masas y resortes es el notable incremento de elementos físicos involucrados y el método de solución. En cuanto a la visualización, se emplean superficies

que únicamente nos permiten visualizar la frontera de los elementos de volumen, conocidas como *surface maps* o mapas de superficie (Figura 2.6).

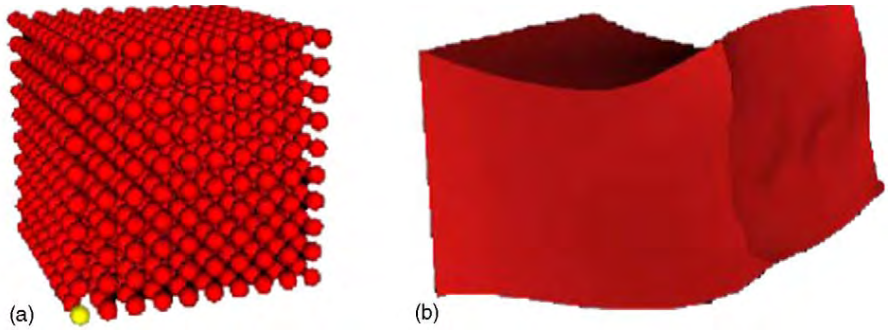


Figura 2.6. *Modelo simple de volúmenes ligados. (a) Sistema sin deformar. (b) Resultado de la deformación usando surface mapping (Tomado de Meier [43], 2004).*

La principal ventaja que presenta este método es en la simulación de cortes, el cual sencillamente consiste en eliminar elementos del dominio; o bien, otras técnicas como erosión o sutura de tejido.

Uno de los algoritmos empleados para solucionar este modelo, es el de *Chain mail* (*Gibson, 1997 [26]*), el cual emplea la discretización para construir una malla de eslabones, donde cada elemento forma el eslabón unido a sus vecinos. La Figura 2.7 muestra el principio básico de este algoritmo. Su simplicidad es comparable con el de masas y resortes; sin embargo, éste tiene la ventaja de propagar la deformación sobre un volumen constante, con lo cual se pueden simular comportamientos cuasi-viscosos de materiales plásticos.

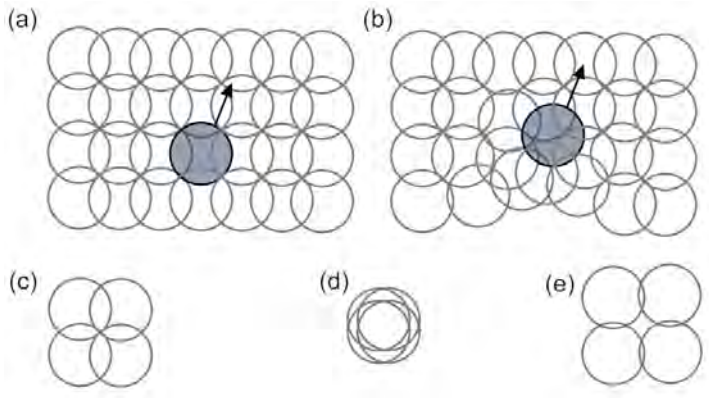


Figura 2.7. *Principio básico del algoritmo Chain mail. Deformación de una malla 2D al moverse un eslabón en la dirección de la flecha. (a) Estado inicial. (b) Estado deformado: los vecinos se desplazan para mantener las distancias mínima y máxima. Conjunto de enlaces: (c) estado inicial, (d) compresión máxima y (e) extensión máxima. (Tomado de Meier, 2004 [43]).*

Para proporcionar el comportamiento elástico, los resortes del algoritmo de *chain mail* emplean una función de energía acumulativa, que se relaja conforme su energía interna. Esta energía interna es proporcional con respecto al desplazamiento de cada elemento. De esta forma, no existe una fuerza asociada a los enlaces; por tanto, la retroalimentación de fuerzas se asume proporcional a la profundidad de la penetración. La profundidad se determina con la ayuda de un mapa de distancias, donde se tabulan las distancias iniciales de todos los elementos con respecto a la superficie.

Modelo masa-tensor

Este modelo es otra extensión del modelo básico de masas-resortes y se basa en la discretización de todo el interior del objeto deformable en tetraedros. En su forma más sencilla, los puntos de masa y los resortes son remplazados con respecto a los vértices de los tetraedros y sus aristas. La discretización se vuelve un poco más complicada cuando se trata de aplicaciones en tiempo real, que requieren el cambio topológico del objeto, lo cual implica un alto costo computacional. Como en la mayoría de los métodos heurísticos, el problema de la arbitrariedad persiste durante la selección de parámetros para asegurar la estabilidad del modelo.

El modelo se acerca a uno de los métodos que emplea la mecánica de los medios continuos: elementos finitos. El comportamiento deformable de cada tetraedro se resume en tensores. La rigidez de cada uno de estos elementos sustituye los resortes, por lo cual la fuerza interna $\underline{f}^i_{int}$, actúa en los diferentes nodos y se opone a las fuerzas que provocan la deformación, causadas principalmente por fuerzas externas y fuerzas conocidas. La relativa ventaja del método es la rápida propagación de la deformación. Sin embargo, esta última se ve afectada por el modelo matemático a resolver, que en muchos casos solo permite deformaciones pequeñas para alcanzar un realismo físico.

Empleando este método, la única forma de conseguir simular cortes en tiempo real, es la eliminación bruta de tetraedros de la malla como sucede en la electrocauterización [43], que como se verá más adelante requiere de una malla de alta resolución.

2.4.2. Métodos de la Mecánica de los Medios Continuos

Un enfoque más realista de los objetos deformables, es aproximar el comportamiento deformable mediante la mecánica de los medios continuos. De esta forma, sobre un material actúa un conjunto de fuerzas, tanto internas como externas, para definir un modelo continuo de elasticidad lineal. En general, podemos decir que el modelo básico de desplazamiento debido a un comportamiento elástico, está dado por la ecuación de Navier:

$$\mu\nabla^2\underline{u} + (\lambda + \mu)\nabla(\nabla \cdot \underline{u}) + \underline{F} = \underline{0}$$

donde λ y μ son las constantes de *Lamè* del material, $\underline{u}$ es el vector de desplazamiento de cualquier punto discretizado del objeto con respecto a su posición inicial y $\underline{F}$ un vector de fuerzas externas.

Para entender mejor el significado de este modelo, se recurre a la teoría básica de la elasticidad (Ver Apéndice B). Actualmente, no existe una solución analítica a este problema y los principales esquemas adoptados son el de elementos finitos y de elementos de frontera.

Método de Elementos Finitos

El método de elementos finitos o de elemento finito (FEM, *Finite Element Method*) es uno de los más populares en las ciencias de la computación para la resolución de ecuaciones diferenciales parciales (Nealen et. al. [54]) que rigen el comportamiento elástico de mallas irregulares. No se conoce un origen exacto del método, sin embargo, fue desde los años 50's que el método comenzó a desarrollarse debido a la investigación en el campo del análisis estructural de los medios continuos (Sadd [67], 2009).

De manera general[2], el método se divide en cinco pasos:

1. Discretización del dominio. El dominio físico que compone el objeto se divide en un conjunto de formas básicas llamadas "elementos finitos", que llenan el volumen del cuerpo. La resolución de la subdivisión depende del tamaño del objeto y de su forma. Los elementos básicos son: el segmento lineal en una dimensión, el triángulo en dos y el tetraedro en tres dimensiones.

2. Aproximación de la solución en términos de valores nodales. Se obtiene una primera aproximación de la solución usando una función de interpolación, lineal o cuadrática.

$$\underline{x}(\underline{p}, t) = \sum_{l=1}^{n} B_l^e \underline{x}_l^e(t) \tag{2.14}$$

donde $\underline{x}(\underline{p}, t)$ es el campo de deformación del cuerpo en función del punto material de coordenadas $\underline{p}$ y tiempo t. B_l^e; $l = 1...n$, es el conjunto de funciones de forma o funciones de interpolación, n es el número de nodos del elemento, y $\underline{x}_l^e(t)$; $l = 1...n$, son las coordenadas de los nodos del elemento. En el caso de dos dimensiones (triángulos), la Ecuación (2.14) sería:

$$\underline{x}(\underline{p}, t) = \sum_{l=1}^{3} B_l^e \underline{x}_l^e(t) \tag{2.15}$$

[2]Dado que no es el objetivo de este trabajo la descripción completa del método, se recomienda consultar el capítulo 16 del libro *Elasticity* de Sadd[67], el capítulo 6 del libro *An Introduction to Computer Simulation* de Woolfson[86]; así como los artículos [43] y [54], para una explicación más a detalle.

Para tres dimensiones (tetraedros):

$$\underline{x}(\underline{p}, t) = \sum_{l=1}^{4} B_l^e \underline{x}_l^e(t) \tag{2.16}$$

3. Desarrollo del sistema de ecuaciones. Dado el conjunto de posiciones de los vértices de un elemento y sus funciones de aproximación, el campo de deformación continua $\mathbf{u}(\underline{p})$ está definido por la Ecuación (2.14). La energía de deformación de un elemento está dada por:

$$E = \int_V \varepsilon(\underline{p}) \cdot \sigma(\underline{p}) d\underline{p} \tag{2.17}$$

donde el punto representa el producto escalar entre dos tensores, el tensor de deformación ε y el tensor de esfuerzos σ (Ver Apéndice B). Entonces, las fuerzas se calculan como las derivadas de la energía con respecto a las posiciones nodales.

$$\mathbf{f} = \frac{\partial E}{\partial \mathbf{p}} \tag{2.18}$$

Al linealizar el sistema, la relación entre un elemento e que conecta n_e nodos del elemento se puede expresar como:

$$\mathbf{f}_e = \mathbf{K}_e \mathbf{u}_e \tag{2.19}$$

donde $\mathbf{f}_e$ está compuesto por las fuerzas nodales y $\mathbf{u}_e$ de los desplazamiento nodales de cada elemento. La matriz $\mathbf{K}_e$ se denomina matriz de rigidez (*stiffness matrix*) del elemento.

Para toda la malla, la matriz de rigidez está dada por la suma de cada una de las matrices de rigidez de cada elemento; esto es:

$$\mathbf{K} = \sum_e \mathbf{K}_e \tag{2.20}$$

De esta forma, la ecuación del movimiento para una malla completa está dada por:

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}_{ext} \tag{2.21}$$

donde $\mathbf{M}$ es la matriz de masas, $\mathbf{D}$ la matriz de amortiguamiento y $\mathbf{f}_{ext}$ las fuerzas externas aplicadas.

4. Resolución el sistema de ecuaciones.

5. Cálculo de las cantidades de interés.

Método de elementos de frontera

Este método es una variante del método de elementos finitos, cuya principal característica es tomar únicamente la frontera del objeto como base para el cálculo de la deformación. De esta manera, la forma integral de la ecuación del movimiento se reduce a una integral de superficie aplicando el teorema de *Green-Gauss* (Nealen et. al. [54]). Así, el problema de tres dimensiones usando tetraedros, se reduce a un problema de dos dimensiones. Sin embargo las desventajas son: que sólo funciona para materiales homogéneos y además se vuelve más complicada la simulación de cambios topológicos cuando se intentan modelar fracturas.

2.4.3. Métodos Híbridos

Son una combinación entre alguno o algunos de los métodos heurísticos, y alguno o algunos de los métodos basados en la mecánica de los medios continuos. En general, emplean la abstracción del método de masas-resortes y el método masa-tensor, para adaptarlos en un enfoque de Elemento Finito, o de manera inversa, logrando optimizar características como velocidad de cálculos, robustez en la deformación y la readaptación de la malla ante cambios topológicos.

2.4.4. Comparación

En 2004, Meier et. al. [43] presentaron un estudio de los diferentes métodos abordados en los apartados anteriores. De acuerdo a sus investigaciones, todos los métodos evaluados son utilizables para aplicaciones de simulación quirúrgica, y poseen características de rapidez, robustez, realismo psicológico y flexibilidad topológica. En este último, uno de los aspectos principales es la funcionalidad del método al agregarle al sistema otras funciones, como por ejemplo, cortes en el modelo. En su artículo original, concluyen que uno de los mejores métodos en términos de cómputo, es el método de masas y resortes; sin embargo, en términos de realismo, el método de elementos finitos es mucho mejor pero más costoso. Ante este problema, una de las alternativas apunta al uso del método de elementos de frontera (BEM), que en la evaluación del estudio mostró un desempeño muy similar al de masas y resortes (Tabla 2.1 y Figura 2.8).

Característica	Peso	Splines deformables	Modelo Masas y resortes	Volúmenes ligados	Modelo masa-tensor	Modelos híbridos	FEM	BEM
Cómputo								
Velocidad	7	3	6	0	4	5	7	7
Robustez	5	3	2	4	4	4	5	5
Pre-cálculos	2	2	2	2	1	1	0	0
Dimensión del sistema de ecuaciones	1	0	1	0	0.5	0.5	1	1
Total	15	8	11	6	9.5	10.5	13	13
Topología								
Flexibilidad (incisiones)	6	3	5	6	3	2	0	4
Rendering	5	3	5	2	5	5	5	5
Dimensión de la malla	2	1	2	0	1	1	1	2
Adquisición de los datos de la malla	2	1	2	1.5	0.5	0.5	0.5	2
Total	15	8	14	9.5	9.5	8.5	6.5	13
Realismo biomecánico								
Comportamiento deformable	4	1.5	3	3	3	3	3	4
Consistencia volumétrica	4	3	4	4	4	3	0	0
Grandes deformaciones	3	0	0	3	4	3	2	2
Dinámica	2	1	2	2	2	1	0	1
Determinación de parámetros	2	0	0	0	1	1	2	1
Total	15	5.5	9	12	14	11	7	8
Total	45	21.5	34	27.5	33	30	26.5	34

Tabla 2.1. Comparación entre modelos deformables aplicados en simuladores de cirugía (*Meier et. al., 2004[43]*). El estudio enlista una serie de características necesarias en una aplicación en tiempo real de acuerdo a tres criterios principales: procesamiento, topología geométrica y realismo biomecánico. Cada criterio suma 15 puntos totales de importancia. Los puntos se distribuyen en pesos máximos, de acuerdo con la importancia de cierta característica. Así, cada método se evalúa en una escala que va del 0 al valor del peso máximo, sumando al final un total que representa el valor diferencial entre uno u otro método.

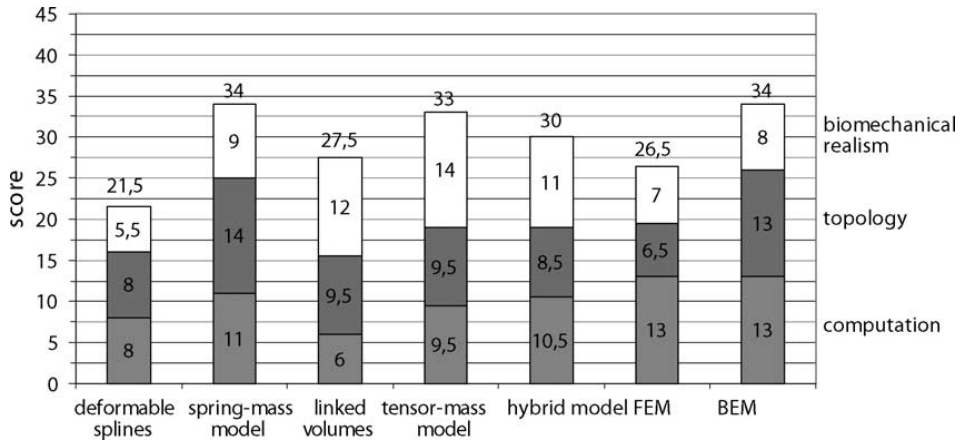


Figura 2.8. *Comparación gráfica entre modelos deformables (Meier, 2004 [43]).*

Sección 2.5

Paquetes de desarrollo para la simulación de objetos deformables

En los últimos años empresas, universidades y organizaciones sin fines de lucro se han dado a la tarea de crear paquetes de desarrollo, basado en software y hardware para dar soporte a la simulación basada en física. Estos paquetes se enfocan principalmente en la simulación de partículas, objetos rígidos y objetos deformables. Se pueden encontrar entonces en el mercado y en la nube del software libre, paquetes muy avanzados para la simulación realista de fluidos, simulación de impactos, simulación de estructuras biomecánicas, simulación de órganos, entre otros, con aplicaciones en animación por computadora, videojuegos, juegos serios y sistemas de entrenamiento virtual.

La tarea clave de muchos de los paquetes de simulación es alcanzar el realismo suficiente para imitar el comportamiento de cierta estructura, tomando en cuenta la aplicación. Así por ejemplo, de acuerdo con Narayanasamy [52] y Johnson [35], podemos distinguir un juego, un juego serio y un simulador para entrenamiento, por el simple propósito que involucran, pues mientras el primero tiene sólo la tarea de entretener, el segundo y el tercero requieren de un propósito académico o de investigación. En este enfoque, un simulador para entrenamiento tiene necesariamente la difícil tarea de proponer un entorno más cercano a la realidad.

Entre los paquetes para simulación física realistas podemos encontrar: NVIDIA PhysX [64], SOFA [71], Unreal Engine [82], OpenTissue [61], Havok [32], jMonkey Engine [36], Newton Game Dynamics [55], entre otros. Específicamente para la simulación de objetos deformables, los paquetes más estables, en constante desarrollo y más empleados

son NVIDIA PhysX, SOFA y Unreal Engine. Unreal Engine emplea PhysX con algunas mejoras y está más orientado al diseño de videojuegos. A continuación se presentan algunas características de estos tres:

- **NVIDIA PhysX**: Es un paquete de desarrollo para la simulación dinámica de objetos, así como la detección de colisiones con un alto grado de realismo. Fue creado originalmente por la empresa de semiconductores AGEIA, quien inventó el chip *PhysX*, una unidad de procesamiento con capacidades para la simulación basada en física desde hardware. Tras la compra de esta empresa por parte de NVIDIA en 2008, el motor pasó a llamarse *NVIDIA PhysX*. Se compone de un kit de desarrollo (SDK) optimizado para CUDA [19] (Arquitectura de Dispositivos para Cómputo en Paralelo) y una PPU (Unidad de Procesamientos de Física) [3], que en los últmios años NVIDIA ha incorporado en sus nuevas tarjetas gráficas. Las capacidades del entorno *PhysX* son: simulación de cuerpos rígidos, cuerpos blandos, telas, articulaciones, fluidos, resortes y simulación de personajes. También posee capacidad para detección de colisiones y algoritmos de respuesta ante éstas. Su gran diferencia con otros motores es que pone énfasis en la simulación, más que en la visualización y *rendering*, por lo que éstas deben ser cubiertas por parte del desarrollador (*Maciel et. al., 2009 [41]*). Generalmente tiene un amplio soporte con *OpenGL* y *Microsoft DirectX*.

- **SOFA** [2]: Es un software de desarrollo enfocado a la simulación en tiempo real de aplicaciones en medicina. Su caracteristica de código abierto le han ganado confianza entre la comunidad científica para usarse como plataforma en el desarrollo de nuevos algoritmos de simulación. Sus principales características son:

  - Debido a su implementación por software y por ser de código abierto, permite la reutilización de algoritmos y la mejora de ellos gracias al acceso a su código fuente (característica que no poseen otros motores, por ejemplo, *Nvidia PhysX*).
  - Edición de propiedades de simulación vía XML (*Extensible Markup Language*), tales como: comportamiento deformable, propiedades de visualización, método de resolución, restricciones, algoritmo de colisiones, etc.
  - Basado en gráficos de escena (*scene-graphs*).
  - Eficiencia para la simulación de interacción entre objetos.
  - Simulación de objetos deformables usando los métodos de masas y resortes, Elemento Finito Lineal y co-rotacional.
  - Simulación de objetos rígidos y fluidos.
  - Modelos de colisión: Esferas, mallas triangulares, campos de distancias.
  - Esquemas de integración: Euler y Runge Kutta.

[3] Por esta razón, algunas veces se le clasifica como un "middleware", mitad software, mitad hardware.

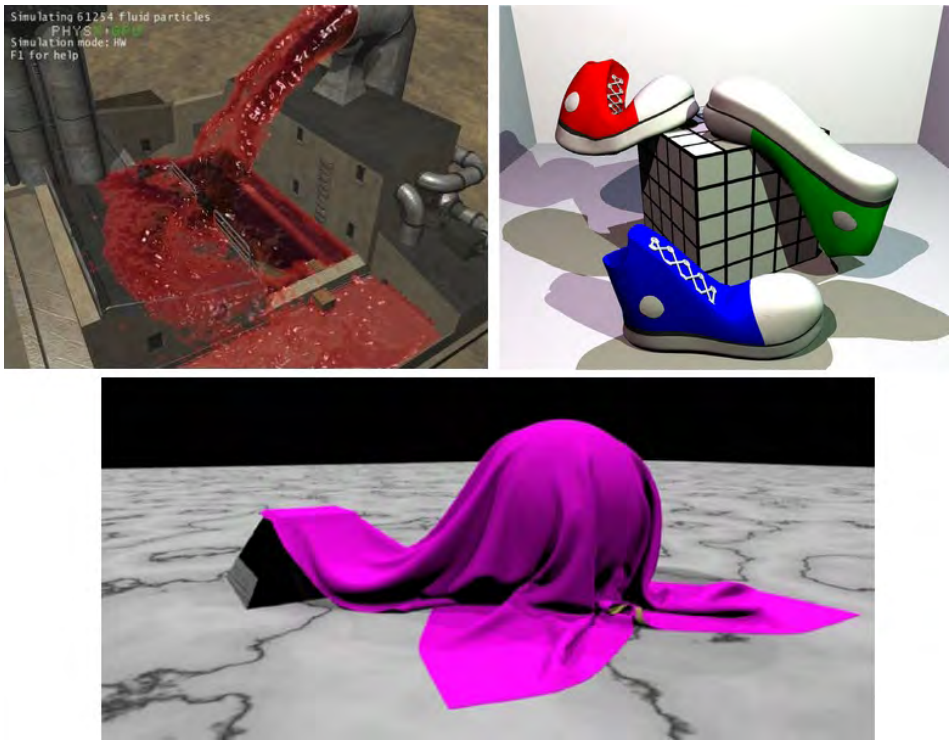


Figura 2.9. *Simulación de fluidos, objetos deformables y tela con NVIDIA PhysX [64].*

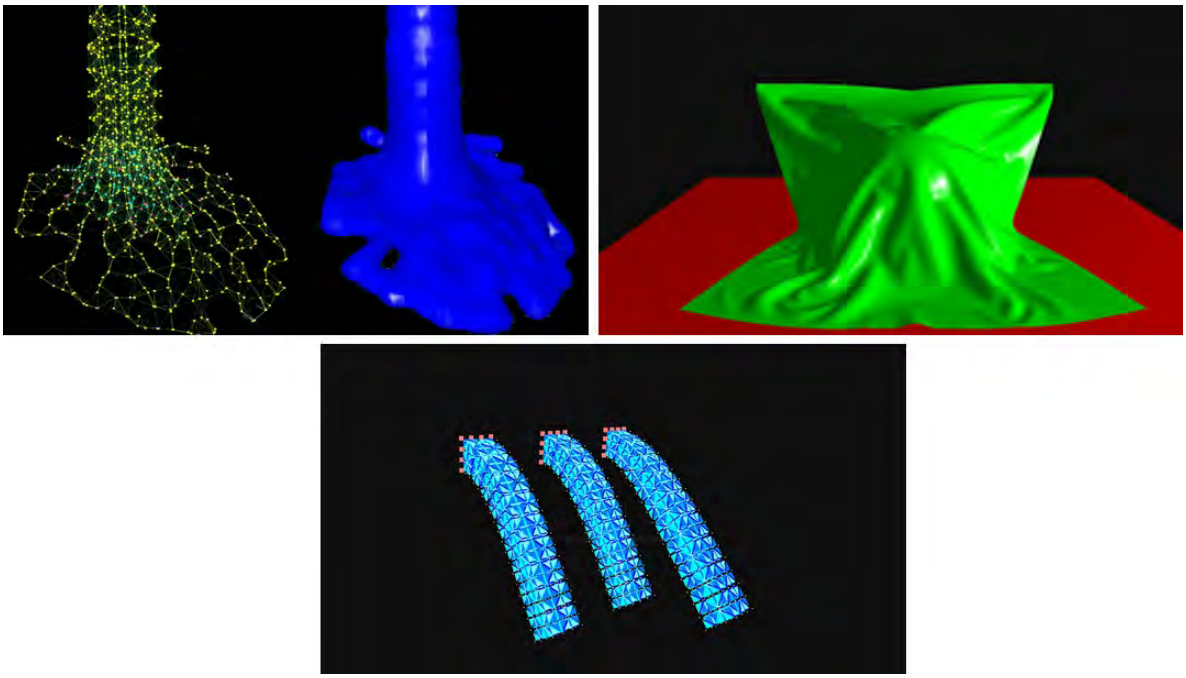


Figura 2.10. *Simulación de fluidos, tela y objetos deformables con SOFA [71].*

- **Unreal Engine**: Es un motor de desarrollo para la creación de contenidos de entretenimiento, muy usado en la industria de los videojuegos, pero fácilmente transportable a otros campos. Está construido y diseñado para hacerlo portable en una gran variedad de plataformas de hardware, desde dispositivos móviles hasta consolas de videojuegos. Su gran desempeño se lo debe a la integración de varias tecnologías de tipo "middleware". Sus principales características son:
  - Animación basada en esqueletos y movimientos usando cinemática inversa, así como *Morphing*.
  - Módulos de inteligencia artificial para control de personajes.
  - Sistema para control de audio.
  - Sistema de efectos de partículas.
  - Sistema para creación de contenido fílmico.
  - Editor de escenarios y personajes.
  - Sistema de visualización y *render*.
  - Simulación de objetos rígidos, cuerpos deformables elásticos y deformación de personajes.
  - *Shaders* (efectos de iluminación) en tiempo real.

Figura 2.11. *Simulación de fluidos, deformación y Morphing con Unreal [82].*

Como se mencionó con anterioridad, en este trabajo el objetivo es simular, además de cortes, tejido blando para aplicarse en simuladores de entrenamiento médico. Por ello, para aprovechar alguno de los paquetes ya disponibles para simular objetos deformables, se decidió usar *NVIDIA PhysX*. En las siguientes secciones, abordaremos un caso de estudio aplicado en la simulación de tejido para un simulador de laparoscopía gástrica virtual, desarrollado por la Universidad Federal de Río Grande del Sur de Brasil y el Instituto Politécnico de Rensselaer en Estados Unidos. Este caso de estudio nos ayudará a comprender un modelo genérico de simuladores de cirugía para entrenamiento médico, además de acercarnos a otro objetivo de la tesis: el simulador de entrenamiento para cirugía de próstata, en desarrollo por el Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET) de la UNAM.

La simulación de tejido blando usando *NVIDIA PhysX* puede encontrarse en la sección de experimentos en el Capítulo 5. Asimismo, se puede encontrar un breve tutorial para la programación de este SDK en el Apéndice A.

2.5.1. Simulación de objetos deformables con *NVIDIA PhysX*

NVIDIA PhysX aplica un esquema de simulación llamado "Dinámica Basada en Posiciones" (Müller, 2007[51]), el cual a diferencia de otros métodos, manipula directamente los vértices de un objeto para calcular variables físicas. De este modo se tiene mayor control sobre colisiones y sobre la interacción entre objetos, ya sean partículas, objetos rígidos u objetos deformables. Por otra parte, como se mencionó con anterioridad, emplea la tecnología CUDA para llevar a cabo el procesamiento de los datos por medio de una PPU. La Figura 2.12 muestra la arquitectura básica del paquete *NVIDIA PhysX*.

Algoritmo de simulación de objetos deformables

Sea un objeto formado por un conjunto de N vértices y M restricciones. Un vértice $i \in [1, ..., N]$ tiene asociada una masa m_i, una posición $\underline{x}_i$ y una velocidad $\underline{v}_i$.

Una restricción $j \in [1, ..., M]$ posee las siguientes características:

- una cardinalidad n_j,
- una función $C_j : \Re^{3n_j} \rightarrow \Re$,
- un conjunto de índices $\{i_1, ..., i_n\}$, $i_k \in [1, ..., N]$,
- un parámetro de rigidez $0 \leq k_j \leq 1$, $k_j \in \Re$,

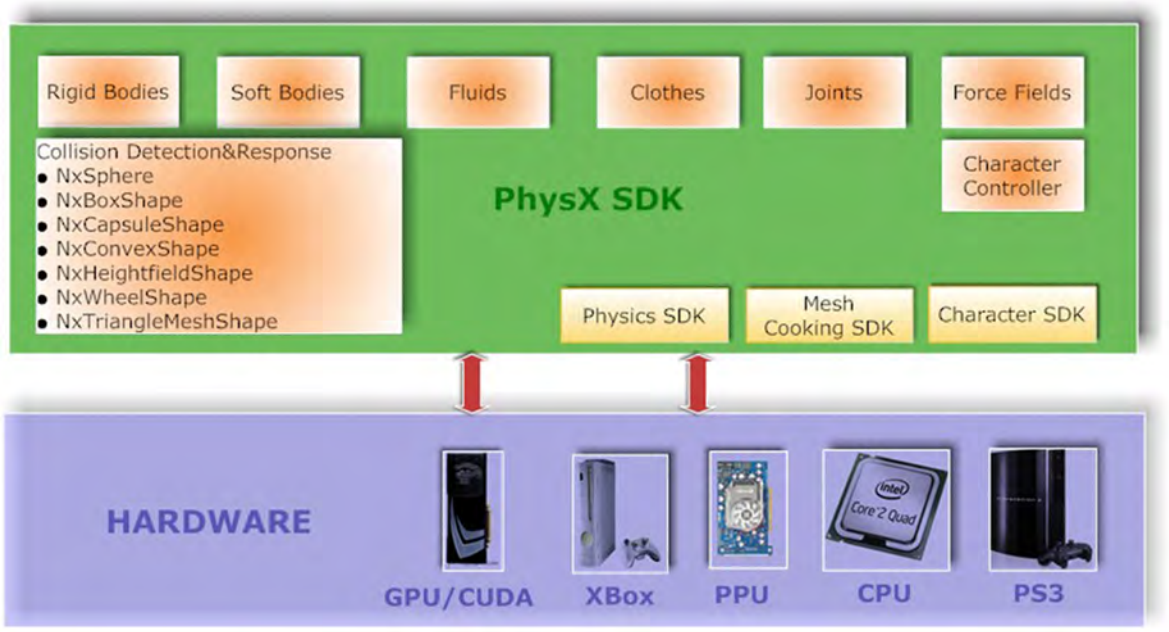


Figura 2.12. *Arquitectura de simulación NVIDIA PhysX [41].*

- un tipo de igualdad o desigualdad. Una igualdad se satisface si $C_j(\underline{x}_{1_j}, ..., \underline{x}_{n_j}) = 0$. Una desigualdad se satisface si $C_j(\underline{x}_{1_j}, ..., \underline{x}_{n_j}) \geq 0$.

Dado un intervalo de tiempo Δt, el objeto dinámico se simula de acuerdo al Algoritmo 1. Las líneas (2) a (6) inicializan las posiciones y se calcula un valor de peso w_i dependiente de la masa, para después calcular la velocidad asociada a cada vértice a partir de la fuerza externa $\underline{f}_{ext}$ que actúa sobre cada uno de ellos – líneas (8) a (10) –. En general, el objetivo de las líneas (8) a (14) es obtener una primera estimación del desplazamiento del vértice mediante integración explícita de Euler. En la línea (11) adicionalmente se toma en cuenta la propiedad de amortiguamiento sobre el conjunto de velocidades; para ello se aplica el Algoritmo 2. La línea (16) genera un conjunto de restricciones dinámicas M_{coll}, que a diferencia de las restricciones iniciales M, van cambiando conforme el objeto deformable interactúa con otros objetos y con su misma deformación. En este punto entra la detección de colisiones, que se lleva a cabo por medio de una búsqueda de tetraedros y puntos en colisión mediante subdivisión espacial o *"spatial hashing"* [79].

A grandes rasgos, el método de *"spatial hashing"* consiste en la subdivisión del espacio de tres dimensiones en paqueñas cajas (AABB's, *Axis-Aligned Bounding Boxes*). En un primer paso, todos los vértices de los objetos son clasificados dentro de las celdas tridimensionales. En un segundo paso, los tetraedros que forman cada malla de volumen son igualmente clasificados con respecto a las mismas celdas. Si un tetraedro intersecta con una celda, todos los vértices que se han clasificado en esa celda en el primer paso se marcan con

Algoritmo 1: Algoritmo de simulación de objetos deformables.

begin
  for $\forall i$ **do**
    $\underline{x}_i = \underline{x}_i^0$;
    $\underline{v}_i = \underline{v}_i^0$;
    $w_i = 1/m_i$;
  end
  while *true* **do**
    for $\forall i$ **do**
      $\underline{v}_i \leftarrow \underline{v}_i + \Delta t \cdot w_i \cdot \underline{f}_{ext}(\underline{x}_i)$;
    end
    dampVelocities($v_1, ..., v_N$);
    for $\forall i$ **do**
      $\underline{p}_i \leftarrow \underline{x}_i + \Delta t \cdot \underline{v}_i$;
    end
    for $\forall i$ **do**
      $M_{coll} \leftarrow$ generateCollisionConstraints($\underline{x}_i \rightarrow \underline{p}_i$);
    end
    while *solverIterations* **do**
      projectConstraints($C_1, ..., C_{M+M_{coll}}, \underline{p}_1, ..., \underline{p}_N$);
    end
    for $\forall i$ **do**
      $\underline{v}_i = (\underline{p}_i - \underline{x}_i)/\Delta t$;
      $\underline{x}_i = \underline{p}_i$;
    end
    velocityUpdate($v_1, ..., v_N$);
  end
end

Algoritmo 2: dampVelocities($v_1, ..., v_N$)

begin
  $\underline{x}_{cm} = (\sum_i \underline{x}_i m_i)/(\sum_i m_i)$;
  $\underline{v}_{cm} = (\sum_i \underline{v}_i m_i)/(\sum_i m_i)$;
  for $\forall i$ **do**
    $\underline{r}_i = \underline{x}_i - \underline{x}_{cm}$;
  end
  $\mathbf{L} = \sum_i \underline{r}_i \times (m_i \underline{v}_i)$;
  $\mathbf{I} = \sum_i \mathbf{R}_i \mathbf{R}_i^T m_i$;
  // $\mathbf{R}_i$ es la matriz de 3×3 que satisface $\mathbf{R}_i \underline{v} = \underline{r}_i \times \underline{v}$
  $\underline{\omega} = \mathbf{I}^{-1}\mathbf{L}$;
  for $\forall i$ **do**
    $\underline{\Delta v}_i = \underline{v}_{cm} + \underline{\omega} \times \underline{r}_i - \underline{v}_i$;
    $\underline{v}_i \leftarrow \underline{v}_i + k_{damping} \cdot \underline{\Delta v}_i$;
    // $k_{damping}$ es el coeficiente de amortiguamiento, $0 \leq k_{damping} \leq 1$
  end
end

ese tetraedro, para después verificar si se encuentra dentro, fuera o pertenece al tetraedro. Si un vértice penetra el tetraedro, se detecta una colisión. Si un vértice pertenece al mismo objeto, se detecta una auto-colisión. Si un vértice es parte del tetraedro, se omite el cálculo de la intersección. El resultado del algoritmo es un conjunto de vértices y puntos que definen los tetraedros en colisión, que al ser proyectados mediante trigonometría básica, permiten definir el conjunto de restricciones M_{coll}, del tipo: $C(\underline{q}, \underline{p}_1, \underline{p}_2, \underline{p}_3) = (\underline{q} - \underline{p}_1) \cdot \underline{n} - h$, donde $\underline{q}$ es el vértice en colisión, $\underline{p}_1, \underline{p}_2, \underline{p}_3$, son los puntos que definen cada triángulo del tetraedro, $\underline{n}$ es la normal del plano de colisión y h es la distancia mínima que deberían guardar los objetos en colisión (Figura 2.13).

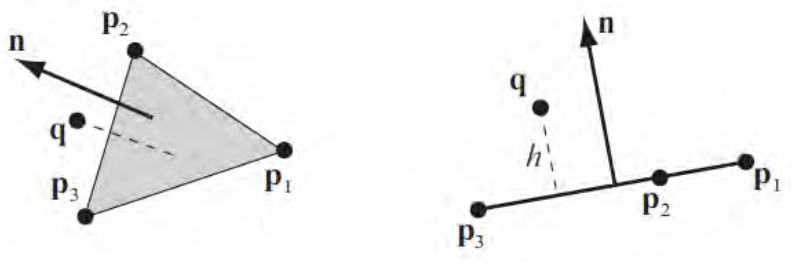


Figura 2.13. *Función de restricción dinámica* $C(\underline{q}, \underline{p}_1, \underline{p}_2, \underline{p}_3) = (\underline{q} - \underline{p}_1) \cdot \underline{n} - h$, *que asegura que un punto* $\underline{q}$ *permanece por encima de un triángulo* $\underline{p}_1, \underline{p}_2, \underline{p}_3$, *al menos la distancia* h *(radio de la partícula) (Tomado de Müler[51], 2007).*

Una vez establecidas todas las funciones de restricción, se conforma un sistema de ecuaciones, que en conjunto con los valores estimados en la línea (13), se resuelve mediante el algoritmo iterativo de Gauss-Seidel (líneas (18)-(20)). Finalmente se actualizan las velocidades instantáneas y las posiciones proyectadas mediante la corrección de cada $\underline{p}$ en el algoritmo iterativo.

Este método puede clasificarse como un método híbrido, dentro de la clasificación dada en la Figura 2.3, pues mezcla un enfoque de masas y resortes de manera implícita, al mismo tiempo que abstrae la idea del método de elemento finito al trabajar con elementos geométricos por separado, para después plantear un sistema de ecuaciones que al resolver describe el comportamiento deformable de un objeto.

Programación

El SDK de *NVIDIA PhysX* provee un conjunto muy completo de funciones y métodos en lenguaje C/C++ para la programación de aplicaciones que involucran la simulación física. Además incorpora un conjunto de ejemplos y tutoriales que incluyen la visualización con funciones primitivas de *OpenGL* y *GLUT*. En este trabajo se ha adicionado el soporte de GUI's usando *Qt* (Ver Apéndice A para mayor información).

En una escena de *PhysX*, el objeto básico es el actor, que tiene asociada una clase de tipo *NxActor*. Hay tres tipos de actores: estáticos, dinámicos y cinemáticos. Los primeros permanecen en una misma posición la mayor parte del tiempo, mientras que los segundos son perturbados por fuerzas externas, lo cual provoca su cambio de posición, rotación o deformación. Los terceros únicamente pueden desplazarse pero no deformarse.

La simulación de objetos rígidos se lleva a cabo mediante actores dinámicos, que tienen asociadas propiedades lineales y propiedades angulares. Entre las propiedades lineales encontramos a la masa del objeto, su posición, velocidad lineal y la fuerza externa. Las propiedades angulares son: el vector de inercia, la orientación, la velocidad angular y el torque. La Tabla 2.2 describe cada una de estas cantidades y su tipo de dato asociado.

La forma de acceder a la información de un objeto rígido es por medio de un descriptor *NxBodyDesc* que contiene, además de las propiedades mostradas, información adicional de la densidad, la fuerza de resistencia al movimiento, fricción, entre otras.

Un cuerpo blando en *PhysX* se considera como un objeto deformable, y está compuesto por una malla de tetraedros y alternativamente una superficie de mapeo, la cual tiene como objetivo la visualización de sólo la frontera del objeto. La clase que implementa dicho tipo de objetos es *NxSoftBodyDesc*, el cual es un descriptor que contiene la información de la malla, como de las propiedades físicas, tales como posición, radio de la partícula, rigidez, fricción, y parámetros de colisión.

Cantidad Lineal	Cantidad Angular
Masa (*NxReal*)	**Inercia** (*NxVector*): Distribución de la masa del cuerpo a lo largo de sus tres dimensiones.
Posición (*NxVector*): Posición del centro de masa, relativo al estado actual.	**Orientación** (*NxQuat* ó *NxMat33*): Orientación de los momentos principales, relativo al estado actual.
Velocidad lineal (*NxVector*)	**Velocidad angular** (*NxVector*): Representado por un eje de rotación con una longitud igual a la magnitud de la velocidad angular.
Fuerza (*NxVector*): Fuerza externa aplicada sobre el objeto.	**Torque** (*NxVector*): Fuerza angular aplicada sobre el objeto. Representado como un eje de rotación con longitud igual a la magnitud de la velocidad angular.

Tabla 2.2. Propiedades de un objeto rígido en el entorno *PhysX (NVIDIA PhysX Tutorials, Lesson 1.7: "Rigidbody Properties", 2008)*.

Los parámetros que influyen sobre la física de un objeto deformable en PhysX son la rigidez, el coeficiente de amortiguamiento y la fricción, por lo que el proceso de calibración de un modelo puede llevarse tiempo de experimentación, dependiendo de la aplicación. Para simulación de tejido blando, en la sección de experimentos podrá encontrar algunos valores usados para estas cantidades físicas.

Sección 2.6

Casos de estudio: Cirugía Laparoscópica y Cirugía de Próstata usando *PhysX*

En medicina, existe una gran variedad de procedimientos quirúrgicos. El área encargada de estudiar y clasificar estos métodos es la cirugía.

Estudiar todo un conjunto de técnicas para simulación puede conllevar a una tarea muy compleja, por lo que investigadores en el área de simulación quirúrgica deciden enfocarse en un padecimiento específico, y a su vez que pueda ser escalable a otras aplicaciones. Los principales motivos a elegir son: la necesidad específica del médico, la incidencia del padecimiento o bien la cantidad de elementos involucrados para fines de investigación.

La simulación de tejido y las operaciones que deben realizarse sobre éste, son elementos fundamentales si se desea obtener un producto realista. Adicionalmente, todos los objetos que intervienen en una cirugía también adicionan grados de realismo. Por ejemplo, simular el instrumento de corte, audio o simplemente un efecto de sangrado, dan al usuario la inmersividad necesaria para crear en su mente la ilusión de estar en una cirugía real.

En esta sección abordaremos dos casos de estudio, el primero un simulador de Laparoscopía Gástrica [41] y el segundo, un simulador de resección transuretral de la próstata [62][63][75][76].

2.6.1. Simulador de laparoscopía esofágica

Este sistema fue desarrollado en un trabajo de colaboración entre la Universidad Federal de Río Grande del Sur de Brasil y el Instituto Politécnico de Rensselaer en Estados Unidos [41]. El simulador se apega a un modelo de simulación que en los últimos años ha tenido éxito entre la comunidad científica. La idea básica consiste en un sistema compuesto de un instrumento de interacción hombre-máquina, un bloque de procesamiento de datos y un sistema de visualización. La Figura 2.14 muestra estos bloques genéricos.

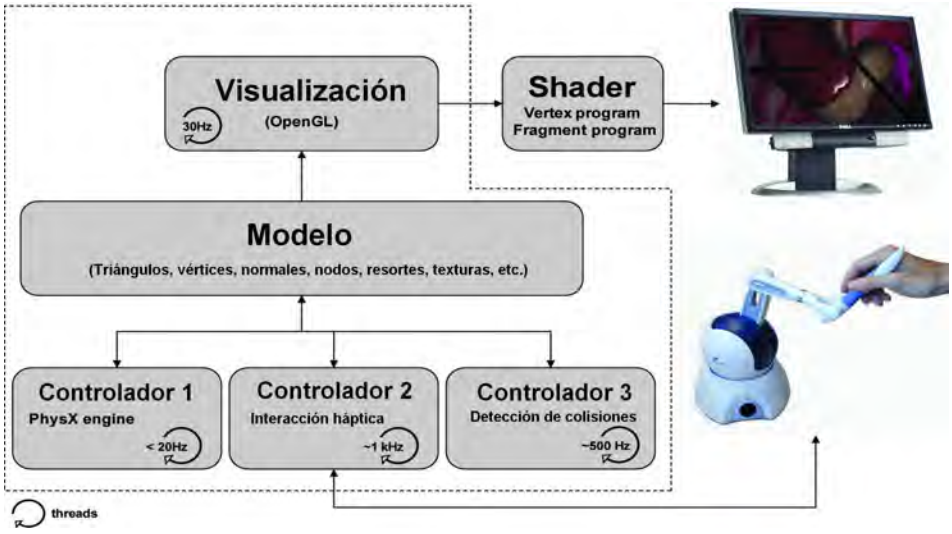


Figura 2.14. *Modelo de simulación para un sistema de Laparoscopía gástrica [41]. El modelo consta de bloques de procesamiento de datos, un sistema de retroalimentación háptica y un sistema de visualización.*

El sistema emplea el motor de *NVIDIA PhysX* para todo el cálculo de deformaciones e interacción con instrumentos. Para la visualización se emplea *OpenGL* en combinación con GLSL (*OpenGL Shaders Languague*, Lenguaje de Shaders de *OpenGL*). Este último para efectos de iluminación. El sistema corre en una PC de escritorio con procesador Intel(R) Core^{TM} 2 Quad 2.66 GHz, con una tarjeta *nVidia GeForce* 8800 GTX, sistema de visualización a 60 Hz y 30 Hz en modo estéreo.

Un experimento importante realizado durante la implementación de este simulador fue la comparación que llevaron a cabo entre una malla deformable usando el método de Elemento Finito y PhysX. Información no disponible en el estudio de Meier[43]. La comparación resultó en una deformación muy similar y por tanto concluyen que PhysX es

una buena alternativa para simuladores en tiempo real en Medicina (Figura 2.15), aunque reconocen las limitantes por no ser de código abierto.

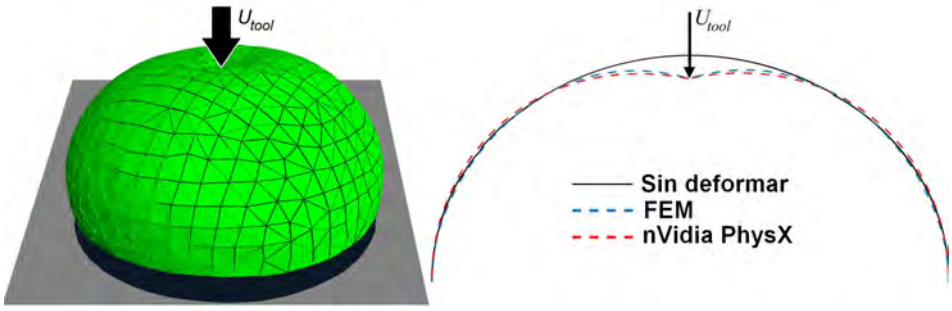


Figura 2.15. *Comparativa entre la deformación provocada usando el método híbrido de PhysX y el método de Elemento Finito (Maciel et. al. [41], 2009).*

2.6.2. El simulador de RTUP de la UNAM

El simulador de Resección Transuretral de la Próstata (RTUP) de la UNAM, para entrenamiento médico, es un sistema en desarrollo por el Centro de Ciencias Aplicadas y Desarrollo Tecnológico de la UNAM. Consiste en dos módulos principales: el primero, una interfase mecatrónica que simula los movimientos que un cirujano puede ejecutar sobre una herramienta real, el segundo, un ambiente virtual que incluye navegación e interacción con tejido. El modelo computacional es muy similar al empleado por *Maciel [41] et. al.* en el simulador de laparoscopía esofágica.

La interacción con la herramienta mecatrónica se implementa en dos modalidades: una interfaz activa (*Santiago [68], 2010*), que provee retroalimentación háptica (robot *Sensible PHANTOM Omni*®) y un diseño pasivo que únicamente permite al sistema visual sensar los movimientos del usuario (*Reyes [66], 2010*); sin otorgar a este último efectos de retroalimentación de fuerzas adicionales a las causadas por la manipulación. El objetivo de emplear dos tipos de interfases es evaluar y comparar los beneficios adquiridos por el practicante al usar cada uno por separado.

La interfaz pasiva se comunica con el software de simulación 3D por medio del protocolo de comunicación USB. Básicamente, el diseño electrónico permite enviar los datos de la herramienta a la computadora. La información del mundo real es adquirida por medio de *encoders*, cuyo objetivo es sensar los movimientos producidos durante la manipulación. Un conjunto de tres contadores de cuadratura se encargan de transformar la información en valores contables y enviarlos a un microcontrolador PIC, el cual está programado para convertir los datos adquiridos, en datos entendibles por el programa de simulación gráfico. El diseño mecánico consiste en una varilla dentada para sensar los desplazamientos longitudinales, un arreglo de discos encargados de ofrecer los suficientes grados de libertad

para rotaciones y un arreglo de sensores ópticos para el desplazamiento de la herramienta de corte.

Para la modalidad de interfaz activa, se usa un robot comercial, el *PHANTOM Omni*® de la empresa Sensable Technologies. Este dispositivo emplea el protocolo de comunicación IEEE-1394a FireWire y es programable bajo su propio SDK. Comparado con la versión pasiva, este dispositivo ya tiene implementados módulos, tanto a nivel de software como de hardware, para la obtención de información de las rotaciones y traslaciones realizadas por el usuario, requerida por el sistema gráfico.

Finalmente, el ambiente virtual está compuesto por un conjunto de modelos anatómicos deformables usando el motor de física de NVIDIA, efectos de visualización con *OpenGL* y efectos de audio; ejecutándose en tiempo real (aprox. $25Hz$) en una computadora *Sun Microsystems* con procesador Dual-Core AMD Opteron 1222, 3.0 GHz, 4 GB RAM, *nVidia Quadro* 3700 con soporte para *PhysX* y *Windows 7* (x64) *Professional Edition*. Los modelos fueron creados mediante técnicas de generación de tetraedros (Ver Capítulo 3) y texturizados mediante mapeo 3D con texturas procedurales que simulan vascularidades (*García [24], 2009*). En cuanto a efectos, implementa un sistema de partículas para simular sangrado y burbujas (*Flores [22], 2011*), además de efectos de iluminación y sombreado [75]. Los efectos de audio son ejecutados en sincronía con el ambiente mediante un dispositivo que simula un sistema de pedales reales, y cuya función es activar un estado de corte o un estado de coagulación. Por su simplicidad y buenos resultados en cuanto a visualización, se implementa sobre el sistema un método de corte por eliminación de elementos, cuyos detalles pueden encontrarse en el Capítulo 4 de este trabajo. La Figura 2.16 muestra el simulador de RTU en su versión actual ilustrando un procedimiento de resección por parte de un médico especialista.

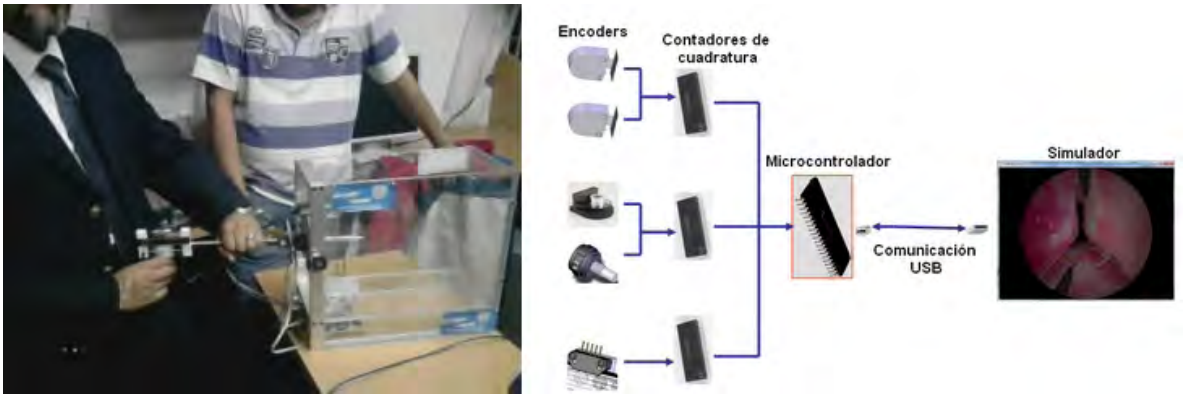


Figura 2.16. *Simulador de RTUP que implementa deformaciones y cortes sobre un modelo virtual de tetraedros. A la izquierda, interfaz mecatrónica de interacción. A la derecha, ambiente virtual.*

CAPÍTULO 3

Generación de volúmenes de tetraedros

En una gran cantidad de aplicaciones que involucran la representación de objetos deformables, se requiere de la simulación de comportamiento físico para dar la impresión de realismo (*Müller et. al., [49]*). Para tal objetivo, el modelo físico actuará sobre un conjunto de elementos geométricos en el espacio de una, dos, o tres dimensiones.

Partiendo de la idea anterior se puede decir que, todo objeto en el espacio n-dimensional puede ser dividido o discretizado en un conjunto finito de elementos básicos de geometría conocida. En el espacio de una dimensión el elemento básico es la línea, en dos dimensiones el triángulo (superficie) y en tres dimensiones el tetraedro (volumen). En este capítulo nos concentraremos en el análisis de las estructuras de volumen.

Un modelo geométrico de volumen está compuesto por dos elementos, un conjunto de puntos o vértices finitos $v_i = (x_i, y_i, z_i)$ ($i = [0, n-1]$, n = número de vértices, $n > 3$) y una función de correspondencia $T_j = (i_1, i_2, i_3, i_4)$ ($j = [0, N-1]$, N = número de tetraedros, $N > 0$), donde i_1, i_2, i_3 e i_4 son los índices a los vértices que forman el tetraedro. Por ejemplo, supongamos que se tiene el siguiente modelo:

$$
\begin{aligned}
v_0 &= (0.5, 0.5, -0.5) \\
v_1 &= (-0.5, 0.5, -0.5) \\
v_2 &= (-0.5, -0.5, -0.5) \\
v_3 &= (0.5, -0.5, -0.5) \\
v_4 &= (0.5, 0.5, 0.5) \\
v_5 &= (-0.5, 0.5, 0.5) \\
v_6 &= (-0.5, -0.5, 0.5) \\
v_7 &= (0.5, -0.5, 0.5)
\end{aligned}
$$

$$
\begin{aligned}
T_0 &= (0,1,3,4)\\
T_1 &= (1,4,5,6)\\
T_2 &= (1,2,3,6)\\
T_3 &= (3,4,6,7)\\
T_4 &= (1,3,4,6)
\end{aligned}
$$

En éste se define un hexaedro (cubo) unitario centrado en el origen, con 8 vértices y 5 tetraedros (Figura 3.1). Podemos decir entonces que un vértice del modelo es $v_4 = (0.5, 0.5, 0.5)$ y un tetraedro es T_0, formado por los vértices 0, 1, 3 y 4.

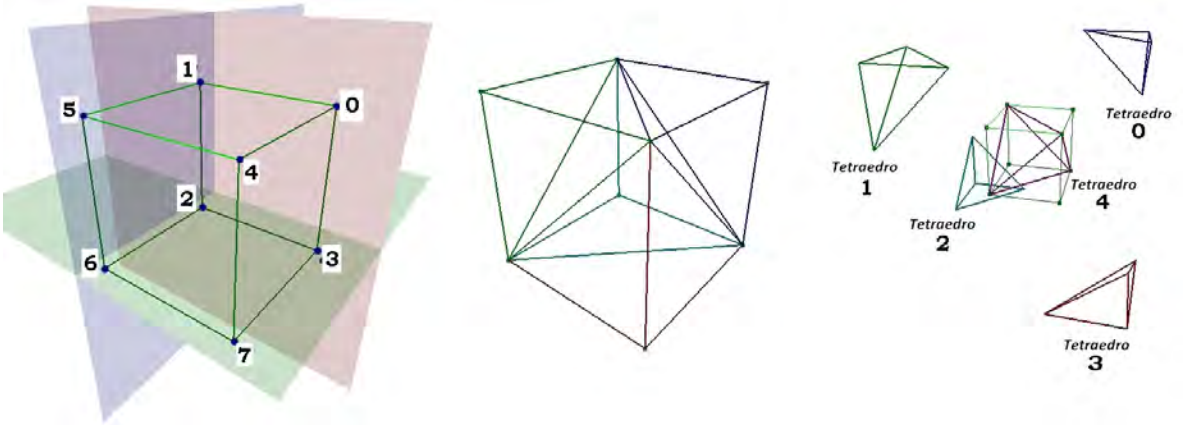


Figura 3.1. *Discretización de un cubo unitario en elementos básicos (tetraedros).*

Se comienza introduciendo con esta idea básica porque a partir de ella se pueden describir objetos más complejos, que computacionalmente son más sencillos de simular a través de una estructura de datos. En las siguientes secciones analizaremos la construcción de objetos de volumen a partir de diferentes fuentes de información: primitivas matemáticas, imágenes y superficies triángulares, para después describir un formato de archivo, propuesto en este trabajo, para su portabilidad computacional, el formato VOG (*Volume Object Geometry*).

Sección 3.1

Generación de volúmenes simples

El proceso de tetraedrización de un volumen, determinado mediante una expresión matemática o de estructura conocida (llamaremos a este tipo de geometrías como volúmenes simples), consiste en la subdivisión del espacio inicialmente en hexaedros, que posterior-

mente se subdividen en cinco tetraedros, dispuestos en una configuración uniforme o una configuración alternada.

Definimos la configuración de un hexaedro, como la relación que guardan sus vértices con la formación de cinco tetraedros; esto es, cada uno de los tetraedros estarán definidos por cuatro de los ocho vértices del hexaedro, referenciados mediante índices. Así, sean A y B dos configuraciones para referenciar los vértices del hexaedro de la Figura 3.1, dadas por:

$$\begin{aligned} A &= \{(0,1,3,4),(1,4,5,6),(1,2,3,6),(3,4,6,7),(1,3,4,6)\} \\ B &= \{(0,1,5,2),(0,5,4,7),(0,3,2,7),(2,5,6,7),(0,2,5,7)\} \end{aligned}$$

Decimos que un volumen se encuentra subdividido en configuración uniforme, cuando todos los hexaedros que lo conforman se subdividen en tetraedros con una configuración A o B. Del mismo modo, decimos que un volumen se encuentra subdividido en configuración alternada, cuando la subdivisión de tetraedros se lleva a cabo alternando la configuración A y B de acuerdo a un criterio de correspondencia, que puede ser en primera instancia la paridad del hexaedro. La Figura 3.2 muestra un ejemplo de estas configuraciones.

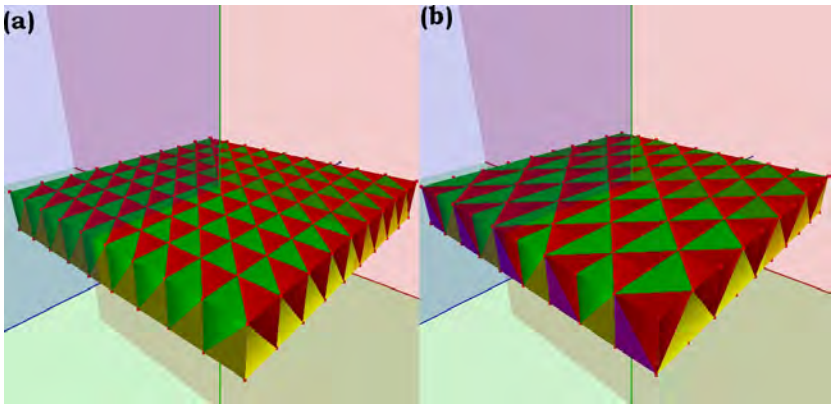


Figura 3.2. *Tipos de configuraciones en la tetraedrización de volúmenes simples. a) Configuración uniforme. b) Configuración alternada.*

Paralelepípedo rectangular

Sea el volumen simple con forma de paralelepípedo rectangular, mostrado en la Figura 3.3, de dimensiones L_x, L_y y L_z. Si se subdivide el espacio que ocupa en hexaedros regulares, como el mostrado en la Figura 3.1, con resoluciones R_x, R_y y R_z, el número de vértices y tetraedros están dados por:

$$\begin{aligned} n_v &= (R_x+1)(R_y+1)(R_z+1) \\ N_t &= 5 \cdot R_x \cdot R_y \cdot R_z \end{aligned}$$

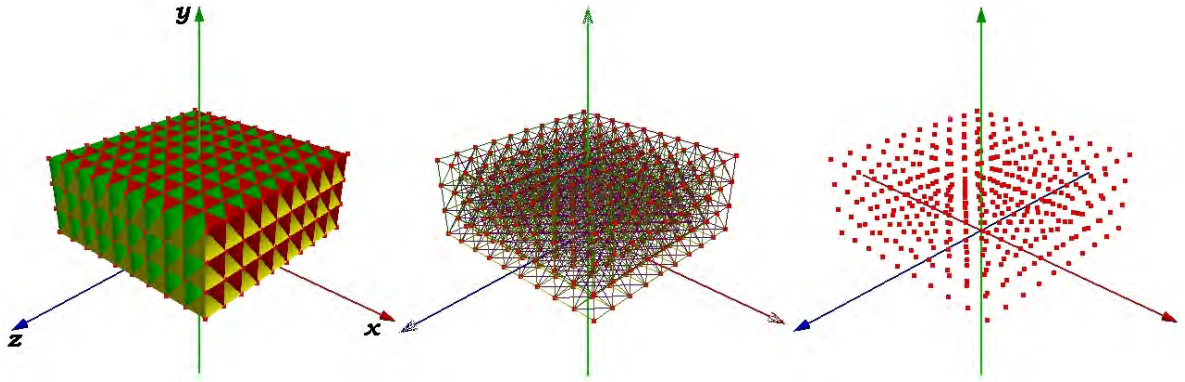


Figura 3.3. *Tetraedrización de un paralelepípedo regular.*

Sean Δx, Δy y Δz los intervalos de separación entre cada vértice, dados por:

$$\Delta x = \frac{L_x}{R_x} \qquad \Delta y = \frac{L_y}{R_y} \qquad \Delta z = \frac{L_z}{R_z}$$

Los vértices del paralelepípedo rectangular están dados por:

$$v_i(x, y, z) = \left(x\Delta x - \frac{L_x}{2}, y\Delta y, z\Delta z - \frac{L_z}{2} \right)$$

donde $0 \leq x \leq R_x$, $0 \leq y \leq R_y$ y $0 \leq z \leq R_z$ con incrementos de una unidad.

De acuerdo con lo anterior, un paralelepípedo rectangular puede subdividirse en un conjunto de hexaedros regulares, y éstos a su vez en tetraedros mediante el Algoritmo 3. Este algoritmo genera en una primera fase, todos los puntos o vértices que discretizan la geometría [líneas (5)-(14)]. Los puntos generados se guardan en una lista **verticesList** [línea (11)], para después ser referenciados mediante sus correspondientes índices en memoria. Puede observarse que la geometría es paralela al plano XZ, centrado en el origen y subdividido en capas sobre el eje Y positivo. Las líneas (15)-(17) realizan el segundo paso, consistente en la creación de índices que definen un conjunto de hexaedros contiguos y que forman un primer mallado de celdas, tales como el de la Figura 3.1.

El algoritmo 4 se ejecuta en la línea (16) procesando cada capa y generando un lista de índices a vértices **indicesList**, para formar conjuntos de 8 elementos (8 vértices). En geometrías planas se realiza una única indización, mientras que para geometrías tubulares (como las que se muestran más adelante), se llevan a cabo dos procesos de indización: uno para definir el conjunto de hexaedros que envuelven la geometría y otro para definir el conjunto de hexaedros que cierran dicha geometría.

Finalmente, en la línea (18) se ejecuta el Algoritmo 5, que subdivide cada hexaedro de 8 vértices en cinco tetraedros con configuración uniforme o alternada (con criterio de

Algoritmo 3: Subdivisión de un paralelepípedo rectangular.

```
1  begin
2      Δx = L_x/R_x ;
3      Δy = L_y/R_y ;
4      Δz = L_z/R_z ;
5      for y = [0 : R_y] do
6          for x = [0 : R_x] do
7              for z = [0 : R_z] do
8                  v_x = xΔx − L_x/2;
9                  v_y = yΔy;
10                 v_z = zΔz − L_z/2;
11                 verticesList.push(v);
12             end
13         end
14     end
15     for layer = [0 : R_y − 1] do
16         createHexLayer(layer, close = false);
17     end
18     createTetrahedra();
19 end
```

paridad). El resultado se guarda en una lista lineal de índices **tetraedra**; la cual, en conjunto con la lista de vértices **verticesList**, definen la nueva geometría discretizada.

Cilindro

Supongamos que al volumen que representa un paralelepípedo rectangular, se le dobla como a una hoja hasta formar un tubo con espesor, entonces podemos formular un volumen cilíndrico de la siguiente manera.

Sea el volumen cilíndrico mostrado en la Figura 3.4, con eje paralelo al eje Z, de dimensiones r_i (radio interno), r_e (radio externo) y h (altura). Si se subdivide el espacio que ocupa en hexaedros regulares, como el mostrado en la Figura 3.1, con resoluciones R_r, R_θ y R_h, el número de vértices y tetraedros están dados por:

$$\begin{aligned} n_v &= R_\theta \cdot (R_r + 1)(R_h + 1) \\ N_t &= 5 \cdot R_\theta \cdot R_r \cdot R_h \end{aligned}$$

Sean $\Delta\theta$, Δr y Δh los intervalos de separación entre cada vértice, dados por:

$$\Delta\theta = \frac{360}{R_\theta} \qquad \Delta r = \frac{r_e - r_i}{R_r} \qquad \Delta h = \frac{h}{R_h}$$

Algoritmo 4: Algoritmo de hexaedrización por capas: createHexLayer(*layer*, *close*)

```
1  begin
2      layerOffset = layer ∗ (R_x + 1) ∗ (R_z + 1);
3      offset= (R_x + 1) ∗ (R_z + 1);
       // Primer grupo de hexaedros
4      for i = [0 : R_x] do
5          for j = [0 : R_z] do
6              i_1 =layerOffset+(i ∗ (R_z + 1) + j);
7              i_2 = i_1 + 1;
8              i_3 =layerOffset+((i + 1) ∗ (R_z + 1) + j + 1);
9              i_4 = i_3 − 1;
10             i_5 =offset+i_1;
11             i_6 =offset+i_2;
12             i_7 =offset+i_3;
13             i_8 =offset+i_4;
14             indicesList.push(i_1, ..., i_8);
15         end
16     end
       // Segundo grupo de hexaedros (Geometrías tubulares)
17     if close = true then
18         j = R_z + 1;
19         for i = [0 : R_x] do
20             i_1 =layerOffset+((i + 1) ∗ j − 1);
21             i_2 =layerOffset+(i ∗ j);
22             i_3 =layerOffset+((i + 1) ∗ j);
23             i_4 =layerOffset+((i + 2) ∗ j − 1);
24             i_5 =offset+i_1;
25             i_6 =offset+i_2;
26             i_7 =offset+i_3;
27             i_8 =offset+i_4;
28             indicesList.push(i_1, ..., i_8);
29         end
30     end
31 end
```

Algoritmo 5: Algoritmo de tetraedrización de un conjunto de hexaedros: createTetrahedra()

```
1  begin
2      id = 0;
3      Cfg = {(0,1,3,4),(1,4,5,6),(1,2,3,6),(3,4,6,7),(1,3,4,6)};
4      for i = [0 :indicesList_size], i+ = 8 do
5          if alternateConfiguration then
6              if id %2 = 1 then
7                  Cfg = {(0,1,3,4),(1,4,5,6),(1,2,3,6),(3,4,6,7),(1,3,4,6)};
8              end
9              else
10                 Cfg = {(0,1,5,2),(0,5,4,7),(0,3,2,7),(2,5,6,7),(0,2,5,7)};
11             end
12         end
13         for j=[0:4] do
14             i_0 = indicesList[i+Cfg[j][0]];
15             i_1 = indicesList[i+Cfg[j][1]];
16             i_2 = indicesList[i+Cfg[j][2]];
17             i_3 = indicesList[i+Cfg[j][3]];
18             tetrahedra.push(t(i_0, i_1, i_2, i_3));
19         end
20         id + +;
21     end
22 end
```

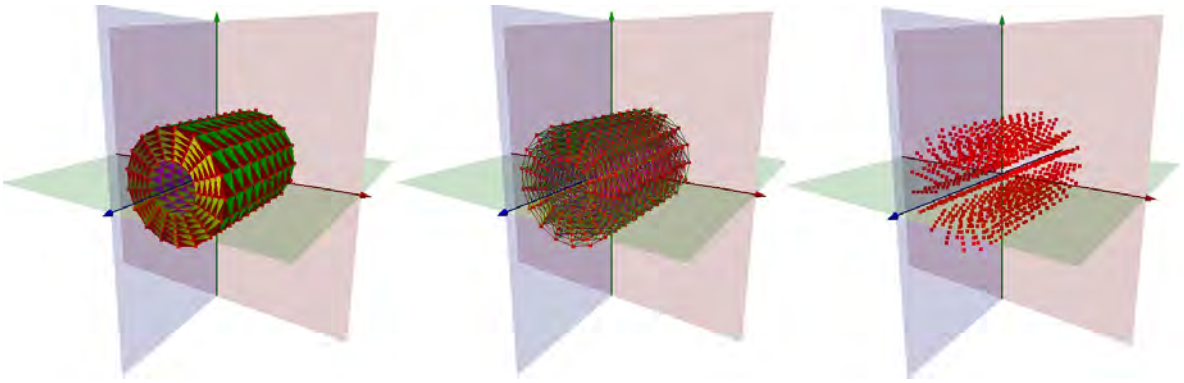


Figura 3.4. *Tetraedrización de un cilindro regular.*

Los vértices del cilindro están dados por:

$$v_i(r,h,\theta) = \begin{cases} x &= (r_i + r\Delta r)\cdot cos(\theta\cdot\Delta\theta) \\ y &= (r_i + r\Delta r)\cdot sen(\theta\cdot\Delta\theta) \\ z &= h\cdot\Delta h - \frac{h}{2} \end{cases}$$

donde $0 \leq r \leq R_r$, $0 \leq h \leq R_h$ y $0 \leq \theta \leq R_\theta$, con incrementos de una unidad.

Tomando en cuenta la abstracción del cilindro con la hoja de papel y las características anteriores, el Algoritmo 6 subdivide el espacio de volumen de un cilindro con espesor formado por tetraedros. De igual manera se emplean los Algoritmos 4 y 5 para concretar el proceso.

Algoritmo 6: Subdivisión de un cilindro regular.

```
1  begin
2      for r = [0 : R_r] do
3          for h = [0 : R_h] do
4              for θ = [0 : R_θ] do
5                  v_x = (r_i + rΔr) · cos(θ · Δθ);
6                  v_y = (r_i + rΔr) · sen(θ · Δθ);
7                  v_z = h · Δh − h/2;
8                  verticesList.push(v);
9              end
10         end
11     end
12     for layer = [0 : R_r − 1] do
13         createHexLayer(layer, close = true);
14     end
15     createTetrahedra();
16 end
```

Esfera

Sea el volumen esférico mostrado en la Figura 3.5, con eje polar en el eje Y, de dimensiones r_i (radio interno) y r_e (radio externo). Si se subdivide el espacio que ocupa en hexaedros regulares, como el mostrado en la Figura 3.1, con resoluciones R_r (resolución radial), R_ϕ (resolución paralela) y R_θ (resolución meridional), el número de vértices y tetraedros están dados por:

$$\begin{aligned} n_v &= R_\theta\cdot(R_r+1)(R_\phi+1) \\ N_t &= 5\cdot R_\theta\cdot R_r\cdot R_\phi \end{aligned}$$

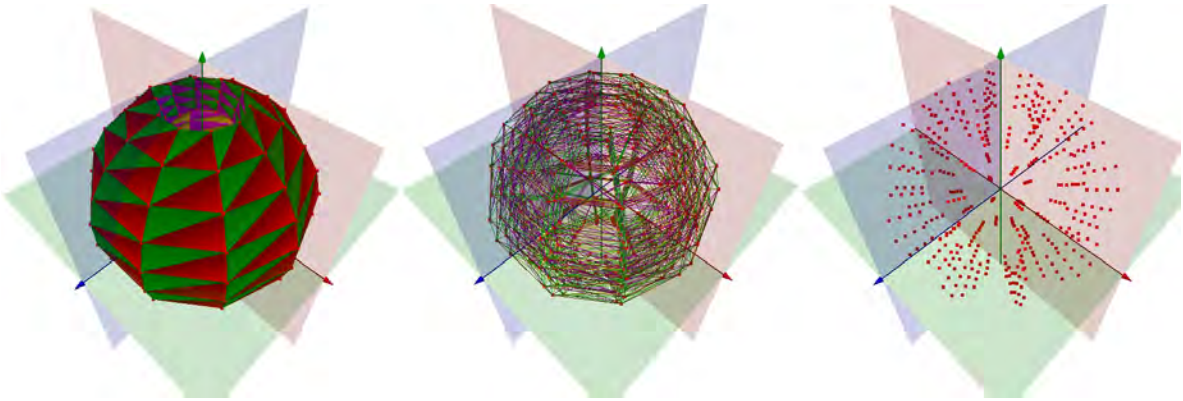


Figura 3.5. *Tetraedrización de una esfera.*

Sean $\Delta\theta$, Δr y $\Delta\phi$ los intervalos de separación entre cada vértice, dados por:

$$\Delta\theta = \frac{360}{R_\theta} \qquad \Delta\phi = \frac{180}{R_\phi} \qquad \Delta r = \frac{r_e - r_i}{R_r}$$

Los vértices de la esfera están dados por:

$$v_i(r,\phi,\theta) = \begin{cases} x &= (r_i + r\Delta r) \cdot cos(\theta \cdot \Delta\theta) \cdot sen(\phi \cdot \Delta\phi) \\ y &= (r_i + r\Delta r) \cdot cos(\phi \cdot \Delta\phi) \\ z &= (r_i + r\Delta r) \cdot sen(\theta \cdot \Delta\theta) \cdot sen(\phi \cdot \Delta\phi) \end{cases}$$

donde $0 \leq r \leq R_r$, $0 \leq \phi \leq R_\phi$ y $0 \leq \theta \leq R_\theta$, con incrementos de una unidad. El Algoritmo 7 muestra el proceso de tetraedrización.

Cardiode cilíndrico

Sea el volumen cilíndrico mostrado en la Figura 3.6, con eje paralelo al eje Z, de dimensiones r_i (radio interno), r_e (radio externo) y h (altura). Si se subdivide el espacio que ocupa en hexaedros regulares, como el mostrado en la Figura 3.1, con resoluciones R_r, R_θ y R_h, el número de vértices y tetraedros están dados por:

$$\begin{aligned} n_v &= R_\theta \cdot (R_r + 1)(R_h + 1) \\ N_t &= 5 \cdot R_\theta \cdot R_r \cdot R_h \end{aligned}$$

Sean $\Delta\theta$, Δr y Δh los intervalos de separación entre cada vértice, dados por:

$$\Delta\theta = \frac{360}{R_\theta} \qquad \Delta r = \frac{r_e - r_i}{R_r} \qquad \Delta h = \frac{h}{R_h}$$

Algoritmo 7: Subdivisión de una esfera.

```
1  begin
2      for r = [0 : R_r] do
3          for φ = [0 : R_φ] do
4              for θ = [0 : R_θ] do
5                  v_x = (r_i + rΔr) · cos(θ · Δθ) · sen(φ · Δφ);
6                  v_y = (r_i + rΔr) · cos(φ · Δφ);
7                  v_z = (r_i + rΔr) · sen(θ · Δθ) · sen(φ · Δφ);
8                  verticesList.push(v);
9              end
10         end
11     end
12     for layer = [0 : R_r − 1] do
13         createHexLayer(layer, close = true);
14     end
15     createTetrahedra();
16 end
```

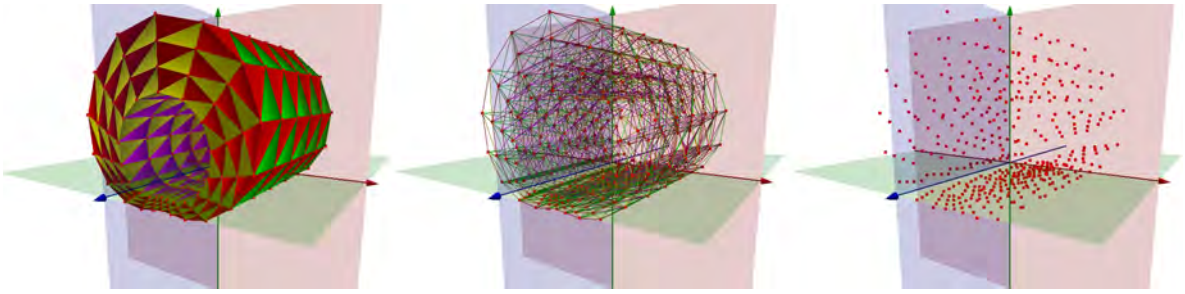


Figura 3.6. *Tetraedrización de un cardioide cilíndrico.*

Los vértices del cardioide cilíndrico están dados por:

$$v_i(r,h,\theta) = \begin{cases} x &= (r_i + r\Delta r)(1.5 + cos(\theta \cdot \Delta\theta)) \cdot sen(\theta \cdot \Delta\theta) \\ y &= (r_i + r\Delta r)(1.5 + cos(\theta \cdot \Delta\theta)) \cdot cos(\theta \cdot \Delta\theta) \\ z &= h \cdot \Delta h - \frac{h}{2} \end{cases}$$

donde $0 \leq r \leq R_r$, $0 \leq h \leq R_h$ y $0 \leq \theta \leq R_\theta$, con incrementos de una unidad. El Algoritmo 8 muestra el proceso de tetraedrización.

Algoritmo 8: Subdivisión de un cardioide cilíndrico.

```
1  begin
2      for r = [0 : R_r] do
3          for h = [0 : R_h] do
4              for θ = [0 : R_θ] do
5                  v_x = (r_i + rΔr)(1.5 + cos(θ · Δθ)) · sen(θ · Δθ);
6                  v_y = (r_i + rΔr)(1.5 + cos(θ · Δθ)) · cos(θ · Δθ);
7                  v_z = h · Δh − h/2;
8                  verticesList.push(v);
9              end
10         end
11     end
12     for layer = [0 : R_r − 1] do
13         createHexLayer(layer, close = true);
14     end
15     createTetrahedra();
16 end
```

Sección 3.2

Generación a partir de imágenes médicas

Una de las grandes fuentes de información en el diagnóstico médico y en el estudio anatómico son las imágenes, siendo la ciencia que estudia este tipo de información la llamada imagenología médica. Montaño [44] define a la imagenología como una disciplina de la medicina que emplea diferentes modalidades de imágenes del cuerpo humano, obtenidas mediante un conjunto de equipos y métodos para llegar en forma rápida y segura a la detección de muchas enfermedades; la describe como una herramienta imprescindible para la atención adecuada y calificada de los pacientes. Por otra parte, Thalmann [81] sostiene que el reto para este campo de estudio es proporcionar las capacidades y técnicas de avanzada para la adquisición, el procesamiento, visualización y el análisis de imágenes biomédicas, que permitan la extracción confiable de información científica y clínica.

En esta sección nos concentraremos en la descripción de una metodología para la generación de una malla de volumen a partir de imágenes, especialmente de imágenes médicas. Sin embargo, no a tal detalle que implica todo el proceso, sino a partir de imágenes que ya han sido preprocesadas, y que contienen la información necesaria para la reconstrucción del volumen. En la Figura 3.7 se muestra una metodología genérica para dicha tarea.

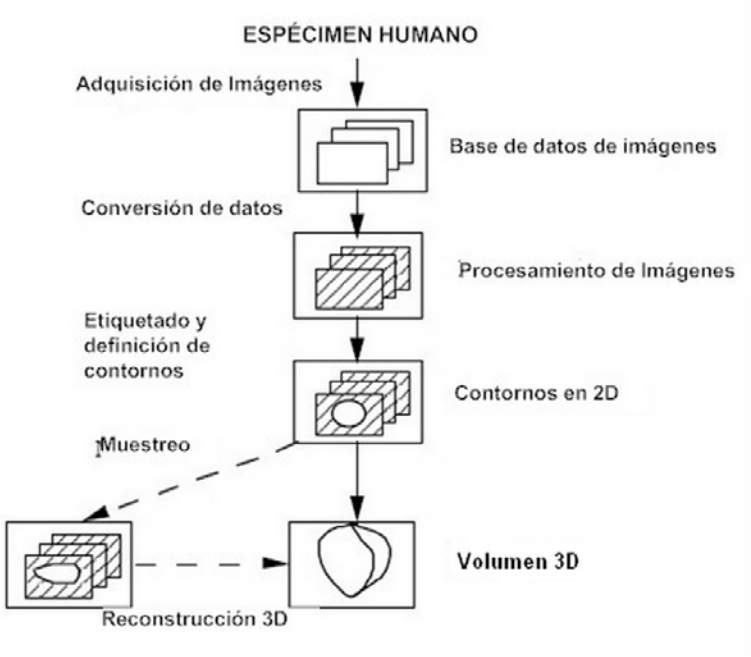


Figura 3.7. *Reconstrucción 3D de modelos anatómicos de volumen, empleando una técnica de contornos (Thalmann [81], Teodoro [75]).*

1. Adquisición de imágenes. Consiste en la obtención de imágenes del objetivo de estudio usando dispositivos especializados de imagen; como por ejemplo, tomógrafos o cámaras fotográficas de alta resolución. Las imágenes obtenidas se almacenan en formatos estándar para tomografía computarizada (CT), ultrasonido, termográficas o anatómicas.

2. Base de datos. Es el conjunto de imágenes obtenidas, dispuestos para su estudio en medios de almacenamiento. Un ejemplo de estas bases es el Proyecto del Humano Visible (*VHP, Visible Human Project* [84]), de la Biblioteca Nacional de Medicina de los Estados Unidos de América, donde se tiene un conjunto de imágenes anatómicas de cuerpos diseccionados *post-mortem*.

3. Conversión de datos. En ocasiones las imágenes dispuestas en las bases de datos vienen en formatos variados, a diferentes resoluciones, en modelos de color específicos, con ruido, etc., por tanto, es necesario preparar los datos en un formato uniforme,

preprocesando cada imagen para ser adaptada a un estándar, y poniendo énfasis en el objetivo a reconstruir.

4. Procesamiento. Consiste en la aplicación de operaciones propias del procesamiento de imágenes, tales como filtrado, registro y segmentación, para aislar la información importante de un modelo anatómico.

5. Contornos en 2D. Consiste en la binarización de las imágenes para obtener un conjunto de planos con los contornos del objeto. En el caso de un modelo de superficie, las imágenes contendrán un solo contorno que define el perímetro del objeto de estudio. En el caso de un modelo de volumen, las imágenes contendrán un contorno con área, delimitada por un contorno externo y para volúmenes huecos, con un contorno interno.

6. Muestreo. Consiste en la discretización de cada contorno en un conjunto de puntos, a partir de un criterio de distancia. Para el muestreo de contornos que definen una superficie, todos los puntos forman parte del contorno. En el muestreo de puntos que definen un volumen, los puntos se encuentran sobre la periferia y dentro del área que define la forma del objeto.

7. Reconstrucción 3D. Consiste en la aplicación de un algoritmo que asocia cada punto en cada imagen, con un elemento geométrico en el espacio de tres dimensiones; para superficies el elemento es el triángulo, para volúmenes el elemento es el tetraedro.

8. Volumen 3D. Es el resultado final de la reconstucción, listo para ser empleado en la simulación de comportamientos basados en física, tales como las deformaciones u operaciones de corte o fractura.

Así, agregando algunas restricciones para la proposición de un método de generación, se tomarán en cuenta algunas consideraciones:

- Sólamente es aplicable para geometrías tubulares huecas con espesor. Por ejemplo, para la generación de conductos anatómicos, como uréteres, vasos, próstata, esófago, etc.

- La geometría debe tener un eje correspondiente a la parte hueca y a alguno de los ejes coordenados. Para tal objetivo, se aplica sobre el conjunto de contornos un algoritmo de alineación y registro.

- En consecuencia de la anterior, los planos paralelos que definen los cortes deben ser transversales al eje; lo que significa en cortes axiales.

- Para geometrías no huecas, se recomienda emplear alguno de los otros métodos descritos más adelante: generación a partir de superficies (lo que implica primero generar una superficie del modelo) o el método de tetraedrización de Delaunay.

3.2.1. Preprocesamiento

En 2008, se presentó un trabajo de reconstrucción de mallas de superficie a partir de imágenes, en la tesis *Modelado de un ambiente virtual para un sistema de simulación de cirugía de próstata* [75], para la construcción de un modelo de una vejiga virtual. El procedimiento propuesto a continuación representa una extensión a mallas de volumen, con un caso de estudio, la reconstrucción de un modelo virtual de la próstata, para ser empleado en el mismo simulador de cirugía virtual.

3.2.2. Muestreo

El muestreo trabaja sobre imágenes que han sido obtenidas de un proceso de segmentación y que se apegan a las restricciones antes descritas. El objetivo es obtener un conjunto de puntos por cada imagen, de tal forma que puedan ser posicionados en un espacio 3D como vértices de la malla, tal como se hizo con las geometrías básicas de la sección anterior. Una vez definidos los puntos de la malla, que necesariamente deben tener una formación conocida, se aplican los algoritmos 4 y 5. El resultado es una malla de tetraedros útil para aplicar deformaciones.

Entonces, partiendo de un conjunto de imágenes binarizadas que definen los cortes axiales de un modelo como las mostradas en la Figura 3.8, con distancia de separación Δz con respecto al eje longitudinal, se aplican los siguientes pasos:

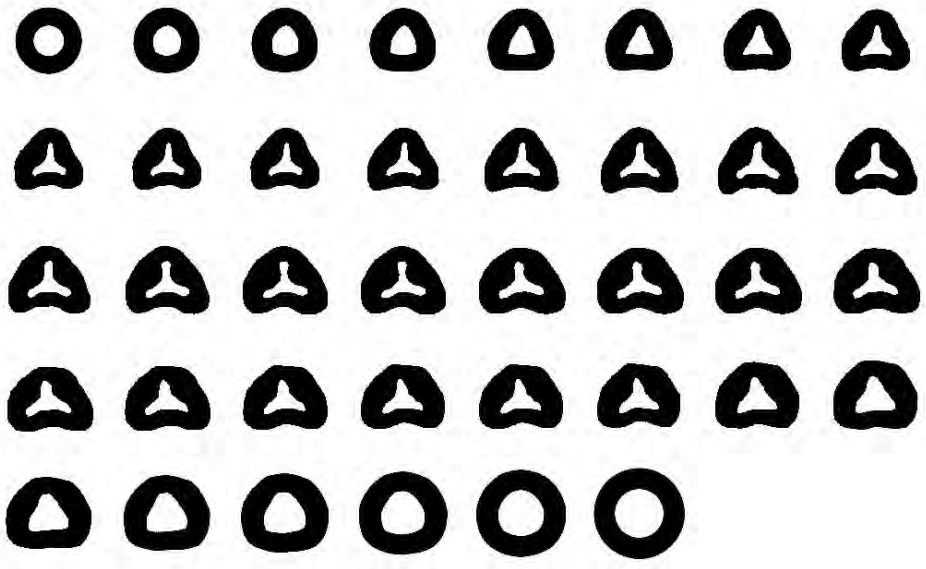


Figura 3.8. *Conjunto de cortes axiales que determinan un modelo de volumen tubular.*

1. Se obtienen un contorno interno y un contorno externo por cada imagen, usando algún método de extracción de contornos. En el proceso, la curva que describe cada contorno es un esqueleto de conectividad-8. El resultado es un conjunto de imágenes de tamaño $2S$ (Figura 3.9), donde S es el tamaño de la pila de imágenes inicial.

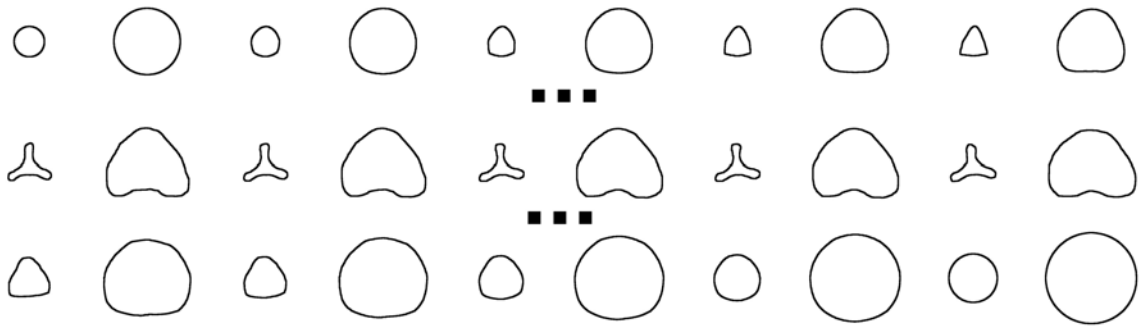


Figura 3.9. *Obtención de contornos internos y externos de un conjunto de imágenes que representan un modelo de volumen.*

2. Dependiendo de la resolución radial del modelo (capas), se obtienen contornos intermedios aplicando iterativamente algún método de interpolación. Por ejemplo, para las imágenes de la Figura 3.8, al obtener el esqueleto se tiene el contorno intermedio como se muestra en la Figura 3.10.

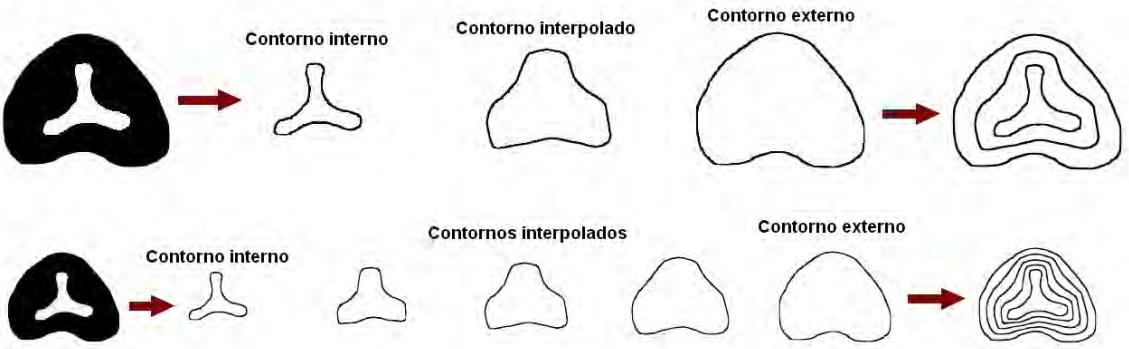


Figura 3.10. *Representación de contornos interpolados usando la operación morfológica de obtención del esqueleto.*

3. Se muestrea cada contorno de manera uniforme, de acuerdo con el Algoritmo 9. Para su ejecución, se parte de la identificación de un punto inicial en la curva. A continuación, se inicia un proceso de recorrido, en el cual, se van guardando las posiciones de los puntos que definen la curva en una lista. Estos puntos por ser resultado de un recorrido

continuo, se encuentran ordenados y representan las coordenadas x y y de cada punto en el plano (imagen) donde se encuentra contenido el contorno. Una vez obtenido el conjunto de puntos ordenados, se aplica un proceso de selección de puntos, en el cual, a partir de un intervalo de muestreo definido, se toman sólo los puntos que satifacen dicho intervalo.

Las variables empleadas para la definición del algoritmo son[1]:

- A: Conjunto binario con la información del contorno a muestrear (imagen de entrada).
- B y C: Conjuntos binarios temporales.
- m, n: m-filas y n-columnas de los conjuntos binarios A, B y C.
- I: Lista de puntos • ordenados.
- M: Lista de puntos muestreados.
- $|P_C|$: Número de puntos • en el conjunto A.
- $|M_C|$: Número de muestras del contorno a obtener.
- Δr: Intervalo de muestreo.
- p_0: Punto inicial del recorrido de coordenadas (x, y).
- p_{-1}: Punto anterior de coordenadas (x, y).
- p_i: Punto actual de coordenadas (x, y).
- p_{+1}: Punto siguiente de coordenadas (x, y).

Claramente, tanto el Algoritmo 9 como el Algoritmo 10 tienen una complejidad lineal $O(n)$. Al final de la ejecución se tiene que: B es igual al conjunto A original, C es el contorno muestreado y M es una lista de puntos muestreados ordenados.

El análisis de los vecinos para determinar el punto anterior y siguiente usa la lógica de la Figura 3.11.

El resultado de aplicar este algoritmo a un conjunto de contornos, es una lista ordenada de puntos en x y y. Sin embargo, para fines de reconstrucción 3D hace falta agregar una tercer componente, la profundidad (z). La tercer componente se calcula a partir del intervalo entre cada corte de las imágenes médicas originales. Así, todos los puntos muestreados que pertenecen a un mismo plano de corte deberán tener la misma componente de profundidad.

Hasta este paso se tiene un conjunto ordenado de puntos que definen muestras del modelo anatómico original. Sin embargo, por si solos son inútiles para fines de simulación y visualización, por tanto es necesaria la aplicación de una metodología que, a partir del origen de los puntos muestreados, se determinen los enlaces existentes entre ellos; definiendo así la geometría tridimensional del objeto.

[1]Con el fin de ofrecer una forma simple de representar los puntos o píxeles de análisis, se emplea el símbolo • para definir un píxel de interés y ∘ para representar píxeles que representan el fondo de la imagen binaria.

Algoritmo 9: Algoritmo de muestreo de contornos

```
1  begin
2      A = initContour(m, n);
3      B(m, n), C(m, n) ← ∅ ;
4      p_0 = p_i = p_{+1} = p_{-1} = (0, 0);
5      I, M ← ∅;
6      |P_C| =numActivePixels(A);
7      for i = [1 : m] do
8          if A(i, n/2) = • then
9              p_0 = (i, n/2);
10             push(p_0 → I);
11             break;
12         end
13         else
14             error();
15         end
16     end
17     findAndSortPixels(A,I);
18     Δr = (|P_C|−1)/|M_C|;
19     for i = [1 : |P_C| − 1], i+ = Δp, i ∈ ℕ do
20         C(I_i^x, I_i^y) = •;
21         push(I_i → M);
22     end
23 end
```

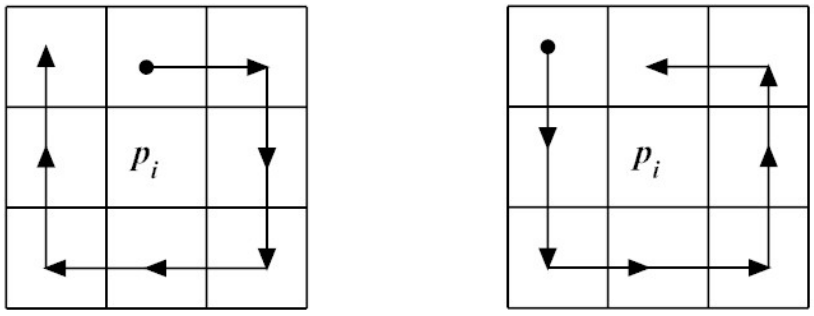


Figura 3.11. *Análisis de vecinos para encontrar puntos anteriores y siguientes en el algoritmo de muestreo. A la izquierda, recorrido para encontrar punto anterior, p_{-1}. A la derecha, recorrido para encontrar punto siguiente, p_{+1}.*

Algoritmo 10: findAndSortPixels(A,I): Busca y lista ordenadamente los puntos que conforman un contorno, tomando como punto de inicio del recorrido a p_0.

```
1  begin
2      p_i = p_0;
3      B(p_i^x, p_i^y) = •;
4      p_-1 =analyzeBackNeighbor(p_i, A);
5      B(p_-1^x, p_-1^y) = •;
6      p_+1 =analyzeNextNeighbor(p_i, A);
7      B(p_+1^x, p_+1^y) = •;
8      push(p_+1 → I);
9      for i = [1 : |P_C| − 2] do
10         p_-1 = p_i;
11         p_i = p_+1;
12         A(p_i^x, p_i^y) = ∘;
13         p_+1 =analyzeNextNeighbor(p_i, A);
14         B(p_+1^x, p_+1^y) = •;
15         push(p_+1 → I);
16     end
17 end
```

3.2.3. Reconstrucción

Sea el volumen anatómico mostrado en la Figura 3.12, con eje paralelo al eje Z. Si se subdivide el espacio que ocupa en hexaedros regulares, como el mostrado en la Figura 3.1, con resoluciones R_r (número de capas), R_θ (número de muestras por contorno) y $R_h = S - 1$ (con S igual al número de planos de corte), el número de vértices y tetraedros están dados por:

$$\begin{aligned} n_v &= R_\theta \cdot S \cdot (R_r + 1) \\ N_t &= 5 \cdot R_\theta \cdot R_r \cdot R_h \end{aligned}$$

El algoritmo 11 subdivide el espacio de volumen en un objeto tubular con espesor formado por tetraedros. Como con las geometrías básicas, se ejecutan los algoritmos 4 y 5 para completar el proceso. La función **loadVertices** carga la información de la nube de puntos obtenida en el proceso de muestreo y los guarda en una lista (**verticesList**). Es importante mencionar que el orden de cómo se encuentran los puntos en memoria debe coincidir con la formación usada con las geometrías básicas. De este modo, al ejecutar el muestreo sobre las imágenes se debe asegurar que primero se almacenan los anillos internos e ir avanzando hacia los anillos externos. La Figura 3.13 muestra el modelo computacional genérico empleado para este método.

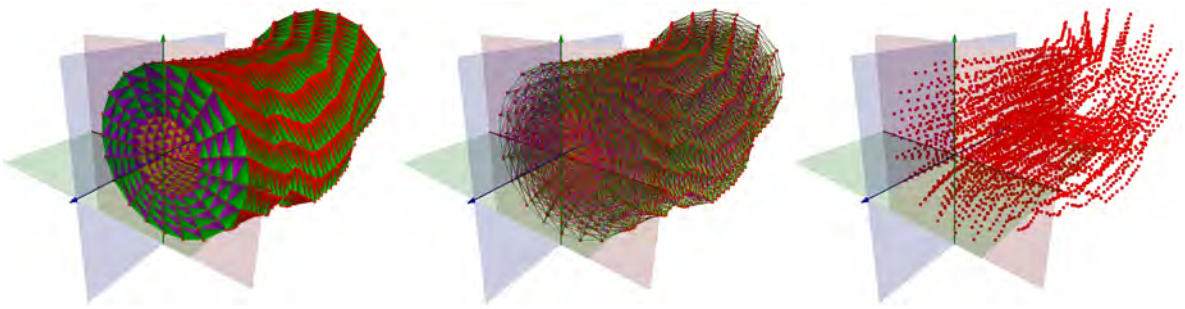


Figura 3.12. *Tetraedrización de un volumen anatómico.*

Algoritmo 11: Subdivisión de un objeto de volumen tubular.

```
1 begin
2     loadVertices(verticesList);
3     for layer = [0 : R_r − 1] do
4         createHexLayer(layer, close = true);
5     end
6     createTetrahedra();
7 end
```

Sección 3.3

Generación basada en superficies

En 2003, Müller y Teschner [48] propusieron un método de generación de mallas de volumen para ser empleadas en simulaciones basadas en física con aplicaciones médicas. El método consta de cuatro pasos y usa como base: una superficie y un espacio estructurado de volumen formado por tetraedros. Su aplicación es más general que sólo a modelos tubulares, sin embargo, depende totalmente de un modelo de superficie inicial.

Así, dada una superficie $\partial\Omega$ estructurada, triangular (Figura 3.14a), que representa la frontera del dominio finito Ω (el objeto), y que sirve como referencia para discretizar el interior de este último en un subconjunto de elementos de volumen, igualmente estructurados, pertenecientes a un espacio definido Ω^*, se ejecutan los siguientes pasos:

1. Se define el espacio Ω^{*h}, discretizado en una malla de hexaedros con forma conocida, generalmente una caja, que encierra a $\partial\Omega$. La resolución de la estructura de hexaedros inicial depende del usuario. Cada hexaedro (celda) se subdivide en cinco tetraedros, en una forma alternada, creando una nueva discretización Ω^{*t}.

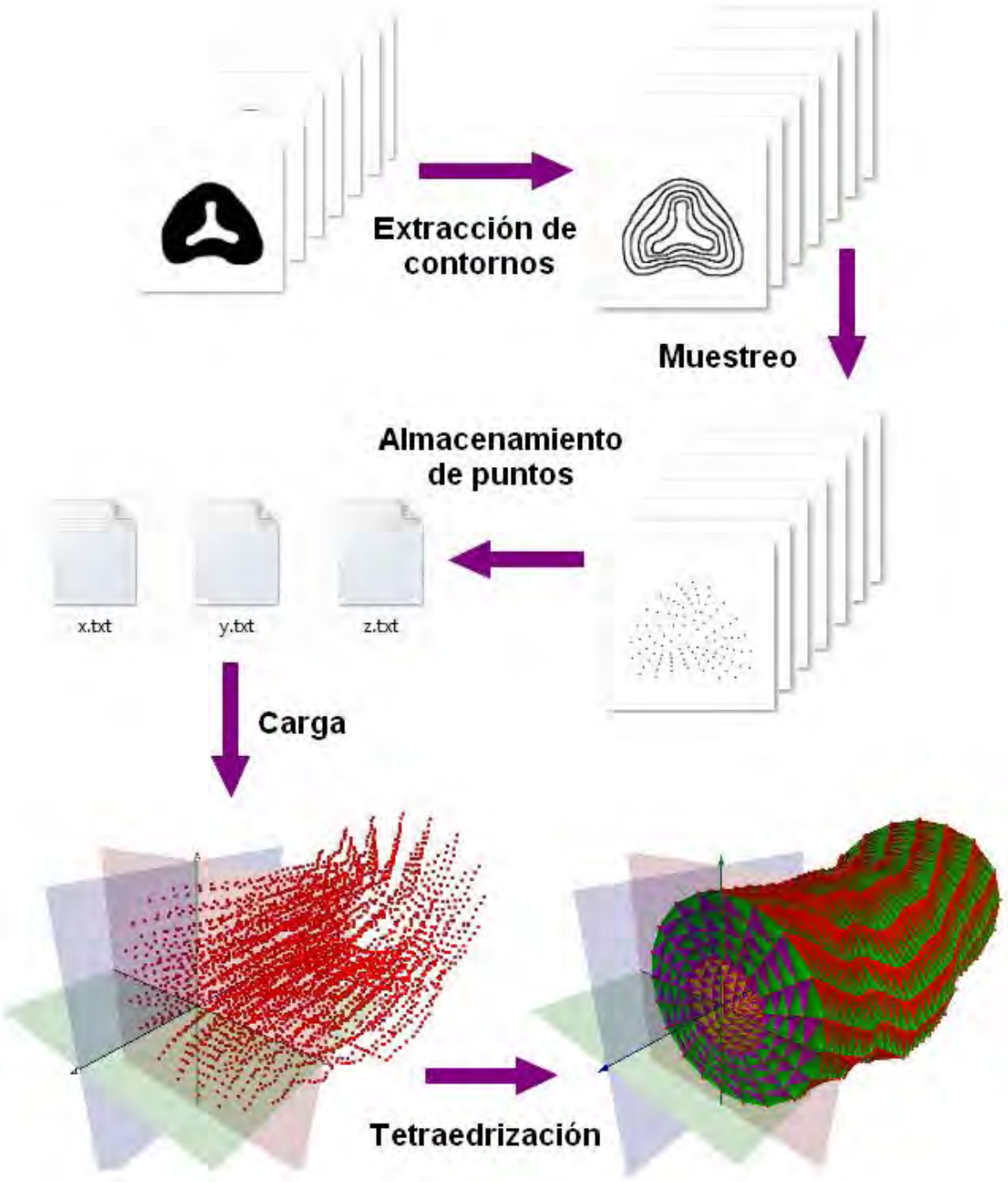


Figura 3.13. *Modelo de reconstrucción de un objeto tubular a partir de contornos.*

2. Se subdivide la malla de tetraedros de acuerdo a la superficie de triángulos, ejecutando los siguientes sub-pasos: 1) por cada vértice que pertenece a la malla triangular se genera un vértice en la malla de tetraedros Ω^{*t}, subdividiendo el tetraedro que contiene al vértice, en cuatro tetraedros más; 2) si un triángulo de la superficie $\partial\Omega$ intersecta con una arista de la malla de tetraedros, la arista se subdivide; y 3) si una arista de la malla de triángulos intersecta con una cara de un tetraedro, entonces la cara del tetraedro se subdivide. Si aristas y caras se subdividen, se generan nuevos tetraedros en el proceso. Sin embargo, en algunas ocasiones el número de tetraedros es grande, por lo que la subdivisión es opcional y depende de la aplicación. En el caso ideal, cuando la malla de superficie y la de tetraedros coinciden en su frontera, se tiene un proceso de subdivisión completo.

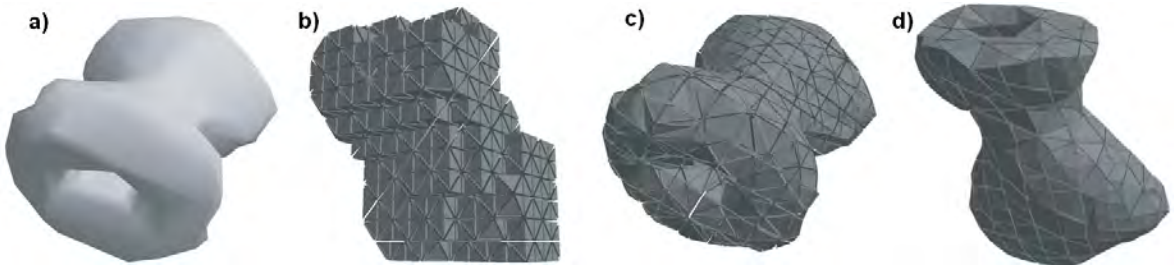


Figura 3.14. *Tetraedrización basada en superficies (Müller y Teschner [48], 2003).*

3. Todos los tetraedros localizados fuera de la superficie se eliminan (Figura 3.14b). Por lo tanto, por cada tetraedro se lanza un rayo desde el centro de éste hacia la superficie inicial $\partial\Omega$. Si el número de intersecciones del rayo con la superficie es par, el tetraedro se considera fuera y se elimina.

4. Finalmente, la malla de tetraedros se optimiza iterativamente (Figura 3.14(c-d)). Las posiciones de todos los vértices se optimizan con respecto a un conjunto de restricciones. Dado que se busca un radio de aspecto pequeño para todos los tetraedros, se pueden usar restricciones en las aristas en cuanto a longitud o bien su orientación si se busca una malla suavizada. Si se aplican todos los pasos de la subdivisión del paso dos, entonces todos los puntos en la frontera del objeto de tetraedros están contenidos en la malla de superficie. En otro caso se fuerza a que los vértices se muevan en dirección de la malla usando el enfoque de optimización.

La Figura 3.15 muestra un ejemplo de la tetraedrización de una superficie que representa un abdomen. Esta técnica es empleada por *NVIDIA PhysX* para la generación de modelos de volumen compatible con su motor de simulación. En la mayoría de sus versiones del SDK incorpora un software de generación de volúmenes a partir de superficies en diferentes formatos: Wavefront OBJ, Unreal PSK, EZ-Mesh, NxuStream XML, COLLADA DAE y NxuStream NXB.

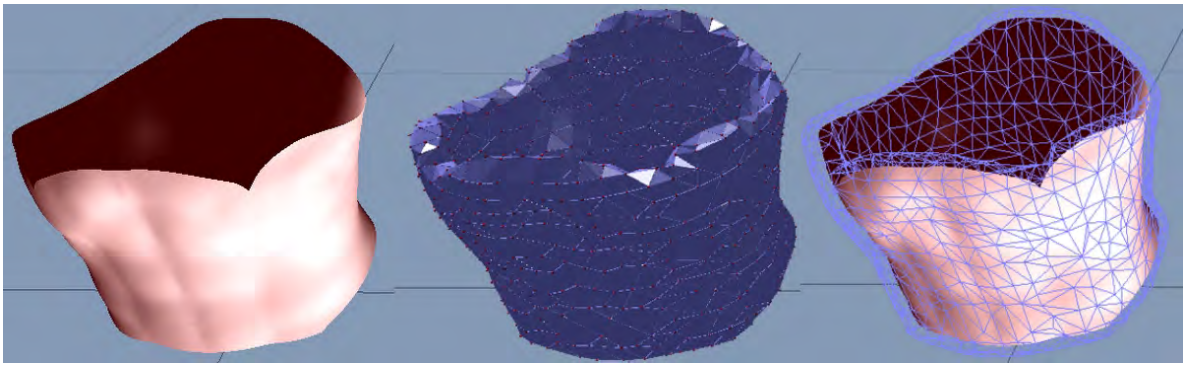


Figura 3.15. *Tetraedrización basada en una superficie de un modelo del abdomen usando PhysX Viewer de NVIDIA PhysX.*

Sección 3.4

Otros métodos de tetraedrización

El primer método descrito para la tetraedrización de objetos simples, tiene la desventaja de trabajar únicamente sobre estructuras planas y geometrías tubulares axiales; sin embargo, pueden obtenerse muy buenos modelos de volumen que tienen una geometría de capas traslapadas, tal como ciertas estructuras biológicas (ejemplo, el tejido). El segundo método requiere necesariamente una primera aproximación mediante una superficie, lo que implica primero generar a ésta. Por otra parte, es muy útil si lo que se requiere es únicamente simular estructuras con paredes delgadas o poco profundas (ejemplo, la piel y el recubrimiento de órganos).

Existen otros métodos más sofisticados, que por el alcance de este trabajo no se abordan. Dos enfoques muy empleados son:

- Métodos de tetraedrización por condición de Delaunay [87][72]. Es una extensión del método de triangulación en dos dimensiones, en la que se busca que un conjunto de vértices o puntos en el espacio conserven una relación geométrica uniforme. En el caso de triángulos, se lleva a cabo una prueba que consiste en verificar que tres puntos únicamente se encuentren contenidos dentro de una circunferencia que los inscribe. De igual forma, en tres dimensiones, cuatro puntos de un tetraedro únicamente podrán pertenecer a una esfera que los inscribe. A esta condición se le conoce como condición de Delaunay.

- Métodos de descomposición en octantes (*octrees*) [89]. Consiste inicialmente en la subdivisión de un dominio geométrico en sub-regiones regulares (cubos). A partir de un análisis de la frontera de un objeto y la subdivisión del dominio, se define una

primera aproximación del objeto, para después refinar cada celda usando un método de *octrees* y aproximarse de manera iterativa al volumen que ocupa el objeto.

Estos dos métodos generan mallas de tetraedros cubriendo una gran cantidad de casos, siendo los más beneficiados aquéllos que requieren tetraedros internos, como por ejemplo para la simulación de hueso, tejido muscular o representaciones del cerebro.

Asimismo como sucede con los motores de física descritos en el capítulo anterior, en el caso de la reconstrucción de volúmenes, existe software especializado para llevar a cabo esta tarea. En este trabajo se elaboró el software para la generación de mallas planas y mallas tubulares simples. *NVIDIA PhysX* posee su propio generador: *PhysX Viewer*. Y finalmente TetGen [80] es un software generador de mallas de tetraedros usando un triangulador de Delaunay 3D.

Sección 3.5

Formato para geometrías de volumen

En la actualidad, la mayoría de los sistemas que simulan comportamiento deformable, poseen su propio formato para la especificación de geometrías. Por ejemplo, el paquete de *NVIDIA PhysX* maneja un formato TET, en conjunto con un OBJ de *Wavefront* (Figura 3.16) para definir una superficie de mapeado. TetGen[2], además de permitir generar una malla de volumen, ofrece la posibilidad de usar su formato propio, que consiste en dos archivos: un archivo NODE donde se define la información de los vértices, y un archivo ELE, que contiene la información de cada tetraedro (Figura 3.17).

En general, un archivo de especificación de geometrías está compuesto por cuatro partes:

- **Encabezado:** Contiene información de las propiedades del modelo, tales como: número de vértices, número de caras (superficies), número de tetraedros (volúmenes), propiedades de visualización (número de coordenadas de textura, número de coordenadas normales, propiedades de color, etc.). El objetivo principal de esta parte es facilitar la reserva de memoria en un programa de carga. Algunos formatos simplemente emplean esta sección para identificar un modelo sin que el programa de carga use esta información.
- **Definición de vértices:** Contiene la lista de vértices del modelo. Generalmente se define una tripleta de valores para especificar las coordenadas (x, y, z) de cada punto.

[2]Software creado por Hang Si[80], Investigador del grupo de Cálculo Numérico y Cómputo Científico del WIAS (*Weierstrass Institute for Applied Analysis and Stochastics*), en Alemania.

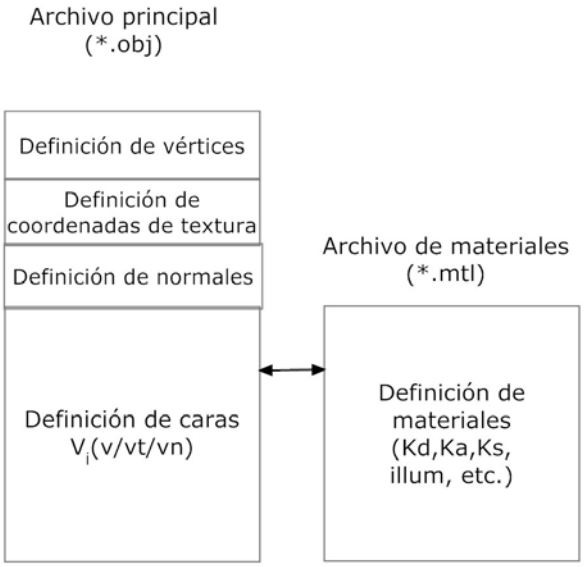


Figura 3.16. *Estructura del formato OBJ y su asociado MTL para definir materiales para visualización.*

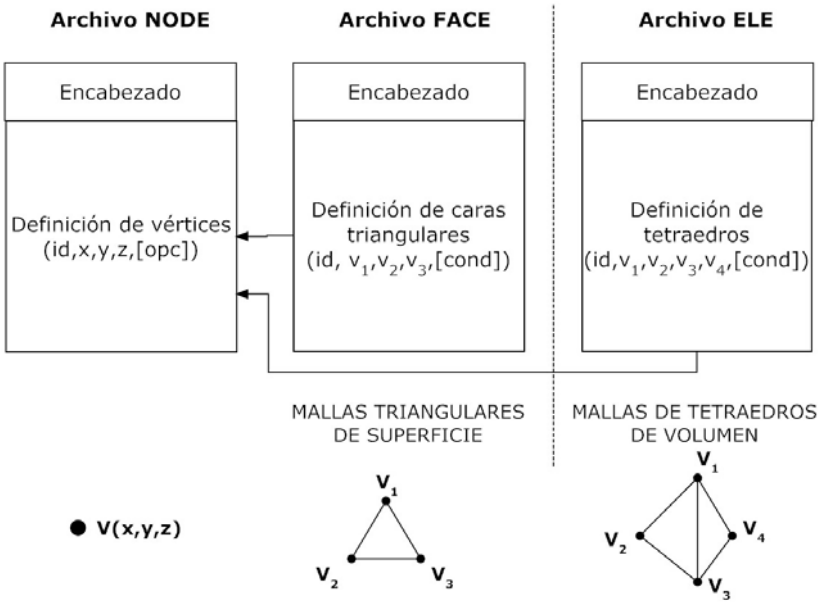


Figura 3.17. *Estructura de los formatos de TetGen (NODE, FACE, ELE) [80].*

- **Definición de geometrías:** Contiene la lista indexada que define un conjunto de geometrías espaciales. Para modelos de superficie, esta sección contiene una lista de índices a los vértices que definen triángulos o polígonos. Para modelos de volumen, esta sección contiene una lista de índices a vértices que definen tetraedros o poliedros.
- **Otras propiedades:** En esta parte se definen propiedades de visualización, propiedades de transformación, propiedades restrictivas (por ejemplo, índices a vértices estáticos, cuando se manejan mallas dinámicas), entre otras.

En este trabajo se propone la especificación de un formato muy sencillo, portable y fácil de leer mediante codificación ASCII, para la definición de geometrías de volumen formadas por puntos e índices a tetraedros. El formato es una combinación entre el formato *.obj y el formato académico de Steve Rotenberg (*.skin) de la Universidad de San Diego en California, EUA. Ambos formatos están diseñados para especificar únicamente superficies triangulares y poligonales. Con el formato propuesto se definen tetraedros empleando la misma sintaxis en un mismo archivo. Se ha decidido llamar a este formato, formato VOG (*Volume Object Geometry*, Geometría de Objeto de Volumen).

En una primera especificación, se definen 5 directivas y un símbolo de comentarios, mostrados en la Tabla 3.1, siguiendo la siguiente sintaxis:

```
# Comment
vert <#>
teth <#>
vertices {
          <x> <y> <z>
    ...
}
tetrahedra{
    <vertex1> <vertex2> <vertex3> <vertex 4>
    ...
}
attachments{
    <vertexi>
    ...
}
```

De esta forma, el cubo unitario de la Figura 3.1 tiene el siguiente archivo VOG asociado:

```
#models/cube.vog
#VOG File format (Volume Object Geometry)
#UNAM-Sergio Teodoro Vite-2011 : sergioteovit
#Statistics
vert 8
teth 5

vertices {
```

Directiva	Parámetro	Significado
#	Comentario	El texto delante de éste especifica un comentario
vert	numVerts	Número de vértices
teth	numTets	Número de tetraedros
vertices	$x\ y\ z$	Define un conjunto de vértices en coordenadas cartesianas (x, y, z).
tetrahedra	$v_1\ v_2\ v_3\ v_4$	Define un conjunto de índices a cuatro vértices que forman un tetraedro.
attachments	v	Define un conjunto de índices a vértices restringidos de movimiento para mallas dinámicas.

Tabla 3.1. Identificadores del formato VOG para la especificación de geometrías de volumen.

```
9       0.5   0.5   -0.5
10      -0.5   0.5   -0.5
11      -0.5   -0.5   -0.5
12      0.5   -0.5   -0.5
13      0.5   0.5   0.5
14      -0.5   0.5   0.5
15      -0.5   -0.5   0.5
16      0.5   -0.5   0.5
17  }
18
19  tetrahedra {
20      0   1   3   4
21      1   4   5   6
22      1   2   3   6
23      3   4   6   7
24      1   3   4   6
25  }
26
27  attachments {
28      0
29      1
30      2
31      3
32  }
```

Como puede observarse, el archivo define 4 restricciones, los vértices 0, 1, 2 y 3 son vértices fijos, que significa que en una simulación estos puntos no se moverían. Esta característica es importante cuando se simulan objetos deformables en escenarios basados en física, donde actúa la fuerza de gravedad. En este caso los vértices fijos permiten un mayor control sobre los demás puntos de la malla.

El cargador para este tipo de archivos consiste en dos partes: un analizador sintáctico o *parser* y un bloque de visualización con *OpenGL*. El analizador sintáctico tiene como entrada el archivo de texto en formato ASCII, que después se convierte a datos operables por el programa de visualización o de simulación. Para guardar los datos de se usan listas. A continuación se muestran algunos fragmentos de código del programa de carga y despliegue usando *OpenGL*.

Programa 3.1: Programa para cargar una malla de volumen con tetraedros en formato VOG.

```
void TetrahedralGeometries::LoadVOG(char *filename){
    FileReader *fReader = new FileReader();
    char buffer[255];
    Vector3D p;
    int value;

    if (fReader->Open(filename)){
        bool tokenOk = fReader->GetToken(buffer);

        while (tokenOk != NULL){
            switch(parseAttribute(buffer)){
                case ATTRIBUTE_VERTICES_NUM:
                    numVertices = fReader->GetInt();
                break;
                case ATTRIBUTE_TETRAHEDRA_NUM:
                    numTetrahedra = fReader->GetInt();
                break;
                case ATTRIBUTE_VERTICES:
                    fReader->GetToken(buffer); // Salta la primera llave
                    while (fReader->CheckCloseBranch() != true){
                        p.setx(fReader->GetFloat());
                        p.sety(fReader->GetFloat());
                        p.setz(fReader->GetFloat());
                        pList.push_back(p);
                    }
                break;
                case ATTRIBUTE_TETRAHEDRA:
                    fReader->GetToken(buffer); // Salta la primera llave
                    while (fReader->CheckCloseBranch() == false){
                        value = fReader->GetInt();
                        tList.push_back(value);
                    }
                break;
                case ATTRIBUTE_ATTACHMENTS:
                    fReader->GetToken(buffer); // Salta la primera llave
                    while (fReader->CheckCloseBranch() == false){
                        value = fReader->GetInt();
                        attachl.push_back(value);
                    }
                break;
                default:
                break;
```

```
            }
            tokenOk = fReader->GetToken(buffer);
        }
        fReader->Close();
    } else {
        printf("\n Unable to open VOG File...");
    }
}
```

Programa 3.2: Programa para dibujar un modelo de tetraedros.

```
void TetrahedralGeometries::DrawGeometry(){
    unsigned int i;

    glPointSize(5.0);
    glPushMatrix();
      glBegin(GL_POINTS);
        for (i=0; i<pList.size(); i++){
            glVertex3f(pList[i].x, pList[i].y, pList[i].z);
        }
      glEnd();
    glPopMatrix();
    glPointSize(1.0);

    glPushMatrix();
      for(i=0; i<tList.size(); i+=4){
          DrawTetrahedra(i);
      }
    glPopMatrix();
}
```

Programa 3.3: Programa para dibujar un tetraedro *i* de una lista de índices.

```
void TetrahedralGeometries::DrawTetrahedra(int i){
    Vector3D P[4];

    P[0] = pList[tList[i]];
    P[1] = pList[tList[i+1]];
    P[2] = pList[tList[i+2]];
    P[3] = pList[tList[i+3]];

    glBegin(GL_TRIANGLES);
      glColor3f(1.0f, 0.0, 0.0);
      glVertex3f(P[0].x, P[0].y, P[0].z);
      glVertex3f(P[1].x, P[1].y, P[1].z);
      glVertex3f(P[2].x, P[2].y, P[2].z);

      glColor3f(0.0f, 1.0, 0.0);
      glVertex3f(P[0].x, P[0].y, P[0].z);
      glVertex3f(P[1].x, P[1].y, P[1].z);
      glVertex3f(P[3].x, P[3].y, P[3].z);
```

```
19
20          glColor3f(1.0f, 1.0, 0.0);
21          glVertex3f(P[0].x, P[0].y, P[0].z);
22          glVertex3f(P[2].x, P[2].y, P[2].z);
23          glVertex3f(P[3].x, P[3].y, P[3].z);
24
25          glColor3f(0.0f, 0.0, 1.0);
26          glVertex3f(P[1].x, P[1].y, P[1].z);
27          glVertex3f(P[2].x, P[2].y, P[2].z);
28          glVertex3f(P[3].x, P[3].y, P[3].z);
29
30      glEnd();
31  }
```

Sección 3.6

Caso de estudio: Generación de un volumen del conducto uretral y próstata.

La uretra masculina es un conducto músculo-membranoso que comunica la vejiga urinaria con el exterior. Su longitud y calibre varia de un individuo a otro. En promedio, el diámetro de las porciones que la conforman varía entre 7 a 15 mm. Los instrumentos de exploración que se emplean en urología, tienen diámetro máximo de 10 mm, lo que les permite introducirse en la mayoría de las uretras [6, p. 19].

La uretra se divide en tres secciones[39, p. 425] (Figura 3.18):

- **Uretra Prostática:** Atraviesa la próstata en unos 3 cm. Es una sección ligeramente dilatada y presenta una ligera curva cóncava. La sección transversal tiene forma de herradura. Dentro de la próstata, la cresta uretral está engrosada en forma de colina, por lo que a esta zona se le conoce como *veru montanum*. A los lados de este último se localizan los orificios que son la desembocadura de la mayoría de los conductos prostáticos[6, p. 20].
- **Uretra membranosa:** Se encuentra rodeada por el esfínter uretral estriado, entre la salida de la uretra de la próstata y su entrada en el cuerpo esponjoso de la uretra.
- **Uretra esponjosa:** Compone el tubo esponjoso del conducto uretral que desemboca en el glande a través del orificio uretral externo. Poco antes de desembocar, el tubo tiene forma dilatada en forma de fosa navicular (lat. *navis=barco*).

La próstata es una glándula masculina de secreción externa, responsable del 40% del líquido seminal. Se encuentra situada alrededor de la uretra posterior[90, p. 298] y unida al

cuello de la vejiga urinaria. Tiene forma de cono aplastado de delante a atrás y cuya base está dirigida hacia arriba. Su tamaño promedio es de 4 cm de ancho, 3 cm de largo y 2 cm de espesor.

Anatómicamente, la próstata se divide en cuatro zonas[6, p. 27] (Figura 3.18):

- La *zona periférica*: Ocupa la mayor cantidad de volumen y se situa en la parte posterior y parcialmente a los lados del conducto uretral.
- La *zona central*: Se situa por encima de la zona periférica y es la unión casi completa con el cuello vesical; únicamente separada por la zona de transición con el conducto uretral posterior.
- La *zona de transición*: Es una zona pequeña que rodea el conducto uretral y se une con el cuello vesical. A su vez se encuentra rodeada por la zona central. Entre esta zona y la zona periférica se localiza el veru montanum.
- La *zona anterior*: Se localiza en la porción anterior de la próstata. Es la segunda en tamaño después de la periferia y se extiende a los lados en forma de una capa delgada que constituye la cápsula prostática.

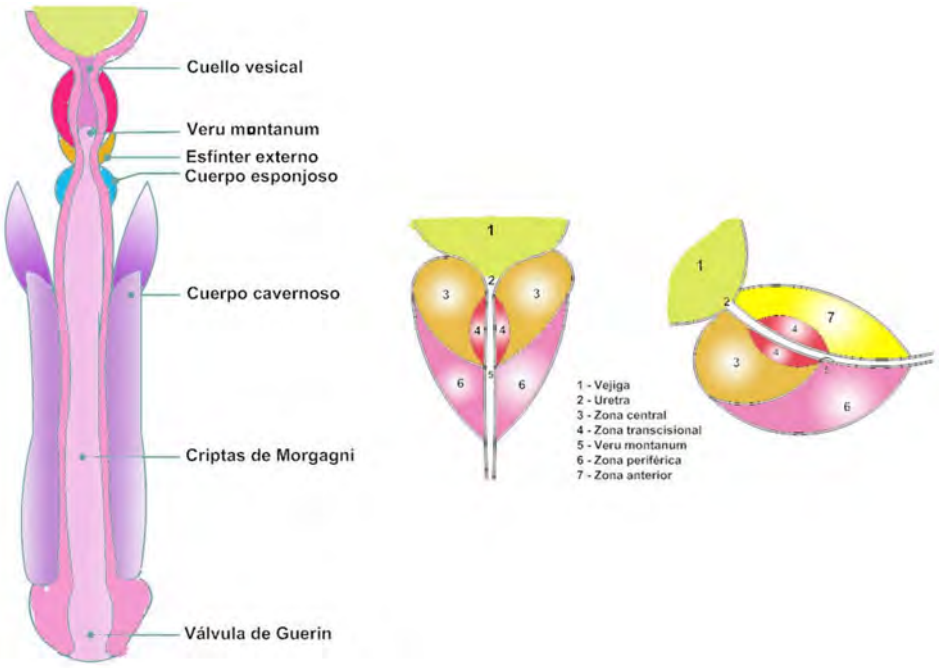


Figura 3.18. *A la izquierda, vista longitudinal de la Uretra masculina[6, p. 21]. A la derecha, anatomía de la próstata[6, p. 27]*

La anatomía de la uretra puede aproximarse mediante un tubo cilíndrico con las dimensiones correctas. Por tanto puede aplicarse fácilmente el método de generación de

mallas de volumen tubulares, descrito en la Sección 1 de este capítulo. De esta forma pueden controlarse el radio interno y el radio externo (definiendo implícitamente un espesor del modelo), un número de capas para definir regiones y una longitud variable. Un ejemplo de una malla de tetraedros de la uretra se muestra en la Figura 3.19, donde se ha aplicado adicionalmente una textura 3D y propiedades de deformación con el motor de física de *NVIDIA PhysX*.

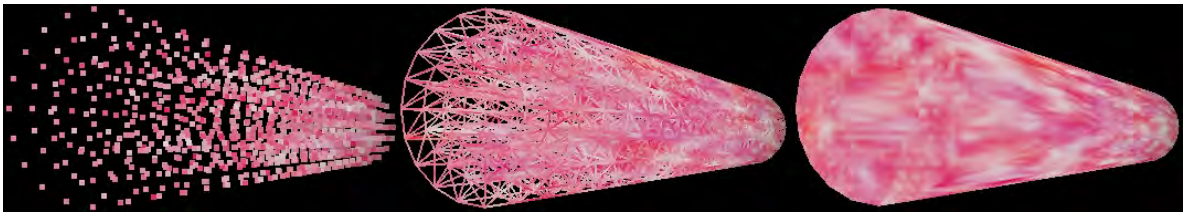


Figura 3.19. *Modelo de tetraedros de una uretra reconstruida mediante el método de geometrías tubulares.*

Para el caso del modelo de la próstata, se hace necesario una reconstrucción a partir de imágenes, que generalmente se obtienen a partir de una adquisición por ultrasonido transuretral *(Arámbula [4], Padilla [62])*. Estas imágenes se someten a un proceso de segmentación como se describió en la Figura 3.7 para obtener una forma binarizada del objeto. Y dado que generalmente se tienen pocas imágenes sobre el eje transversal, se requiere además aplicar un método para generar planos intermedios. En imágenes, una técnica comúnmente empleada es la interpolación morfológica, consistente en la aplicación de *operaciones binarias*[3] sobre un par de imágenes.

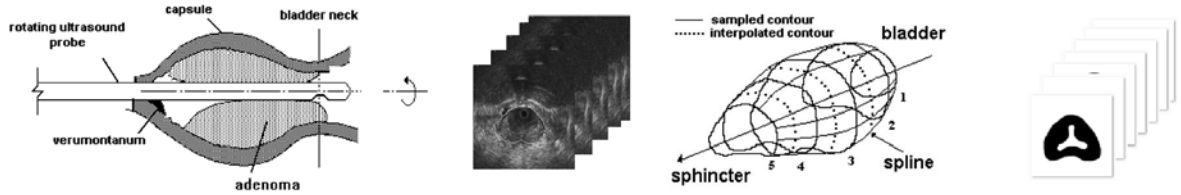


Figura 3.20. *Pre-procesamiento de imágenes de ultrasonido transuretral de la próstata (Padilla [62]) para la obtención de contornos. De izquierda a derecha, se obtienen 5 imágenes transuretrales de dimensión* 512×512*, las cuales se segmentan mediante una técnica de contornos activos (Arámbula[4]).*

La interpolación morfológica parte de dos conjuntos A y B (imágenes binarias) para generar un tercero, C, cuyo contenido es la interpolación de A y B. Se basa en el concepto de

[3]En imágenes, las operaciones binarias son aquellas que operan sobre representaciones de ceros y unos (blanco y negro). Entre las principales operaciones binarias se encuentran: Dilatación $\oplus$, Erosión $\ominus$, Apertura $\circ$, Cierre $\bullet$, Adelgazamiento $\otimes$, entre otros.[28, cap. 9]

media morfológica, la cual se define por la ecuación 3.1.

$$C = M(A, B) = \bigcup_{\forall \lambda} \{(A \oplus \lambda B) \cap (A \ominus \lambda B)\} \tag{3.1}$$

Esta ecuación se lee como: la interpolación morfológica de los conjuntos A y B es igual a la unión de todas las intersecciones resultantes de dilatar y erosionar el conjunto A por el conjunto B (elemento estructurante) en un factor λ[4].

La Figura 3.20 muestra el pre-procesamiento para la reconstrucción de un modelo de la próstata. La Figura 3.8, usada en la sección de muestreo, corresponde a la pila de imágenes de entrada al algoritmo de reconstrucción tras el pre-procesamiento. El resultado final es un modelo de tetraedros al cual se le pueden asignar propiedades de textura (como se mostró con el modelo de la uretra) y propiedades físicas (Figura 3.21).

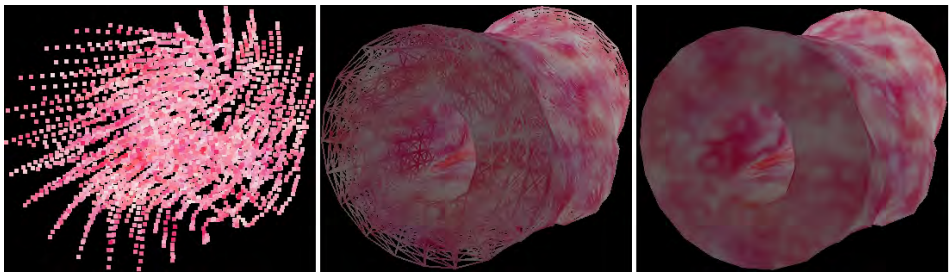


Figura 3.21. *Modelo de tetraedros de una próstata reconstruida mediante la técnica de iso-superficies.*

Es importante notar que al modelo de la Figura 3.21 se han agregado dos secciones adicionales, para crear una zona de acople entre el modelo de la uretra y la próstata, así como una zona de acople entre el modelo de la próstata y el cuello de la vejiga. Finalmente, la Figura 3.22 muestra los modelos incorporados en un simulador de cirugía de la próstata, descrito en el capítulo anterior como caso de estudio.

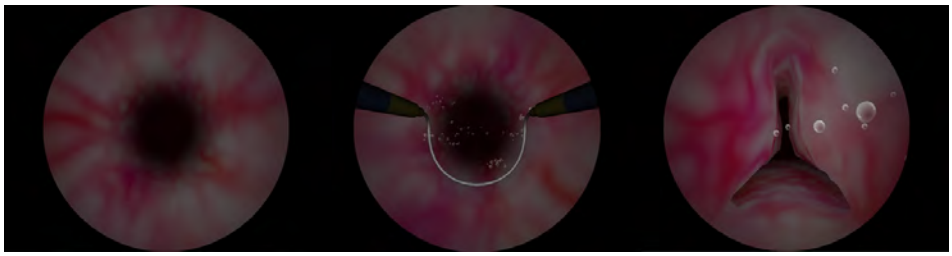


Figura 3.22. *Modelos del conducto uretral y próstata en un simulador de cirugía virtual.*

[4]Para mayor información, consulte Soriano[72, cap. 2] y Iwanowski[34, p. 50-51]

CAPÍTULO 4

Modelado Computacional de Cortes de Tejido Blando

En los capítulos anteriores, presentamos una idea general de la forma de representar objetos o estructuras anatómicas deformables mediante modelos, desde su construcción geométrica, hasta la asignación de propiedades físicas y visuales para proporcionarles detalles de realismo. Sin embargo, hemos tocado poco el tema de la operabilidad sobre éstos. En este capítulo abordaremos los métodos computacionales para la simulación de una acción en específico, presente en prácticamente casi todas las intervenciones quirúrgicas: el corte de tejidos blandos.

Desde el punto de vista médico, *Martínez et. al.*[42] mencionan que dos elementos que generalmente se requieren en la práctica quirúrgica son: el pleno conocimiento de la anatomía a tratar y el dominio de la técnica a aplicar. De tal forma que un cirujano o aprendiz debe ser capaz de ejecutar movimientos sistemáticos y ordenados sobre tejidos y órganos, incidiéndolos, separándolos, extirpándolos o reparándolos, de forma que no lesione sus componentes o estructuras. Partiendo de esta idea, es posible crear un modelo computacional capaz de involucrar la mayor cantidad de elementos presentes en una acción quirúrgica; tanto físicos, médicos y matemáticos, con el fin de crear situaciones en condiciones simuladas de procedimientos de corte, sobre estructuras igualmente simuladas.

La elección de uno u otro modelo de simulación computacional para la operabilidad de estructuras, dependerá en muchas circunstancias del tipo de aplicación y del caso de estudio. Así por ejemplo, si lo que se requiere es incidir sobre tejido, necesariamente requeriremos de una técnica que divida una malla geométrica. O bien, si lo que se busca es extirpar un fragmento de tejido, necesitaremos de una técnica que permita eliminar componentes de una malla. Éste y otros tipos de casos se abordan a continuación, no sin antes introducir los elementos médicos que nos motivan a la búsqueda de estas técnicas.

En general, las técnicas quirúrgicas constan de cinco tiempos fundamentales (Figura 4.1), que se ejecutan de manera combinada o secuencial durante una intervención (*Martínez [42], Archundia [5]*): incisión, hemostasia, exposición, disección y sutura.

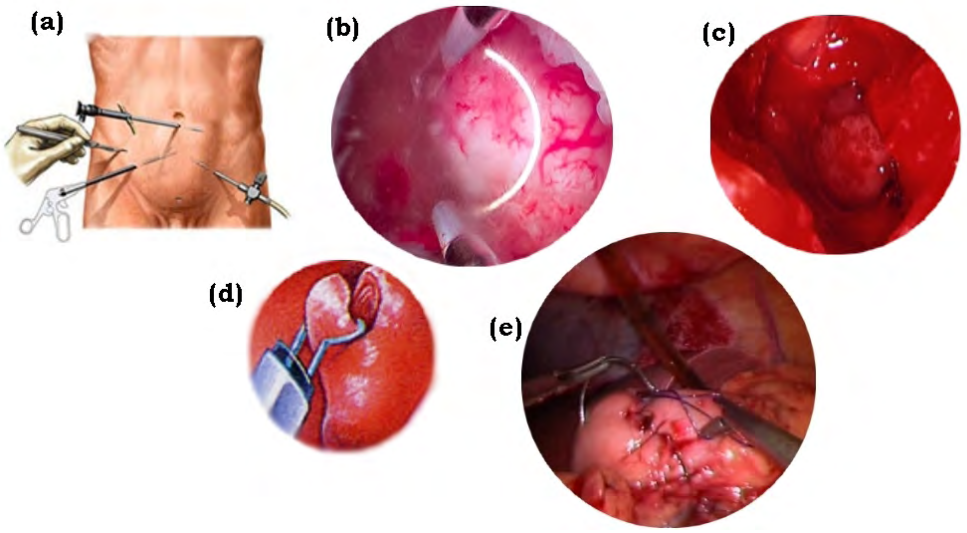


Figura 4.1. *Tiempos en cirugía de mínima invasión. a) Insición. b) Coagulación usando radiofrecuencia. c) Obstrucción por sangrado. d) Disección de tejido. e) Sutura.*

1. *Incisión*. Se emplean instrumentos de corte, desde bisturíes y tijeras, hasta instrumentos de electrofulguración, con el fin de abordar de manera metódica y controlada a cierto órgano o tejido. En el caso específico de las técnicas de mínima invasión, como la cirugía endoscópica, torascópica, laparoscópica, artroscópica y culdoscópica, las incisiones en piel son muy pequeñas o nulas.

2. *Hemostasia o coagulación*. Es el procedimiento de eliminar de manera temporal o definitiva una hemorragía. Los métodos de hemostasia son muy variados y dependen mucho de la aplicación; así pueden ejecutarse desde simples pinzamientos hasta complicados procedimientos con radiofrecuencia y láser.

3. *Exposición*. Consiste en la separación, aspiración o tracción de elementos obstructivos durante un procedimiento, garantizando el mayor campo visual y acceso al objetivo, para cortar, reparar o extirpar. La separación se logra mediante retractores manuales o automáticos. La aspiración se lleva a cabo mediante cánulas de aspiración o jeringas, para irrigar sangre o residuos orgánicos.

4. *Disección*. Consiste en la liberación de estructuras anatómicas del tejido conjuntivo que las rodea para llevar a cabo un tratamiento reconstructivo o resectivo.

5. *Sutura*. Consiste en la fijación o empalme de los tejidos con la finalidad de acelerar el proceso de cicatrización y garantizando la recuperación del área tratada.

Sección 4.1

Cortes de tejido

En la actualidad, existen metodologías estándares bien definidas para efectuar una acción de corte sobre un tejido, dependiendo de su tipo y disposición espacial. En este trabajo se abordarán cuatro tipos: incisión, disección, retracción o extirpación, y electrocauterización. A continuación se describe de manera general en qué consiste cada una de éstas[1]. Sobre tejidos blandos, estas técnicas se emplean para eliminar tumores, melanomas y sarcomas (Ferraina[21], 2002).

4.1.1. Incisión

Archundia [5] dice: del latín *incidere*, cortar, se le llama incisión a la sección metódica de las partes blandas con instrumentos cortantes para abordar un órgano o tejido. El trazo en una incisión se selecciona en función de la circunstancia. Así, se puede clasificar un corte con respecto a su trazo como: recto, curvo, mixto, semicircular y fusiforme. De acuerdo con la dirección del corte, en relación con el eje de la región a tratar, se clasifican en: longitudinal, transversal y diagonal.

El proceso de incisión en el caso de la cirugía videoasistida y cirugía endovascular, consiste en cortes múltiples y muy pequeños, en comparación con una incisión convencional sobre piel o recubrimientos. En cirugías de mínima invasión, se introducen instrumentos llamados *trocares*[2] de manera temporal, en ciertos puntos definidos (de radios que van desde los 10 mm hasta los 30 mm), y que representan los puertos de acceso a otras herramientas de corte y drenaje. En el caso de la cirugía endovascular se elimina la incisión clásica y se realiza una punción de menos de 5 mm para introducir catéteres endovasculares.

4.1.2. Disección

La disección es una técnica empleada para dividir un tejido con respecto a un plano de corte y usando para ello herramientas como hojas con filo de precisión o bisturíes. El término generalmente se aplica también para la acción de separar estructuras vegetales y animales.

[1] Para una mayor referencia sobre las técnicas de cortes, puede encontrar mayor información en [42] y [5].

[2] Trocar: varilla afilada, puntiaguda, que se adapta a un tubo, para perforar la piel y la pared de una cavidad o canal en el cuerpo, con el objetivo de aspirar líquidos, instilar un medicamento o solución, o para guiar la colocación de un catéter blando. *Diccionario Mosby de Medicina, enfermería y ciencias de la salud* [46].

4.1.3. Retracción

Consiste en la extirpación de un tejido o parte de éste usando herramientas de retracción, como pinzas. Suele ser la técnica usada para la eliminación de tejido con daños permanentes y de fácil desprendimiento, así como también para toma de biopsias.

4.1.4. Electrocauterización

Este tipo de corte tiene como base la técnica del cauterio[3], que actualmente se lleva a cabo mediante electrocauterización y electrofulguración. El campo de estudio de este tipo de técnicas se le conoce como Electrocirugía y es muy popular para procedimientos de mínima invasión.

La electrocirugía emplea energía de radiofrecuencia (RF) para cortar y coagular tejido. En este sentido se pueden diferenciar dos estados, el *estado de corte* en el que el tejido es literalmente vaporizado en cuanto se le hace pasar un electrodo a través de éste, mientras que las paredes en la región quedan selladas evitando el sangrado excesivo, y el *estado de coagulación* en el cual se hace incidir una descarga de radiofrecuencia en el momento en que el electrodo y el tejido se encuentran a cierta distancia para provocar la fulguración[4].

Asimismo existen dos modos de operación: el monopolar y el bipolar. El modo monopolar consiste en la emisión de ondas de radiofrecuencia desde un electrodo de alta densidad llamado *electrodo activo*, a un electrodo superficial llamado *electrodo inactivo* o *pasivo* (Figura 4.2A). El modo bipolar consta de un circuito por el cual la energía es descargada en los polos (pasivo y activo) y pasa a través del tejido (Figura 4.2B); empleada para cirugías donde se requieren cortes muy pequeños y de gran precisión.

Las técnicas en electrocirugía a su vez se diferencían en cuatro categorías (*Bussiere[12], 1997*):

1. Incisión. Se emplea una hoja delgada como electrodo y se realiza un corte puro, dejando una necrosis lateral, determinada en gran medida por la velocidad del movimiento (Figura 4.3a).

2. Resección. Se emplea un electrodo en forma de lazo para remover parte del tejido, dejando un zurco con la forma del lazo y dejando la pared al mismo tiempo coagulada (Figura 4.3b). Esta técnica funciona tanto en modo monopolar como bipolar.

[3]El cauterio es un instrumento o agente que provoca la cuagulación y destrucción de los tejidos por medio de calor o agentes caústicos. *Diccionario Mosby de Medicina, enfermería y ciencias de la salud* [46].

[4]*Del lat. fulgurare*, relampaguear. Accidente causado por un rayo. *Diccionario de la Real Academia Española, 2011.*

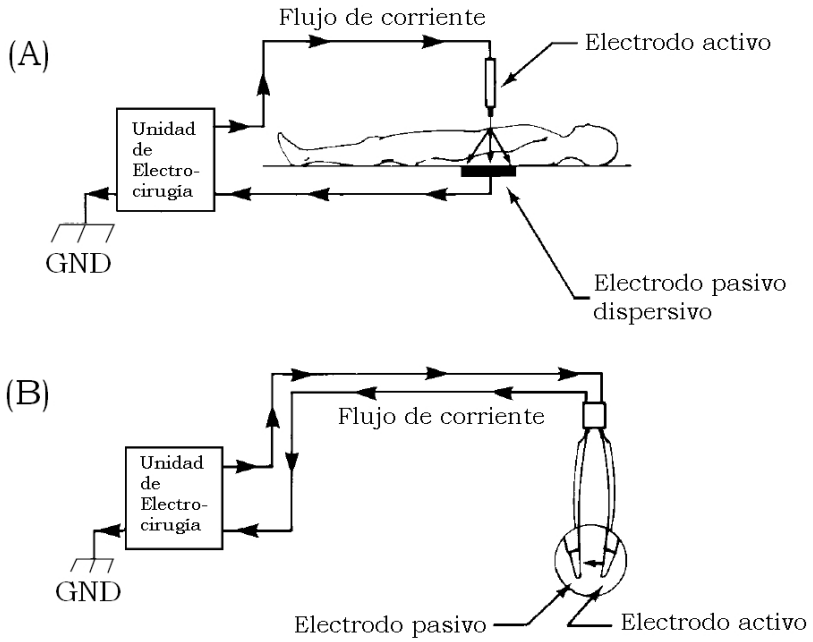


Figura 4.2. *Modos de operación de dispositivos en electrocirugía. A) Modo monopolar. B) Modo bipolar. (Bussiere[12], 1997)*

3. Fulguración. Se emplea una punta o un lazo para eliminar hemorragias precisas sobre la superficie del tejido (Figura 4.3c). Al igual que la anterior, la técnica funciona tanto para modo monopolar como bipolar.

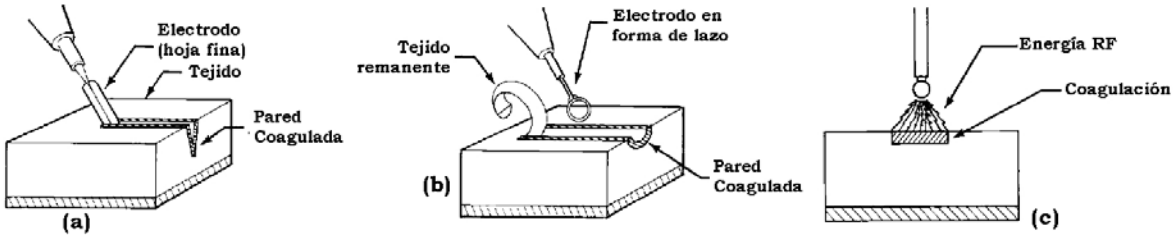


Figura 4.3. *Técnicas de corte en electrocirugía. a) Técnica de incisión. b) Técnica de resección. c) Técnica de fulguración. (Bussiere, 1997 [12])*

Sección 4.2

Instrumental quirúrgico para cortes de tejido

Como en muchas otras áreas y campos del conocimiento, en la práctica quirúrgica se emplean herramientas manuales y automáticas para llevar a cabo tareas de corte, coagulación, tracción y sutura, entre otras. Sin embargo, podemos distinguir las herramientas utilizadas en una cirugía de mínima invasión y aquéllas usadas para realizar cortes en la cirugía convencional.

Dada la gran cantidad de herramientas existentes, a continuación nos concentraremos sólo en aquellos instrumentos empleados para llevar a cabo las acciones de corte descritos en la sección anterior.

4.2.1. El Bisturí

Es posiblemente la herramienta más utilizada durante la incisión y acciones de disección. Se compone de un mango y una hoja de corte desechable; sus tamaños varían dependiendo del tamaño del corte, y pueden ser de corte fino (números 10 al 15) o corte grueso (números del 20 al 25).

El manejo del bisturí igualmente depende del tipo de incisión, así se pueden realizar cortes superficiales y extensos, cortes regulares y profundos, o cortes pequeños (Figura 4.4). En todos los casos, el bisturí debe iniciar formando una perpendicular al plano de incisión.

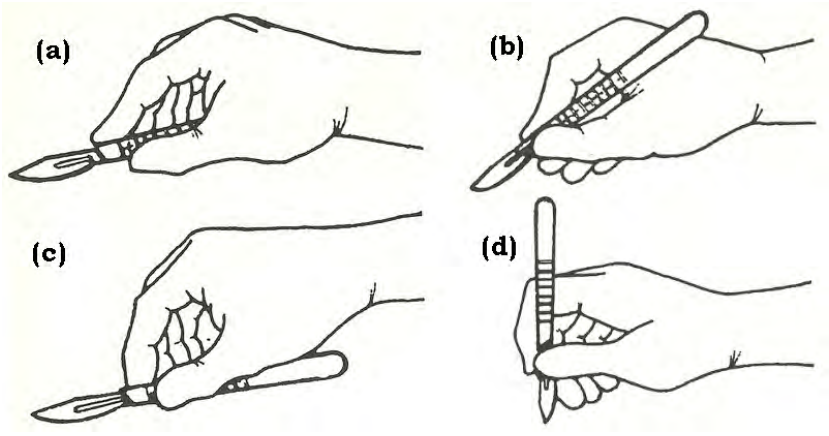


Figura 4.4. *Modos de uso con Bisturí para cortes. a) y c) Cortes superficiales y extensos. b) Cortes regulares y profundos. d) Cortes pequeños. (Martínez [42])*

4.2.2. Tijeras

Dependiendo del tipo de corte, existen una gran variedad de tijeras; sin embargo, las más empleadas son las rectas, usadas para cortar hilo y otros materiales de curación, y las curvas, empleadas para el corte de tejidos. De este último tipo las *tijeras Mayo* son empleadas para seccionar tejidos resistentes, las *tijeras Metzembaum* empleadas para corte de tejidos finos y precisos, y las *tijeras de Iris*[5], para cortes muy delicados (Figura 4.5).

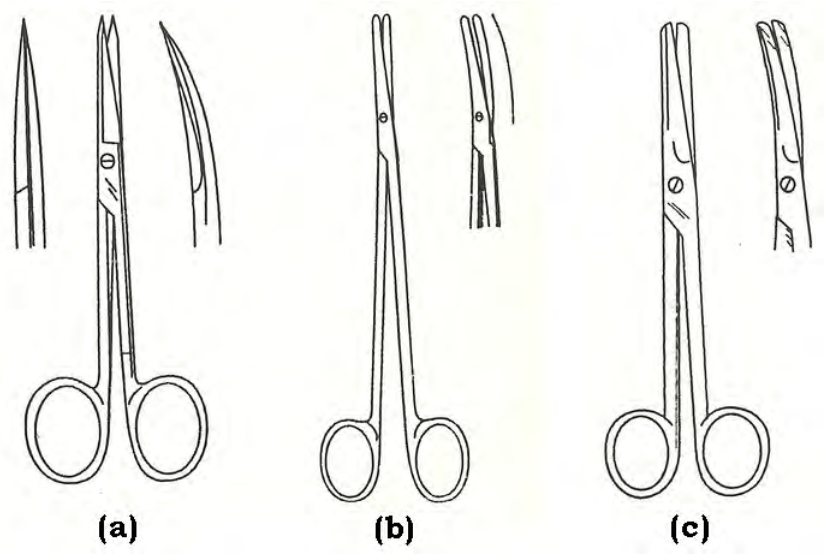


Figura 4.5. *Tipos comunes de tijeras en cirugía. a) Tijera de Iris (cortes muy finos). b) Tijera Metzenbaum (cortes finos y precisos). c) Tijera Mayo (cortes en tejidos resistentes). Martínez Dubois, Cirugía: Bases del Conocimiento quirúrgico, 2001 [42].*

4.2.3. Pinzas

Las pinzas, en el sentido estricto no son una herramienta de corte, sino una herramienta para el control de hemorragias y de tracción. Sin embargo, en algunas aplicaciones médicas, como lo es la toma de biopsias, las pinzas juegan un papel importante en la separación de tejidos. En cirugías de mínima invasión, las pinzas sirven en la exposición del objetivo, estirando y en algunos casos desgarrando tejidos obstructivos. Para este própósito, en cirugía endoscópica, se emplean variantes de las pinzas convencionales, muy pequeñas, y generalmente situadas en uno de los extremos de una varilla que se acopla en sistemas específicos para estas aplicaciones, como un resectoscopio. La apertura de las

[5]Las tijeras de Iris deben su nombre a que inicialmente se emplearon para cirugías oftalmológicas del iris; sin embargo, actualmente han sido empleadas para otros tratamientos (*Archundia [5]*).

pinzas se lleva a cabo por un mecanismo situado al otro extremo, similar al mecanismo usado en las tijeras (Figura 4.6).

4.2.4. El resectoscopio

El resectoscopio (Figura 4.6) es un instrumento electroquirúrgico usado para la remoción de tejidos blandos en las especialidades de urología y ginecología. Así, por ejemplo, se usa en cirugías de próstata, útero, vejiga y uretra (*Smith et. al. [70]*). Se trata de un instrumento endoscópico que combina una cámara y un kit de instrumentos de corte, coagulación y aspiración, efectuadas sobre una incisión o a través de conductos naturales. Posee además una lente que permite una amplia visualización del área a tratar. Adjunta a ella se encuentra un lazo de resección, que se activa mediante corriente eléctrica. Cuando el cirujano activa el modo de corte, éste puede obtener partes de tejido con tan solo deslizar el asa sobre la región a remover. Al mismo tiempo, el asa cauteriza el tejido evitando el sangrado. En caso de sangrado, se activa el modo de coagulación y se produce el efecto de fulguración.

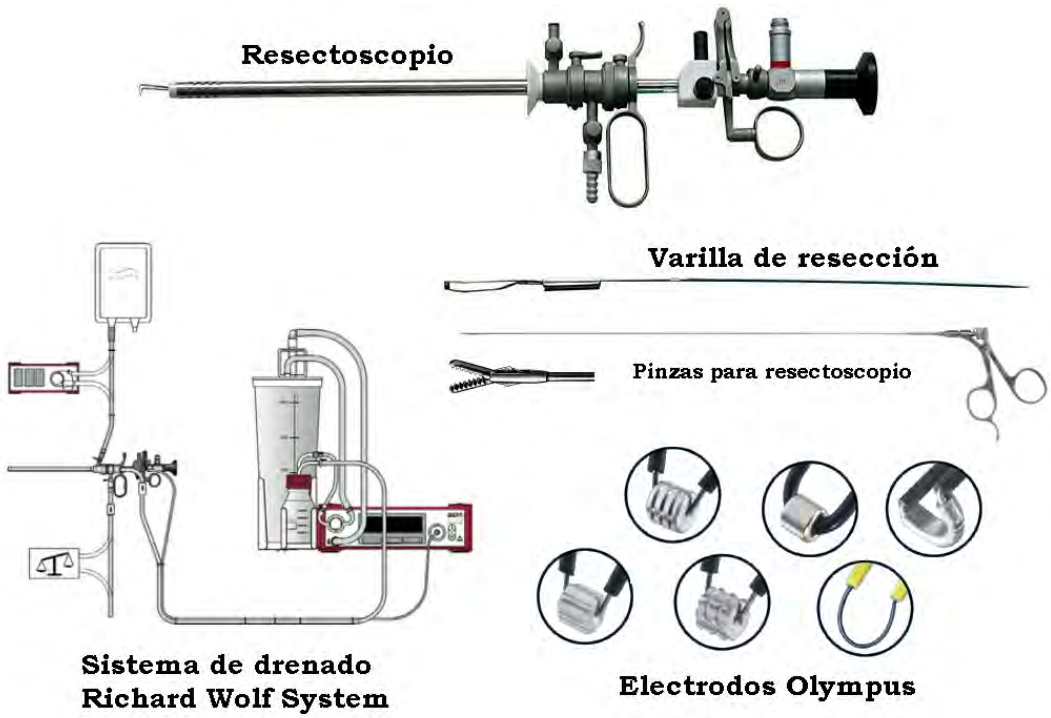


Figura 4.6. *Resectoscopio y herramientas de corte, coagulación y aspiración.*

Sección 4.3

Métodos de simulación computacional

La posibilidad de interactuar y transformar estructuras geométricas tridimensionales, es una de las características fundamentales para lograr la inmersión de un usuario en un entorno virtual. Si a ésto agregamos la necesidad de funcionar en tiempo real, surge un problema del procesamiento debido a la complejidad de los algoritmos y de las acciones a resolver. Es por ello que muchos de los métodos para la simulación de acciones como corte sobre mallas, tengan asociadas técnicas heurísticas; esto es, metodologías que basan su principio de funcionamiento en abstracciones geométricas que no necesariamente se apegan a lo que sucede en el mundo real.

Algunas de estas técnicas tienen un amplio espectro de aplicaciones, incluyendo la simulación de intervenciones quirúrgicas, visualización interactiva de conjuntos de datos volumétricos, escultura virtual, modelado basado en formas libres, entre otros (*Bielser*[9], 2004). En la simulación quirúrgica, los algoritmos de corte permiten la interacción con tejidos virtuales, usando herramientas como bisturíes o pinzas.

Los primeros algoritmos para simular cortes usaban mallas de superficie (*Basdogan et. al.*[8], 1999), debido a que fueron también las primeras representaciones para objetos en tres dimensiones. La generalización para mallas de volumen surgió tras popularizarse los métodos de simulación de deformación, con métodos como el de elemento finito [56][57][69]. Muchas de las aplicaciones se dieron en el campo de la cirugía virtual.

En este trabajo se abordan cinco metodologías para la simulación de cortes: separación de nodos, retracción, eliminación de elementos, deformación plástica y cambio topológico. Por su complejidad, éste último es una descripción general del trabajo de *Bielser*[9]: *A state machine for real-time cutting of tetrahedral meshes*. El objetivo final es mostrar la variedad de métodos aplicables para la simulación de cortes sobre tejido deformable, abordado en el Capítulo 2, y determinar cuáles de ellos son mejores para una u otra circunstancia. Por ejemplo, cuáles de ellos son útiles para simular incisiones o disecciones, o bien, cuáles de ellos son mejores para simular resecciones o extirpación de tejidos blandos.

4.3.1. Método de separación de nodos (*split*)

Uno de los métodos más simples para la simulación de cortes es el llamado por separación de nodos o *split*, el cual tiene las siguientes características:

- Trabaja sobre los nodos de una malla, por lo que el primer requisito es disponer de un

método de detección de colisiones basado en puntos.

- Por la razón anterior, es implementado sobre sistemas físicos que operan sobre la dinámica de las partículas, como el método de Müller[51], de mallas de superficie o volumen deformables.
- Es relativamente independiente de la conectividad entre sus elementos, ya que sólo necesita de una lista de puntos a separar y un algoritmo para resolver las referencias de los vértices en los elementos de conectividad (triángulos, cuadrados, tetraedros o hexaedros).
- Dado que en el proceso se necesita definir un plano de corte, es necesario establecer un criterio para la definición de esa geometría.
- Puede ser empleado para simular incisiones, disecciones y resección en procedimientos quirúrgicos.

A grandes rasgos, el método se divide en tres partes. En una primera parte se determina qué vértices se encuentran afectados por el corte y se construye un criterio de pertenencia de los tetraedros (volumen) o triángulos (superficie) asociados a cada uno de ellos; para tal tarea, se define un plano que contiene al punto afectado. En una segunda parte, los vértices afectados se duplican. Finalmente, en el tercer paso, se actualizan las referencias de todos los triángulos o tetraedros que contienen al vértice que sufrió una separación en base a su signo. En las siguientes subsecciones se aborda el análisis del método y el algoritmo asociado para el caso de mallas de tetraedros. Sin embargo, el algoritmo es totalmente válido para cortar mallas de superficie cambiando únicamente los criterios de pertenencia.

Construcción del plano de corte

Decimos que un plano está determinado, cuando se conoce un punto $P_0(x_0, y_0, z_0)$ contenido sobre éste y un vector normal a su superficie $\underline{N} = (A, B, C)$ (Figura 4.7). Así, todo punto $P(x, y, x)$ está contenido en el plano si se cumple que :

$$A(x - x_0) + B(y - y_0) + C(z - z_0) = 0 \tag{4.1}$$

de otra forma, el punto P puede estar por encima o por debajo de éste.

Asimismo, decimos que un vector $\underline{P}_i = \underline{p} - \underline{p}_0 = (x - x_0, y - y_0, z - z_0)$ está contenido en el plano si se cumple que:

$$\underline{P}_i \cdot \underline{N} = 0 \tag{4.2}$$

de otra forma, si la componente escalar λ, resultado de proyectar $\underline{P}_i$ sobre $\underline{N}$:

$$\lambda = \frac{\underline{P}_i \cdot \underline{N}}{||\underline{N}||} < 0 \tag{4.3}$$

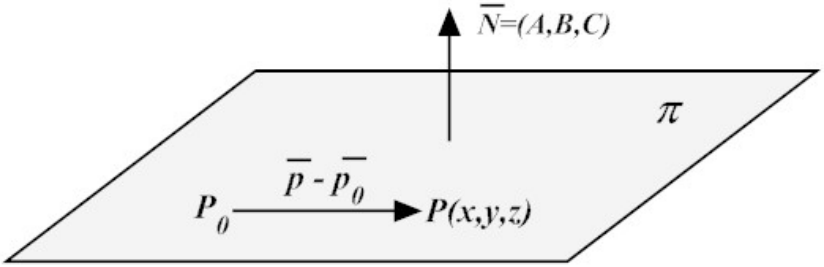


Figura 4.7. *El plano (Aguilar[1], 2010).*

entonces el punto P se encuentra debajo del plano. Si:

$$\lambda > 0 \tag{4.4}$$

el punto P se encuentra por encima del plano.

Criterio de pertenencia de un tetraedro

Definimos un plano de corte $\Pi(P_0, \underline{N})$, como el plano que subdivide una malla de volumen en dos regiones no vacías; no necesariamente simétricas o con el mismo número de puntos. Todo tetraedro tiene asociado un vértice de posición P_0 y a P_1, P_2, P_3 como posiciones de los vértices adyacentes a P_0, satisfaciendo una de las siguientes condiciones (Figura 4.8):

C1: Uno de sus vértices adyacentes se encuentra por encima (+) o por debajo (−) del plano de corte y los otros dos sobre éste. En tal caso, el signo del tetraedro es igual al signo del vértice que no está en el plano.

C2: Dos de sus vértices se encuentran, ambos, por encima (+) o por debajo (−) y el otro se encuentra sobre el plano de corte. En tal caso, se toma el signo común entre los dos vértices que no pertenecen al plano.

C3: Los tres vértices se encuentran, todos, por encima (+) o por debajo (−) del plano. En tal caso, se toma el signo en común de los tres vértices.

C4: Un vértice se encuentra por encima (+) o por debajo (−) y los otros dos del lado opuesto. En tal caso, se toma el signo en común de los dos vértices opuestos.

C5: Un vértice se encuentra por encima (+) o por debajo (−) del plano, el segundo del lado contrario y el tercero sobre el plano. En tal caso, se toma cualquier signo arbitrariamente (+/−).

C6: Los tres vértices son co-planares. Condición no válida.

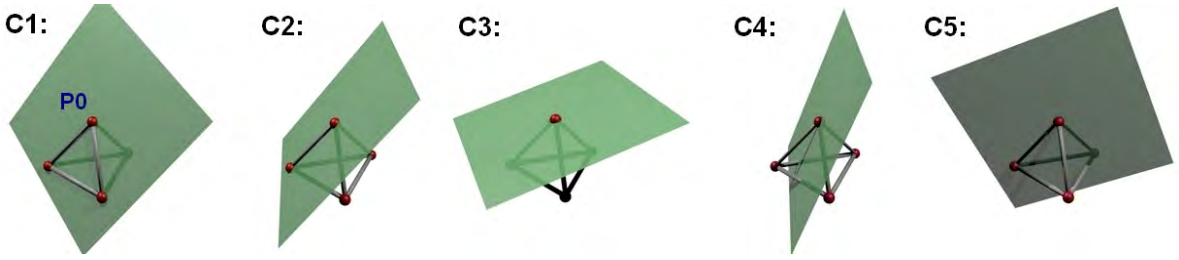


Figura 4.8. *Casos para determinar el signo de un tetraedro y aplicar el algoritmo de corte por separación de nodos.*

Algoritmo

Mediante el Algoritmo 12, aplicamos la idea general del método de separación de nodos, antes descrito. Al inicio se marcan todos los vértices que se ven afectados por el corte; éstos vienen dados por el algoritmo de colisiones. A continuación para cada uno de los vértices V^i_{corte} se obtiene un vector normal $\underline{N}$, cuyo criterio de elección depende de la geometría de la herramienta de corte. Así, por ejemplo para cortes con bisturí, se elige la dirección normal al eje del instrumento. Si la herramienta de corte fuera un lazo de resección, se elegiría por cada vértice un vector perpendicular a la tangente de la curva que apunte al centro del lazo. En la línea 5 se define un plano Π a partir del punto V^i y la normal $\underline{N}$. En la línea 6 se ejecuta un método de búsqueda de todos los tetraedros asociados a V^i, creándose una lista T_{corte}. Cada tetraedro de la lista es etiquetado dependiendo de su condición con respecto al plano Π como: encima (+) o debajo (−). Para ello se aplica el Algoritmo 13, clasificando de acuerdo con la Tabla 4.1. El método **ObtenerSigno** se encarga de buscar en la tabla el signo correspondiente a la terna $(\lambda_1, \lambda_2, \lambda_3)$.

Algoritmo 12: Algoritmo de corte por separación de nodos

```
1  begin
2      Marcar todos los vértices afectados por el corte: V_corte;
3      for ∀V^i_corte do
4          N ← ObtenerNormal();
5          Π ← DefinirPlano(V^i, N);
6          T_corte ← ObtenerTetraedrosAsociados(V^i);
7          for ∀T^i_corte do
8              EtiquetarTetraedro(T^i_corte, Π);
9          end
10         Duplicar(V^i → V^i*);
11         ReconfigurarTetraedros(T_corte, V^i, V^i*);
12     end
13 end
```

Algoritmo 13: EtiquetarTetraedro(T, Π)

```
1 begin
2     V_0 = Π.P_0;
3     N = Π.N;
4     for (i = [1 : 3]) do
5         V_i = T_i − V_0;
6         λ_i = (V_i · N) / |N|;
7     end
8     T_signo ← ObtenerSigno(λ_1, λ_2, λ_3);
9 end
```

Debe observarse que, cuando un tetraedro entra en la Condición 5 (C5), el signo es arbitrario, por lo que puede elegirse cualquiera de los dos (+ ó −); teniendo en cuenta que una mala elección podría resultar en una mala referenciación en el método **ReconfigurarTetraedros**. Una forma de resolverlo es asignando siempre el mismo signo, ya sea + ó −, a todos los tetraedros que caen en esta condición.

En la línea 10 del Algoritmo 12 se duplica el vértice V^i y se crea un nuevo vértice llamado V^{i*}. Este nuevo vértice se almacena en la lista de vértices de la malla dinámicamente, o bien, para optimizar, algunas veces se reserva memoria adicional durante la carga de un modelo. Finalmente en la línea 11 se reconfiguran todos los tetraedros de la lista T_{corte}, asignando el nuevo vértice V^{i*} a todos aquellos tetraedros que tienen signo (−), mientras que los tetraedros con el signo (+) conservan su referencia al vértice V^i; o de manera inversa.

λ_1	λ_2	λ_3	Signo	Criterio		λ_1	λ_2	λ_3	Signo	Criterio		λ_1	λ_2	λ_3	Signo	Criterio
+	+	+	+	C3		−	+	+	+	C4		0	+	+	+	C2
+	+	−	+	C4		−	+	−	−	C4		0	+	−	+/−	C5
+	+	0	+	C2		−	+	0	+/−	C5		0	+	0	+	C1
+	−	+	+	C4		−	−	+	−	C4		0	−	+	+/−	C5
+	−	−	−	C4		−	−	−	−	C3		0	−	−	−	C2
+	−	0	+/−	C5		−	−	0	−	C2		0	−	0	−	C1
+	0	+	+	C2		−	0	+	+/−	C5		0	0	+	+	C1
+	0	−	+/−	C5		−	0	−	−	C2		0	0	−	−	C1
+	0	0	+	C1		−	0	0	−	C1		0	0	0	NA	C6

Tabla 4.1. Criterio de clasificación de tetraedros de acuerdo con su posición respecto a un plano de corte. $\lambda_i = \{encima, debajo, sobre\} = \{+, -, 0\}$.

El resultado es una malla con $n + |V_{corte}|$ vértices, donde n es el número de vértices antes de aplicar el algoritmo y $|V_{corte}|$ es el número de vértices involucrados en el corte. No se crean nuevos tetraedros en el proceso, por lo que el número de tetraedros en la malla se mantiene constante. Un diagrama que resume el método completo se presenta en la Figura 4.9.

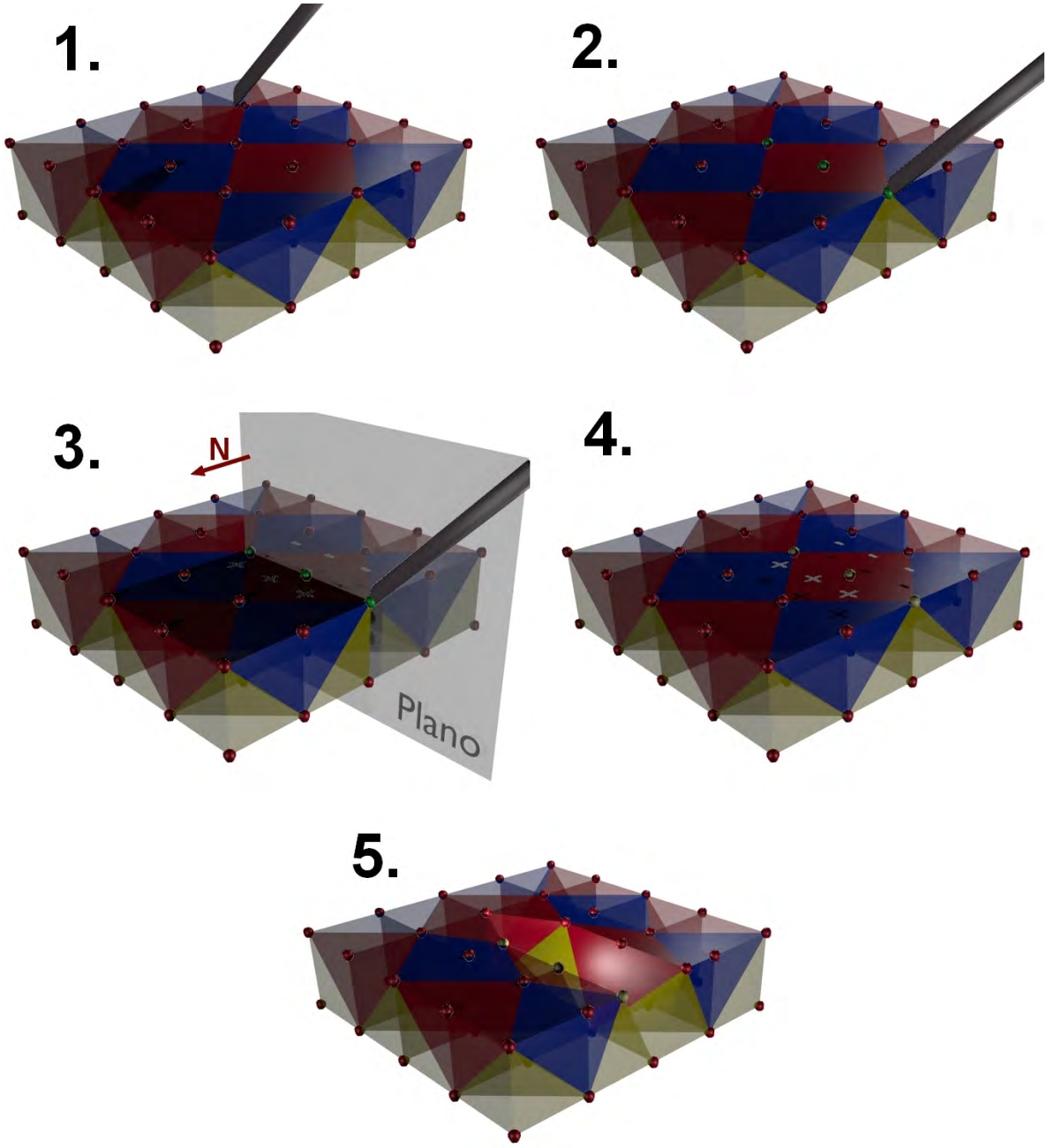


Figura 4.9. *Método de separación de nodos. 1) No hay interacción entre la herramienta y la malla. 2) Se marcan los nodos en colisión. 3) Se crea un plano y se etiquetan los tetraedros. 4) Se duplican los nodos. 5) Se reconfigura la malla.*

La Figura 4.10 muestra un ejemplo de la aplicación de este método sobre una malla de tetraedros que representa un tejido con estructura regular.

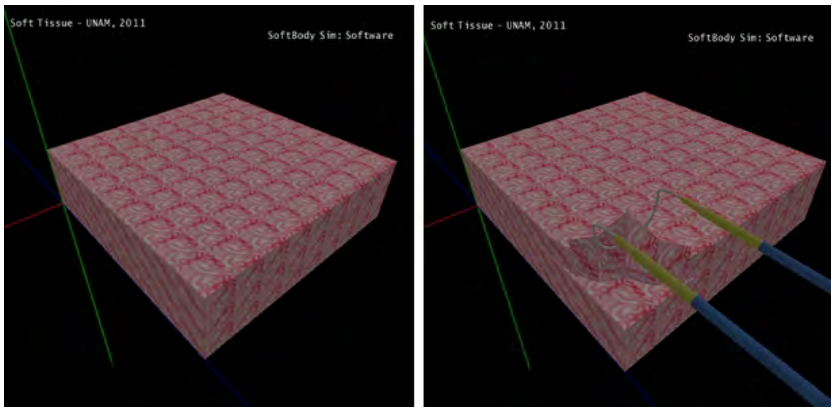


Figura 4.10. *Método de separación de nodos aplicado sobre una malla que representa un tejido deformable usando NVIDIA PhysX.*

Dentro de los paquetes de software, el motor de física *NVIDIA PhysX* posee una instrucción para aplicar este método sobre un nodo de un objeto deformable *NxSoftBody*, siendo la instrucción para ejecutarlo:

```
bool NxSoftBody::tearVertex(const NxU32 vertexId,
                            const NxVec3 &normal
 );
```

donde `vertexId` es el índice al vértice que se va a separar y `normal` es el vector normal al plano de corte. Esta instrucción, en la versión del SDK 2.8.1, se encuentra aún en fase experimental por lo que posee algunas limitaciones en cuanto al tipo de mallas.

Criterios de pertenencia para cortes de mallas triangulares

Como se mencionó al principio, el método para mallas de tetraedros es totalmente aplicable para mallas de superficie triangulares. La diferencia radica en la definición del criterio de pertenencia, que en este caso se aplica para triángulos. Cada triángulo se clasifica de acuerdo con la Tabla 4.2, donde se ha redefinido los criterios de pertenencia de acuerdo con las siguientes condiciones:

C1: Los dos vértices se encuentran por encima (+) o por debajo (−) del plano de corte. En tal caso, se toma el signo común entre ambos vértices.

C2: Un vértice se encuentra por encima (−) o por debajo (−) del plano de corte, y el otro del lado opuesto. En tal caso, se toma arbitrariamente cualquier signo (+/−).

C3: Un vértice se encuentra sobre el plano de corte y el otro por encima (+) o por debajo (−). En tal caso, se toma el signo del vértice que no está sobre el plano.

C4: Los tres vértices del triángulo son coplanares y corresponden al plano de corte. En tal caso, se elige arbitrariamente cualquier signo (+/−).

λ_1	λ_2	Signo	Condición
+	+	+	C1
+	−	+/−	C2
+	0	+	C3
−	+	+/−	C2
−	−	−	C1
−	0	−	C3
0	+	+	C3
0	−	−	C3
0	0	+/−	C4

Tabla 4.2. Criterio de clasificación de triángulos de acuerdo con su posición respecto a un plano de corte. $\lambda_i = \{encima, debajo, sobre\} = \{+, -, 0\}$.

4.3.2. Método de retracción

El método anterior, como pudo observarse, se basa totalmente en un análisis geométrico, tanto de la malla del objeto, como de la interacción entre ésta y una herramienta de corte. Una extensión más interesante es el llamado método de retracción o "*tearing*" (rasgado), que consiste en limitar la elongación de una arista (resorte) de un tetraedro (malla de volumen) o triángulo (malla de superficie) dentro de un umbral de deformación.

Una de las ventajas, es que no necesariamente se limita el modelo de deformación elegido, ya que el método actúa una vez que ésta ha finalizado; verificando los cambios significativos en las elongaciones de las aristas y aplicando, si procede, el algoritmo de separación sobre nodos adyacentes.

Dadas sus características, en cirugía virtual es ideal para simular extirpación de tejido o toma de biopsias. Siendo en estos casos necesario el uso de mallas de volumen. Los casos con mallas de superficie se limitan a aplicaciones de rasgado de telas, entre otras.

A continuación se describe la idea básica del método para mallas de tetraedros, teniendo en cuenta que se restringe a un corte local; es decir, en una malla deformable se fija uno de sus nodos a una herramienta de retracción, y se desplaza junto con ella para provocar un cambio. Sin embargo, el método es extensible a múltiples interacciones con múltiples nodos. La descripción para superficies puede encontrarse en [51].

Elementos básicos

El método para mallas de tetraedros requiere de los siguientes elementos:

- Un conjunto de nodos con conectividad, que forman una malla deformable.
- Todo nodo V_i tiene asociado un conjunto de nodos adyacentes V_j, conectados mediante elementos elásticos. El comportamiento elástico estará dado por el modelo físico de deformación.
- Un criterio de elongación máxima ε_{max}, determinado en el mismo modelo de deformación o mediante la verificación de cambios entre las elongaciones anteriores (ε_j^0) y actuales (ε_j^1). En este trabajo usaremos la segunda opción por ser fácil de entender y además es independiente del modelo de deformación.
- Varios planos de corte Π_j, definidos por cada nodo V_j y su respectiva normal N_j, definida como:

$$N_j = V_i - V_j. \tag{4.5}$$

Estos elementos básicos se muestran en la Figura 4.11.

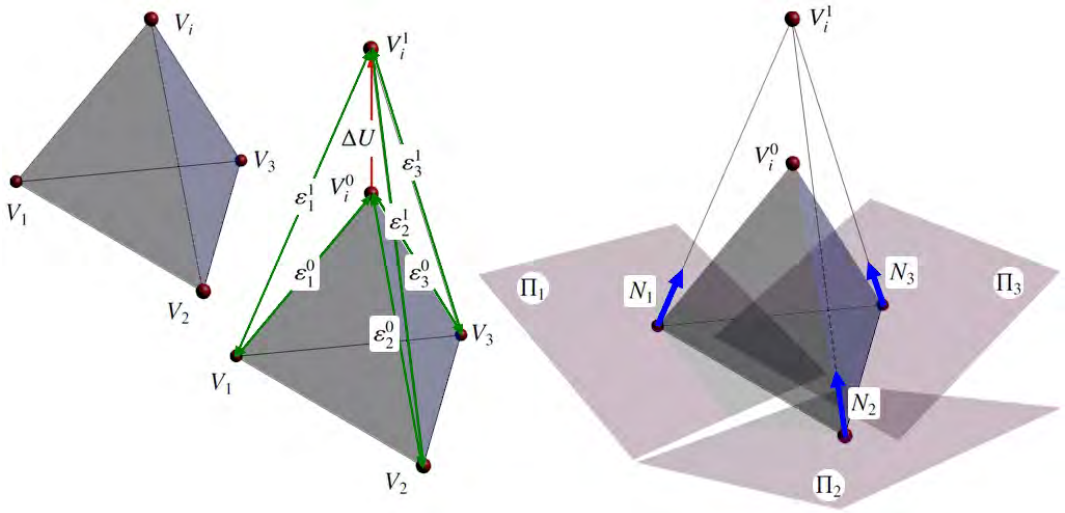


Figura 4.11. *Elementos básicos para aplicar el método de corte por retracción.*

Metodología

El método consta de cuatro pasos, desde que la herramienta hace contacto con la malla hasta que se efectúa una separación de elementos en la estructura. Éstos se describen a continuación:

1. Hay un contacto entre la herramienta de corte (por ejemplo, unas pinzas) y la malla de tetraedros deformable. Se detecta el punto de contacto con la superficie y se realiza un ajuste entre este último y el nodo de la malla más cercano V_i. El criterio para decidir el punto más cercano es, en primera instancia, un criterio de distancias. Se fija ese nodo a la punta de la herramienta, conocido. Se marcan todos los tetraedros T_{corte} involucrados con el nodo V_i y se calculan las elongaciones iniciales ε_j^0. Se crea una submalla de tetraedros Ω_i con T_{corte} como elementos iniciales.

2. La herramienta se desplaza, provocando una deformación ΔU sobre el nodo V_i y sobre cada tetraedro T_{corte}. En cada paso se calculan las elongaciones $\varepsilon_j = \varepsilon_j^1 - \varepsilon_j^0$.

3. Por cada cambio en la deformación, se verifica si la elongación ε_j en las aristas (que comunican V_i con sus vértices adyacentes V_j), está dentro del umbral $\varepsilon_j^{max} - \varepsilon_j^0$. En caso de que ε_j se encuentre por encima del umbral, se aplica el Algoritmo 12 de separación de nodos sobre el nodo V_j, con la diferencia de que cada vez que se lleva a cabo una separación, el nodo original se conserva en la malla original, mientras que el nodo duplicado se asigna a la submalla Ω_i.

4. Una vez que todos los tetraedros T_{corte} han perdido su conectividad con la malla original, se elimina el nodo V_i de la malla original y se asigna a la submalla Ω_i.

Al final del método se tendrán dos sub-mallas, una sobre la que se realizó el corte y otra, más pequeña, con el pedazo extirpado. La Figura 4.12 muestra la secuencia de pasos sobre una simulación de tejido blando.

Otro criterio para definir el factor de elongación es verificar si:

$$\varepsilon_j = \frac{\varepsilon_j^1}{\varepsilon_j^0} \leq \alpha \tag{4.6}$$

donde α es un factor de elongación máxima y constante para todas las aristas. Si $\varepsilon_j > \alpha$ se efectúa un proceso de separación de nodos. De este modo se pueden simular materiales muy resistentes (valores de α grandes) o materiales poco resistentes (valores de α muy pequeños).

Configuración en NVIDIA PhysX

Así como el SDK de *PhysX* tiene soporte para el método de *split*, también tiene soporte experimental para el método de retracción, siendo un fragmento de código para configurar esta funcionalidad sobre un modelo deformable del tipo *NxSoftBody*, el mostrado a continuación:

```
1 SoftTissue::SoftTissue(NxScene *scene,
2                         NxSoftBodyDesc &desc)
3 {
```

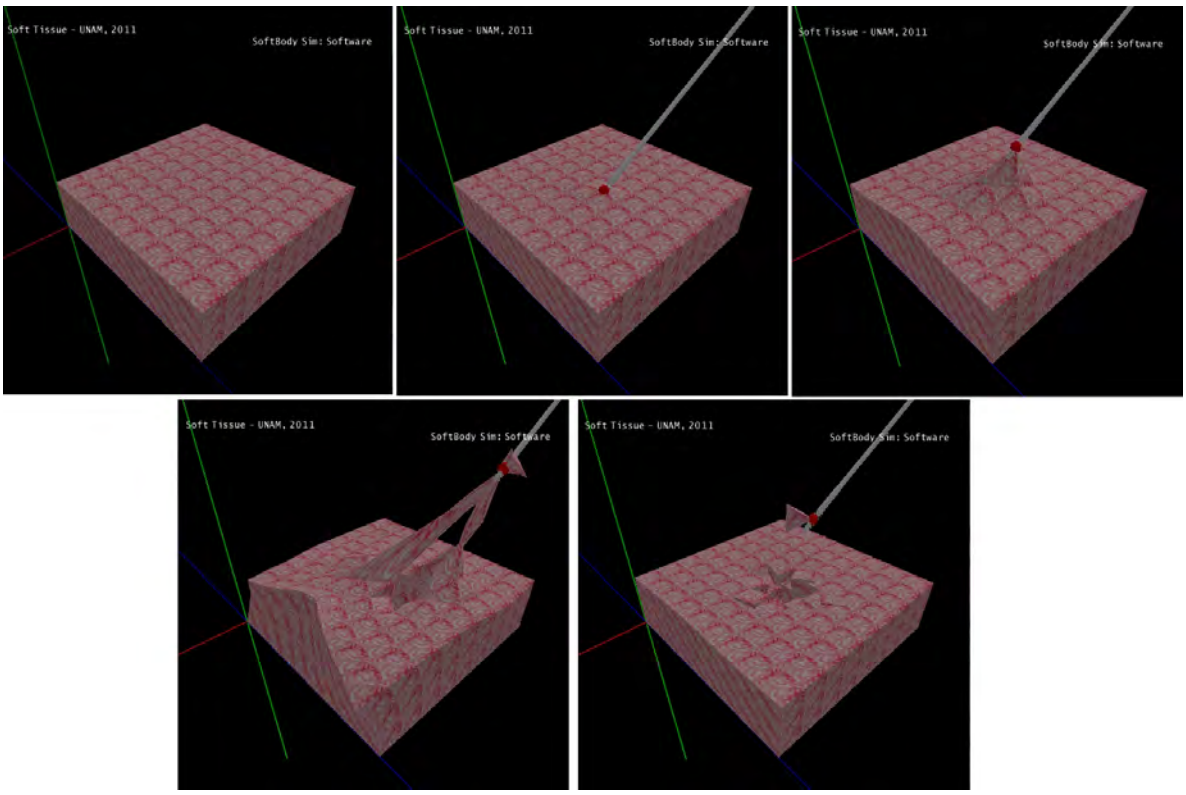


Figura 4.12. *Método de retracción para simular desprendimiento de tejido.*

```
4
5       // Descriptor de la malla deformable
6       NxSoftBodyMeshDesc meshDesc;
7
8       // Generamos un plano
9       // con resolución 10x10 y 1.0 de espesor
10      generatePlane(meshDesc, 10, 10, 1.0);
11
12      // Inicializamos el modelo deformable
13      init(scene, desc, meshDesc, NULL);
14
15      // Restringimos algunos vértices
16      NxVec3 *pos;
17      pos = new NxVec3(0,0,0);
18      mSoftBody->attachVertexToGlobalPosition(0, *pos);
19      pos->set(0,0,10);
20      mSoftBody->attachVertexToGlobalPosition(10, *pos);
21      pos->set(10,0,10);
22      mSoftBody->attachVertexToGlobalPosition(407, *pos);
23      pos->set(10,0,0);
24      mSoftBody->attachVertexToGlobalPosition(396, *pos);
25
26      // Activamos el factor de rasgado
27      mSoftBody->setTearFactor(2.0);
28
29      // Tear Factor = 1.0 (material poco resistente)
30      // Tear Factor > 10.0 (material muy resistente)
31
32      // Cargamos y asignamos una textura
33      Texture *tex = new Texture("Textures/", "tissue.bmp");
34      tex->LoadBMP();
35      textureID = tex->getID();
36  }
```

Para preparar el método para la simulación, en la creación del objeto suave se configura:

```
1   void SetupSoftTissue(){
2     // Descriptor de cuerpos deformables
3     NxSoftBodyDesc softBodyDesc;
4
5     softBodyDesc.globalPose.t = NxVec3(0,0,0); // Condición inicial
6     softBodyDesc.particleRadius = 0.1f;        // Radio de la partícula
7     softBodyDesc.volumeStiffness = 0.5f;       // Rigidez del volumen
8     softBodyDesc.stretchingStiffness = 0.5f;   // Rigidez del resorte
9     softBodyDesc.friction = 1.0f;              // Fricción
10    softBodyDesc.solverIterations = 5;         // Número de iteraciones
11    softBodyDesc.dampingCoefficient = 0.7f;    // Coeficiente
12                                               // de amortiguamiento
13    softBodyDesc.density = 1.0;
14    ...
```

```
15    softBodyDesc.flags |= NX_SBF_TEARABLE;
16    ...
17    SoftTissue *softTissue = new SoftTissue(gScene, softBodyDesc);
18    ...
19  }
```

Para mover un nodo una vez efectuada una detección del vértice V_i se implementa:

```
1  NxVec3       toolTip;         // Punta de la herramienta
2  int          selectedVertex; // Vértice seleccionado
3  NxSoftBody   *softBody;       // Objeto deformable
4
5  void MoveSoftBody()
6  {
7    if (selectedVertex >= 0)
8    {
9      softBody->attachVertexToGlobalPosition(selectedVertex, toolTip);
10     softBody->setPosition(toolTip, selectedVertex);
11   }
12 }
```

4.3.3. Método de eliminación de elementos o drenado

El método de eliminación de elementos o drenado, es otro de los métodos más simples para la implementación de cortes sobre mallas de volumen, e inclusive sobre mallas de superficie. Una de sus grandes ventajas es que no depende del modelo de simulación, por lo que puede aplicarse sobre objetos rígidos o sistemas de partículas.

El método requiere de lo siguiente:

- Un conjunto de elementos básicos, que pueden ser puntos (partículas), triángulos (superficies) o tetraedros (volúmenes), que forman una geometría espacial.
- La interacción entre una herramienta de corte y la geometría. Para ello se implementa un algoritmo de detección de colisiones basado en los elementos básicos. Así, por ejemplo, para partículas se requiere la detección de colisiones entre un objeto rígido y puntos. Para superficies necesitamos un algoritmo de detección entre un objeto rígido y los triángulos de la superficie. En el caso de volúmenes requerimos de un algoritmo para determinar la colisión entre un objeto rígido y un conjunto de tetraedros. En todos los casos el algoritmo de detección debe tomar en cuenta elementos completos, por lo que una de las características importantes de la geometría, es que debe estar formada por una gran cantidad de elementos básicos (geometría de alta resolución).
- Una técnica computacional para la liberación de referencias en memoria de los elementos removidos.

Algoritmo

El algoritmo es bastante simple, la única complejidad radica en el método de detección de colisiones elegido. El Algoritmo 14 muestra el pseudocódigo del método. Como puede observarse todo se reduce a eliminar un elemento de la geometría, y dado que generalmente una malla se compone de una lista de vértices y una lista de referencias a vértices que forman triángulos o tetraedros, la tarea consiste en realizar una operación de *pop* sobre la lista de elementos.

Algoritmo 14: Algoritmo de drenado

```
1 begin
2     Marcar todos los elementos en colisión: E_coll;
3     for ∀E^i_coll do
4         Eliminar E^i_coll de la geometría;
5     end
6 end
```

Lo interesante del algoritmo es que permite concentrarnos en la geometría de la malla, más que en su propia programación. Esto significa que el reto principal que ofrece es el planteamiento de una metodología para la generación de mallas de alta resolución, pues de otro modo el método es inservible, dado que se crean efectos indeseables.

La Figura 4.13 muestra algunos ejemplos del método de drenado sobre geometrías de volumen. Como puede observarse, este método es ideal para simular resecciones de tejido, principalmente. Por el contrario, es poco realista e ineficiente para la simulación de incisiones o retracción.

Configuración en *NVIDIA PhysX*

Como en los dos métodos anteriores, el SDK de *NVIDIA PhysX* posee capacidad para este método. La forma de configurarlo es mediante un actor *NxActor*, al cual se le activa una bandera de drenado. En el siguiente fragmento de código se crea una esfera drenadora.

```
NxActor    *drainModel;

void NvidiaPhysX::CreateActors(void){
  // Creamos geometría
  drainModel = CreateSphere(NxVec3(-10.0, 3.0, 0.0), 1.5, 1.0);
  // Configuramos objeto drenador
  drainModel->getShapes()[0]->setFlag(NX_SF_SOFTBODY_DRAIN, true);
  drainModel->getShapes()[0]->setFlag(NX_SF_SOFTBODY_TWOWAY, true);
}
```

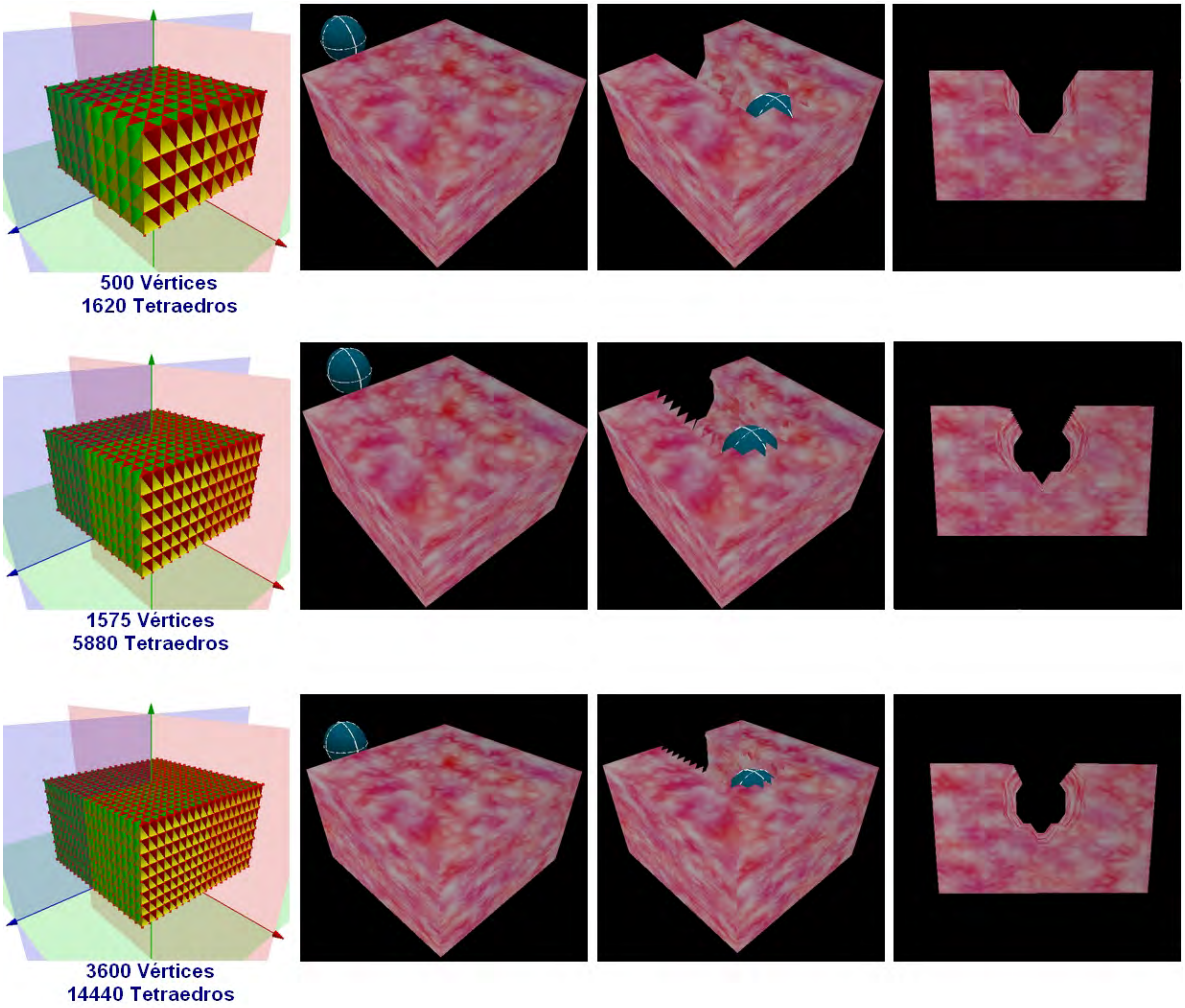


Figura 4.13. *Ejemplos y efectos del algoritmo de drenado. De izquierda a derecha, una esfera penetra sobre un objeto formado por nodos y tetraedros. La esfera drena todos los tetraedros con los cuales detecta una colisión, mientras que éstos van desapareciendo conforme avanza. Puede notarse que conforme aumenta la resolución de la malla, el hueco va tomando la forma del objeto rígido.*

4.3.4. Método de deformación plástica

Otra forma de representar cortes es mediante una malla que tiene un comportamiento elástico y bajo ciertas condiciones un comportamiento plástico, decimos entonces que se comporta como un material elastoplástico[50]. Un objeto tiene comportamiento elástico cuando al ser sometido a un esfuerzo, éste recupera su forma original; sin que ello signifique recuperar su estado inicial debido a la pérdida de energía durante su restitución. Por otro lado, decimos que un objeto tiene un comportamiendo plástico cuando al ser sometido a un esfuerzo, éste no recupera su forma original[85], sino que toma la forma del objeto que imprimió dicha fuerza. La Figura 4.14 muestra un ejemplo de estos principios básicos.

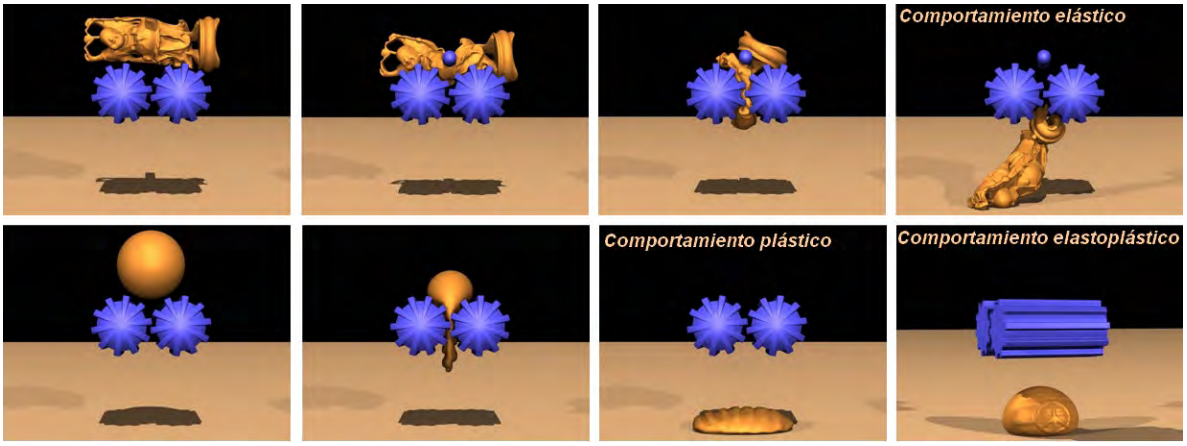


Figura 4.14. *Comportamiento elástico y plástico de un material (Irving[33], 2006).*

El concepto de simulación de cortes usando un comportamiento plástico consiste en la interacción de una herramienta de corte, que sobrepone una fuerza externa a un modelo sobre su superficie provocando su deformación. Mientras el objeto de corte se encuentre en contacto con dicha superficie, el modelo se comportará como un material elastoplástico, tomando la forma del objeto de corte y restituyendo su propiedad elástica una vez que dicha interacción finalice.

La ventaja principal de este método, desde cualquier tipo enfoque, es que una malla, ya sea de superficie o de volumen, conserva el mismo número de nodos y tetraedros original durante toda la simulación; esto significa, por una parte, en una ganancia de tiempo de procesamiento, al no tener que verificarse eliminación de elementos, creación de nuevos nodos o algún cambio topológico de la malla. Por otra parte no se requieren mallas de muy alta resolución, siendo posible la implementación de este método sobre superficies, agregando únicamente algunos parámetros restrictivos, como puede ser un umbral de la deformación o la creación de nodos “fantasma” sobre la superficie para restringir la propagación de la deformación, evitando inflamientos excesivos.

La desventaja del método radica en su carácter heurístico, pues no representa lo que sucede en la realidad durante una acción de corte. Por ello, es inutilizable para procedimientos de incisión o disección, en los que se busca abrir paso sobre la malla del objeto. Sin embargo, es muy útil para simular resecciones de tejido en conjunto con otros efectos; como la adición de un pedazo de tejido que simula un desprendimiento del volumen extirpado. En este caso, el método ayuda a despejar el espacio ocupado por el pedazo ficticio y dejar como rastro el hueco que ocupaba.

El método tiene dos formas de plantearse: uno basado en el comportamiento elastoplástico real de un material y otro basado en la dinámica de los nodos que forman la malla de volumen o superficie. El primero emplea el mismo modelo usado para el cálculo de deformaciones agregando nuevos parámetros físicos que lo dotan de un comportamiento plástico. El segundo se centra en restringir la dinámica de las partículas de la malla cuando han sufrido un proceso de deformación. Este último método, por no agregar más variables de simulación al modelo de deformación, es mucho más sencillo de implementar y presenta buenos resultados visuales.

Un ejemplo del primer enfoque es el trabajo de Irving[33], Teran y Fedkiw, que presentan en su artículo *Tetrahedral and hexaedral invertible finite elements*. En éste, incorporan la propiedad de plasticidad en un modelo de deformación de elemento finito agregando una descomposición multiplicativa mediante:

$$F_D = F_e F_p \tag{4.7}$$

donde F_p es la fuerza de deformación plástica permanente y F_e es la fuerza de deformación elástica. Esta formulación multiplicativa permite una completa separación entre los cálculos del modelo elástico y los cálculos del modelo plástico. Otras formulaciones, emplean una descomposición aditiva[78][59][33], esto es:

$$F_D = F_e + F_p \tag{4.8}$$

Esta última formulación permite incorporar en el mismo modelo deformable la propiedad de plasticidad y resolver el sistema mediante algún método de integración. Por ejemplo, Teschner[78] aplica este principio sobre un algoritmo de integración de Verlet de la siguiente forma:

$$x(t+h) = 2x(t) - x(t-h) + h^2 \frac{F(t)}{m} + O(h^4) \tag{4.9}$$

$$v(t+h) = \frac{x(t+h) - x(t-h)}{2h} + O(h^2) \tag{4.10}$$

con $F(t) = F_D(t) + F_A(t) + F_D()$, donde $F_A(t)$ son las fuerzas de conservación del área y $F_V(t)$ son las fuerzas de conservación del volumen. Para mayor información acerca de este método de integración ver el Apéndice C.

Un ejemplo de cómo incorporar el modelo de plasticidad dentro del cómputo de deformaciones se muestra en el Algoritmo 15 de Müller-Gross[50](2004). En el cual se aplica el método de elemento finito (FEM) para resolver el modelo 4.11.

$$M\ddot{x} + C\dot{x} + K'x + f_0' - f_{plastic} = f_{ext} \tag{4.11}$$

Algoritmo 15: Algoritmo de deformación elastoplástica de Müller-Gross[50]

```
1  begin
2      for ∀ elemento e do
3          Calcular B_e(x_0), P_e(x_0), K_e(x_0);
4      end
5      Inicializar x^0, v^0;
6      i ← 0;
7      while true do
8          for ∀ elemento e do
9              Calcular R_e(x^i);
10         end
11         for ∀ elemento e do
12             Actualizar ε_{plastic,e};
13         end
14         Construir K = Σ_e R_e K_e R_e^{-1};
15         Construir f_0 = − Σ_e R_e K_e x_0;
16         Construir f_plastic = Σ R_e P_e ε_{plastic,e};
17         Calcular fuerzas externas: f_ext;
18         Resolver (M + ΔtC + Δt^2 K)v^{i+1} = Mv_i − Δt(Kx^i + f_0 + f_plastic − f_ext) para v^{i+1};
19         Actualizar x^{i+1} = x^i + Δtv^{i+1};
20         for ∀ elemento e do
21             Calcular el tensor de stress: σ = EB_e(R_e^{-1}x − x_0);
22             if máximo eigen-valor de σ > umbral de fractura then
23                 Fracturar malla;
24             end
25         end
26         i ← i + 1;
27     end
28 end
```

Una parte interesante del algoritmo anterior se encuentra entre las líneas 20-25, donde se agrega además la posibilidad de fragmentar la malla de acuerdo con un criterio de rigidez. Esto es, calculando el tensor de *stress* (Ver Apéndice B), podemos condicionar de acuerdo con las propiedades mecánicas del objeto, en qué momento la fuerza es lo suficientemente grande como para fracturarlo. El método de fractura o *fracturing* es ampliamente reportado en [50][59][47][49][85].

Otro ejemplo de simuación plástica, esta vez usando el segundo enfoque, consiste en que una vez finalizada la deformación de un nodo de la malla, se pasa por un estado de reinicio; esto es, todo parámetro involucrado en el modelo de deformación se congela, y sirven para reiniciar todas las cantidades iniciales de posición y elongación como si se tratara de un estado inicial. Así, en el cada ciclo de cálculo, la malla de triángulos o tetraedros iniciará con la forma anterior, propagando el efecto plástico indefinidamente mientras haya una condición de deformación válida. Un ejemplo de condición de deformación no válida es cuando se llega a tal grado de compresión de los elementos conectivos (resortes), que no se puede efectuar otra deformación.

A continuación se muestra un fragmento de código en C que aplica este segundo enfoque, donde tras aplicar una deformación usando un modelo de masas y resortes, se verifica una condición de corte (Línea 17), seguida de una verificación en el estado de deformación plástica (Línea 39).

```
1  // Función manejadora tras efectuarse una
2  // deformación sobre un nodo usando un modelo
3  // de masas-resortes
4  MSVertexHandler massSpringVolume( void *obj,
5                                    void *msvertex )
6  {
7      // Objeto de volumen
8      MSObject    *msobj = NULL;
9
10     // Lista de nodos
11     MSVertex    *msv = NULL;
12
13     msv = (MSVertex *)msvertex;
14     msobj = (MSObject *)obj;
15
16     // Se comprueba un estado de corte
17     if (toolGetMode() == TOOL_MODE_CUTTING)
18        volumePlasticDeformation( msobj, msv );
19     ...
20 }
21
22 // Método de deformación plástica
23 void   volumePlasticDeformation( MSObject *msobj,
24                                  MSVertex *msv )
25 {
26    MSEdge          *mse = NULL;   // Resorte
27    MESHEdge        *e = NULL;     // Arista
28    MESHVertex      *v = NULL;     // Vértice geométrico
29    MESHVertex      *nv = NULL;    // Vértice normal
30    MSVertex        *msvn;         // Nodo con Masa
31    MESHTetrahedro *TET;           // Tetraedro
32
33    ...
34    // Vértice
```

```
35      v = (MESHVertex*)msv->vertex;
36
37      // Se verifica el tipo de deformación
38      // MS_PLASTIC: Deformación plástica
39      if (msv->typeOfDeformation == MS_PLASTIC) {
40
41          // 1. Reseteo de posiciones iniciales
42          //    y anteriores, tanto para el vértice
43          //    geométrico, como el vértice físico (nodo)
44          msv->Pold.x = msv->P.x;
45          msv->Pold.y = msv->P.y;
46          msv->Pold.z = msv->P.z;
47          msv->P0.x = msv->P.x;
48          msv->P0.y = msv->P.y;
49          msv->P0.z = msv->P.z;
50          v->P.x = msv->P.x;
51          v->P.y = msv->P.y;
52          v->P.z = msv->P.z;
53          v->P0.x = msv->P.x;
54          v->P0.y = msv->P.y;
55          v->P0.z = msv->P.z;
56
57          // 2. Reseteo de aristas geométricas y
58          //    aristas físicas (resortes)
59          if ( dlIsEmpty( v->edges ) == FALSE ) {
60              dlMoveFirst( v->edges );
61              do {
62                  e = (MESHEdge *)dlGetCurrent( v->edges );
63                  mse = (MSEdge *)e->physlink;
64                  if (e->V[0] == v) nv = e->V[1];
65                  else              nv = e->V[0];
66                  msvn = (MSVertex *)nv->physlink;
67
68                  mse->initlen = vtDistance3D( v->P, nv->P );
69                  mse->len = mse->initlen;
70                  e->length = mse->initlen;
71
72              } while( dlMoveNext( v->edges ) == TRUE );
73          }
74
75          // 3. Reconfiguración de tetraedros
76          if ( dlIsEmpty( v->tetrahedra ) == FALSE ) {
77              TET = (MESHTetrahedro *)dlGetCurrent( v->tetrahedra );
78              meshTetrahedroVertexOrdering( TET );
79          }
80      }
81      ...
82  }
```

Como puede observarse, el método se aplica en tres pasos, que tienen por objetivo

la reestructuración de la totalidad de la malla en una forma concreta. En el primer paso se reinician las posiciones, en el segundo todas las longitudes de los resortes asociados al nodo en cuestión y en el tercero todas las propiedades de los tetraedros asociados.

La Figura 4.15 muestra un ejemplo de deformación plástica usando este último enfoque, sobre una malla de tetraedros y una superficie de triángulos para simular cortes en un sistema de cirugía de próstata[76] (CCADET-UNAM, 2009).

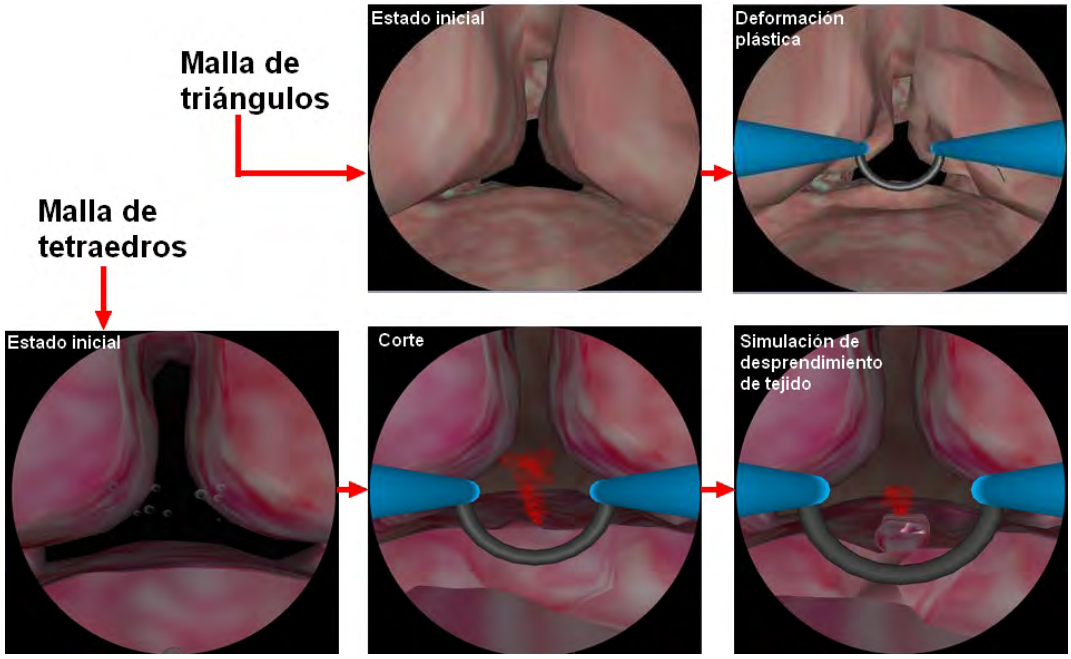


Figura 4.15. *Deformación plástica sobre mallas de triángulos[76] y tetraedros[22], para la simulación de cortes resectivos.*

4.3.5. Método de cambio topológico

Un enfoque más realista para la simulación de cortes es el basado en el cambio topológico de una malla de volumen. El principio básico consiste en disponer de un método de subdivisión espacial y de un criterio de análisis de posibles intersecciones entre una herramienta de corte con una primitiva de volumen, por ejemplo el tetraedro.

En 2004, *Bielser[9] et. al.*, presentaron un algoritmo de cambio topológico para la simulación de cortes en mallas de tetraedros basado en una máquina de estados. El trabajo sostiene que restringiendo el análisis sobre las posibles intersecciones en un tetraedro podemos lograr establecer un diagrama de estados, en el cual, se verifica un tipo de intersección sobre caras y aristas, definiendo una topología del tetraedro particular. Esta máquina de estados se muestra en la Figura 4.16 y tiene las siguientes reglas:

- Los estados se etiquetan con un *token*, que consiste en la concatenación de la letra que representa la topología de la arista afectada y el número asociado a la topología de la cara en intersección.
- Cada estado se ordena por el número de intersecciones con aristas de izquierda a derecha, y por el número y tipo de intersecciones con caras de arriba hacia abajo.
- Cada tetraedro guarda su estado de subdivisión actual.
- El estado O0 contiene el estado inicial del tetraedro.
- Cuando una arista o cara entra en un estado de corte, el tetraedro cambia su estado. El correspondiente estado de transición determina la modificación en la subdivisión actual.
- Tras algunas transiciones, cada tetraedro llega a un estado estado final, etiquetado con el número 3, que significa que la herramienta de corte ha dejado de afectar completamente el tetraedro.

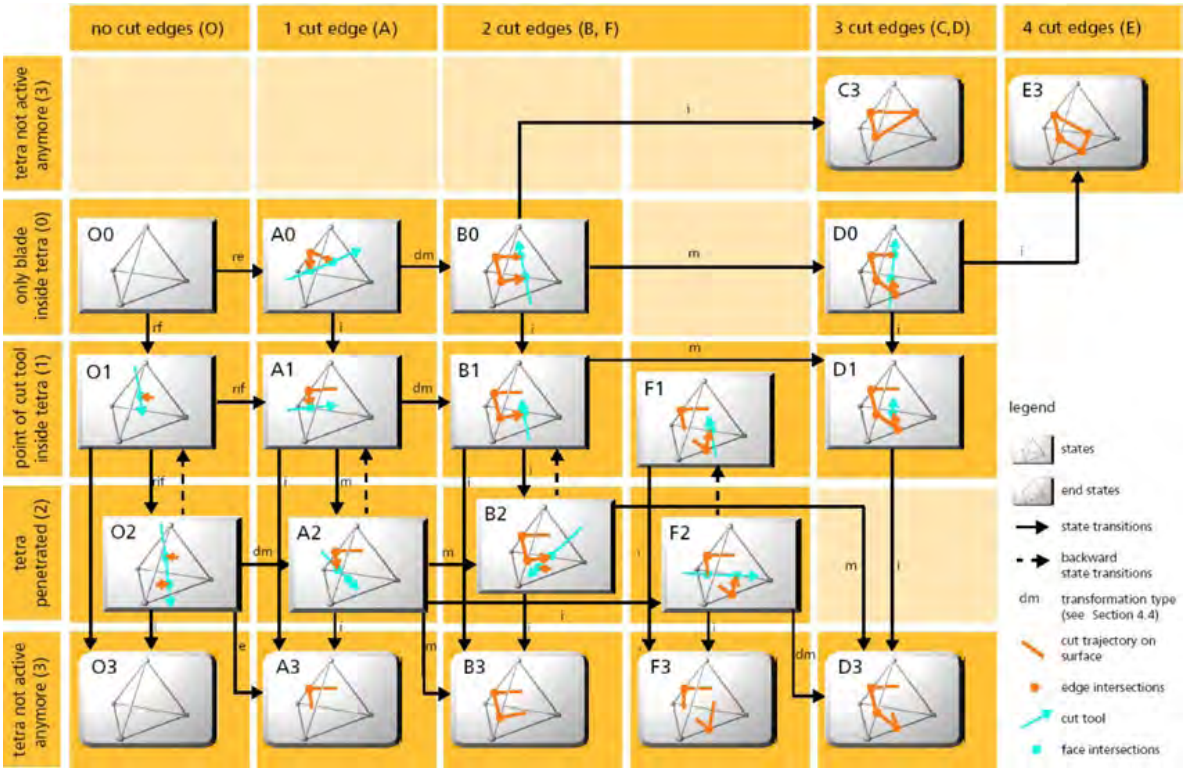


Figura 4.16. *Diagrama de máquina de estados para el cambio topológico de un tetraedro (Bielser[9], 2004).*

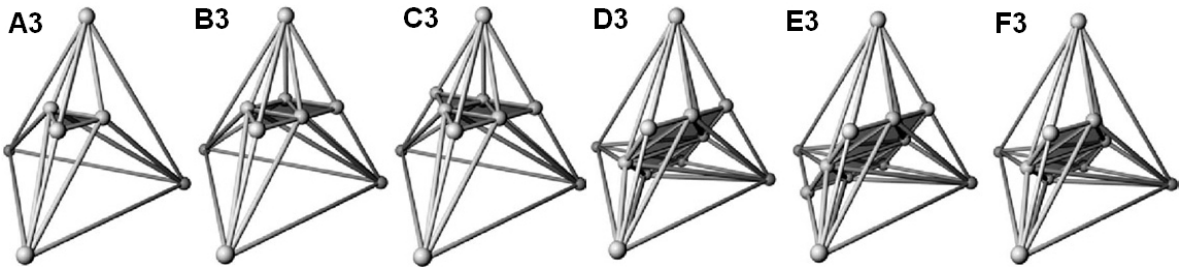


Figura 4.17. *Estados finales en la subdivisión de un tetraedro (Bielser[9], 2004).*

De esta forma, los estados finales A3-E3 representan las seis combinaciones posibles para subdividir un tetraedro, que se muestran en la Figura 4.17.

El Algoritmo 16 muestra el funcionamiento general del cambio topológico sobre un conjunto de tetraedros de una malla de volumen. Antes que todo, el primer paso consiste en la determinación de qué elementos se encuentran involucrados en la intersección herramienta-tetraedro; ésto generalmente está dado por un algoritmo de detección de colisiones. El algoritmo debe ser capaz de determinar las intersecciones con aristas y caras de un tetraedro, aún si se trata de mallas deformables, pues por ejemplo, en una cirugía virtual se deforman los tejidos. Los elementos se marcan en una lista de tetraedros marcados: T_{corte}.

Mientras no haya nuevas intersecciones con caras o aristas en un tetraedro, las intersecciones actuales en caras se ajustan con las posiciones de la intersección. Cuando ocurre una nueva intersección con una cara o arista, se valida un nuevo estado de subdivisión. Para un estado válido se subdivide el tetraedro mediante el Algoritmo 17.

El Algoritmo 17 determina la transformación entre el nuevo patrón topológico del tetraedro y su actual representación geométrica; es decir, busca en qué combinación del diagrama de estados encaja la nueva configuración y aplica la transición al nuevo estado.

En casos donde no encaja la configuración, se subdivide completamente el tetraedro de acuerdo con su último estado de subdivisión válido. Entonces, un conjunto repetido de pasos marcan todas las intersecciones entre la herramienta de corte y los sub-tetraedros. Cada subtetraedro afectado por la intersección también se subdivide una vez que se haya completado el cambio de estado de su tetraedro padre.

Ejemplo básico de corte

Un ejemplo de la aplicación de este algoritmo es el mostrado en la Figura 4.18.

a. Se inicia con un tetraedro con ninguna interacción por parte de alguna herramienta de

Algoritmo 16: Algoritmo de cambio topológico de Bielser[9].

```
1  begin
2      Marcar todos los tetraedros afectados por el corte: T_corte;
3      for ∀T^i_corte do
4          if T_i tiene nuevas intersecciones then
5              determina el nuevo estado de subdivisión;
6              if nuevo estado de subdivisión válido then
7                  SubdividirTetraedro(T_i);
8              end
9              else
10                 Determinar el último estado válido;
11                 Subdividir el tetraedro completamente;
12                 Marcar todos los sub-tetraedros afectados por el corte: sT_corte;
13                 for ∀sT^j_corte do
14                     Manejar transición de padre a sub-tetraedro.
                       SubdividirTetraedro(sT^j_corte);
15                 end
16             end
17         end
18         else
19             Ajustar las posiciones de intersección con intersecciones en caras.
20         end
21     end
22 end
```

Algoritmo 17: SubdividirTetraedro(T_i)

```
1  begin
2      Determinar la transformación del patrón topológico;
3      Aplicar estado de transición;
4  end
```

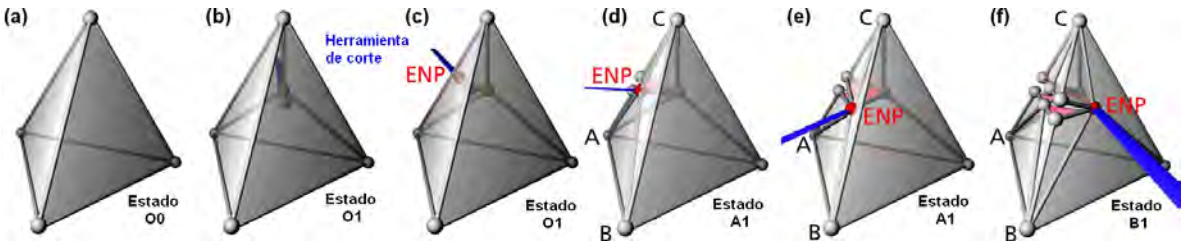


Figura 4.18. *Ejemplo básico del algoritmo de cambio topológico para la simulación de cortes (Bielser[9], 2004).*

corte (línea acotada con una punta), en el estado O0 del diagrama de estados.

b. Una herramienta de corte entra en contacto con una de las caras de tetraedro, en tal configuración que el estado cambia a O1. Se insertan los nuevos elementos en el tetraedro. Se guarda el punto de entrada en un nuevo nodo y se crea otro llamado ENP (*entry point*), que se moverá con la posición de intersección dentro del tetraedro.

c. Dado que no hay nuevas colisiones con las otras aristas o caras, el ENP se mueve sobre la misma cara definiendo una trayectoria de corte.

d. Cuando la herramienta de corte cruza una de las aristas de la cara en cuestión, se efectúa un cambio de estado a A1, se divide la arista y se efectúa una primera subdivisión del tetraedro. Conforme se mueve el ENP sobre la otra cara, se van haciendo más visibles las nuevas aristas, caras y tetraedros.

e. La herramienta de corte se mueve a lo largo de la siguiente cara ABC, conservándose en el mismo estado.

f. Una segunda arista se ve afectada y se repite el mismo procedimiento que en (d). Hay un cambio de estado a B1.

El movimiento del punto ENP puede continuar hacia otra de las aristas (Estado D1), moverse a lo largo de otra cara no vecina (Estado B2), o bien, no registrar más movimientos, lo que significaría que la herramienta de corte ha dejado de afectar la topología del tetraedro; en ese caso se finalizaría en el estado B3.

Este trabajo, por sus alcances, no implementa este método a nivel de software; sin embargo, es importante mencionar que es muy útil para simular cortes en volúmenes de datos, incisiones sobre tejidos (Figura 4.19) u órganos en cirugía virtual usando escalpelo y para esculpir sobre formas geométricas.

Figura 4.19. *Simulación de cortes de tejido usando el algoritmo de cambio topológico (Bielser[9], 2004).*

4.3.6. Comparativa

Una vez analizadas las diferentes metodologías para la simulación computacional de cortes, se presenta la Tabla 4.3, donde se muestra un resumen de las características de cada método.

Criterio	Separación de nodos	Retracción	Eliminación de elementos	Deformación plástica	Cambio topológico
Simulación					
Incisión	Sí	No	No	No	Sí (Ideal)
Extirpación	No	Sí (Ideal)	Sí	No	No
Resección	Sí	No	Sí (Ideal)	Sí	Sí
Geometría					
Elemento básico	Nodo	Nodo	Tetraedro	Nodo	Tetraedro
Malla ideal	Triángulos	Tetraedros	Tetraedros	Tetraedros	Tetraedros
Resolución	Media	Media	Alta	Media	Baja
Complejidad	2	3	1	4	5

Tabla 4.3. Tabla comparativa de los cinco métodos de simulación de cortes para tejidos blandos.

A partir de la tabla, se puede decir que el mejor método para simular incisiones es el de cambio topológico, mientras que para simular extirpación o toma de biopsias de tejido es mejor el método de retracción. Para simular procedimientos de resección, el que se apega más a la realidad es el de eliminación de tetraedros, aún con la desventaja de requerir mallas muy densas.

Finalmente, puede verse que en la tabla se ha clasificado cada método por su nivel de complejidad de implementación, asignando valores del 1 al 5; siendo el método de eliminación de elementos el más sencillo y el de cambio topológico el más complejo de implementar dado el número de operaciones que se deben llevar a cabo para obtener resultados.

CAPÍTULO 5

Experimentos y resultados

Como se mencionó en el objetivo principal al inicio de este trabajo, son tres los temas de estudio, y que a lo largo de todo el contenido se han venido abordando. El primero son las metodologías para la simulación de comportamientos deformables, el segundo la representación geométrica de un objeto mediante su discretización espacial y el tercero las metodologías para la simulación de diversas técnicas de corte sobre tejido simulado. Por ello, este capítulo está dedicado a detallar algunos experimentos realizados tras el estudio de los métodos, y su validación bajo cuatro parámetros principales: funcionalidad, realismo visual, interactividad y fluidez. Elementos fundamentales para toda aplicación que se desarrolla en tiempo real.

De todo el conjunto de métodos en cuanto a objetos deformables y cortes, se han seleccionado los siguientes para su reporte, debido a los resultados interesantes obtenidos:

- Simulación de tejido: Método híbrido de *nVidia PhysX*.
- Simulación de corte: Separación de nodos, retracción, eliminación de elementos y deformación plástica.

Para tal tarea se han desarrollado tres aplicaciones de software y una integración en un simulador de cirugía de próstata (CCADET-UNAM). Cada una de estas aplicaciones pretende ser una herramienta de apoyo para la demostración, enseñanza, investigación o el desarrollo de nuevo software usando los diferentes métodos computacionales. En el caso concreto de la integración en un simulador de entrenamiento, además del propósito computacional, se tiene un propósito médico, en el cual se tiene la premisa de mejorar las habilidades de un aprendiz en las técnicas quirúrgicas con un ambiente virtual. Adicionalmente, se desarrollaron

programas para el procesamiento de imágenes en la reconstrucción de modelos anatómicos, éstos no se abordan completamente en este trabajo por no ser aún lo suficientemente interactivos, sin embargo, se muestran algunos resultados derivados.

Las especificaciones de desarrollo son:

- Lenguajes de programación: *C/C++* (Simulación), *MATLAB Script* y *Java* (estos últimos dos para el procesamiento de imágenes).
- Plataforma: *Windows 7* (32-bits).
- Plataforma de desarrollo: *Visual Studio* 2008 & 2010.
- Bibliotecas para GUI: *Qt v4.7.3*.
- Bibliotecas para visualización: *OpenGL*.
- Motor de física: *nVidia PhysX*.
- Versión del SDK de física: 2.8.1.
- Paquetes adicionales: *MATLAB* (Procesamiento de Imágenes), *ImageJ* (Procesamiento de Imágenes), *NetBeans* (IDE Java), Blender (Modelado 3D basado en superficies).
- Procesamiento para pruebas de simulación experimental: *AMD Athlon II X2 240 Processor@2.8GHz*, 2.0MB Caché.
- Tarjeta gráfica para pruebas de simulación experimental: *MSI N430GT-MD2GD3*, con las siguientes características:
  - Procesador gráfico: *NVIDIA GeForce GT 430*.
  - Memoria: 2GB DDR3 (128 bits).
  - Slot PCI Express/PCI Express 2.0. HDMI/DVI/D-Sub.
  - Soporte: *OpenGL* 4.0, tecnología *NVIDIA CUDA* C/C++, *NVIDIA PhysX*, *DirectX* 11, *Blue-Ray* 3D, entre otros.
- Procesamiento en el simulador de cirugía de próstata: *Sun Microsystems*, *Dual-Core AMD Opteron* 1222, 3.0 GHz, 4 GB RAM, *nVidia Quadro* 3700 con soporte para *PhysX* y *Windows 7* (x64) *Professional Edition*.

Sección 5.1

Software de generación de mallas de tetraedros

El objetivo del desarrollo de este software es disponer de una herramienta para la creación de modelos de prueba para las simulaciones de tejido y corte. En su versión actual, el programa se compone de una interfaz de usuario con *Qt* y una interfaz de visualizacion con *OpenGL*, mostradas en la Figura 5.1.

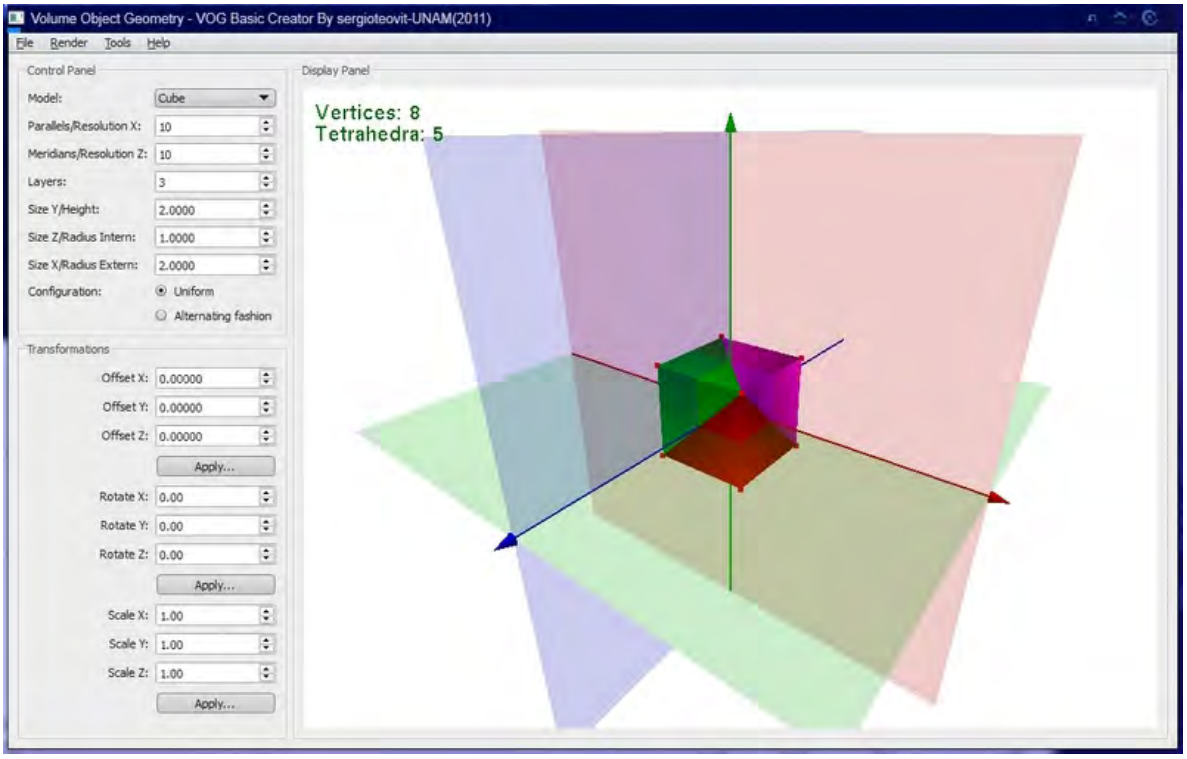


Figura 5.1. *Entorno gráfico del generador de geometrías de volumen.*

Sus principales funcionalidades son:

1. Generación de volúmenes con geometrías básicas: cubo, caja, esfera, cilindro, cardioide 3D y cardioide 3D en coordenadas esféricas.
2. Generación de mallas multiresolución mediante parámetros de subdivisión.
3. Configuración de dimensiones. Permite la definición del tamaño de cada uno de los objetos básicos. Se especifican tres cantidades principales, para objetos planos: tamaños con respecto a los tres ejes coordenados, para objetos tubulares: profundidad, radio interno y radio externo.

4. Selección de modo de generación. Se disponen de los dos modos de disposición de los tetraedros: configuración uniforme y configuración alternada.

5. Transformaciones. Ofrece la posibilidad de transformar la malla bajo tres operaciones básicas: traslación, rotación y escalamiento.

6. Modos de *render*. Mediante el menú de *render*, se puede visualizar la escena con algunos efectos básicos: modo malla de alambre, modo sólido y modo nube de puntos. Por otra parte permite activar o desactivar los planos y ejes coordenados.

7. Visualización de estadísticas. Esta funcionalidad permite obtener el número de vértices, número de tetraedros, tiempo de carga y memoria usada para almacenar una malla.

8. Tetraedrización a partir de nubes de puntos. Se dispone de una herramienta adicional, que se activa mediante el menú *Tools/ImportXYZ Data*, que sirve para generar una malla tubular de tetraedros a partir del método de iso-superficies descrito en el Capítulo 3. La interfaz gráfica se muestra en la Figura 5.2.

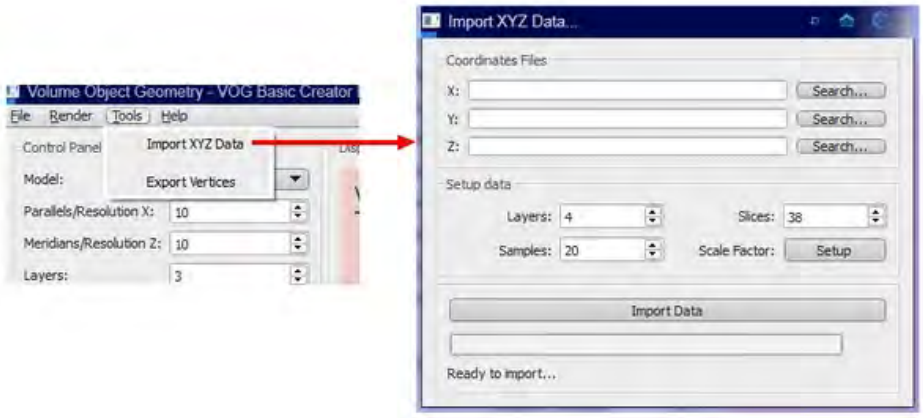


Figura 5.2. *Entorno gráfico del generador de mallas tubulares usando el método de iso-superficies.*

9. Exportar vértices. Los vértices que forman la malla pueden exportarse en formato Wavefront OBJ para su visualización con cualquier paquete de modelado, por ejemplo *Blender*[10] (menú *Tools/Export Vertices*).

10. Cargar y guardar modelos. Mediante el menú *File* es posible almacenar los modelos creados con la interfaz en formato VOG (*Volume Object Geometry*). Asimismo es posible cargar un modelo con el mismo formato. En esta versión únicamente es posible cargar y generar un modelo a la vez.

Para validar la funcionalidad y utilidad del software, se generó un conjunto de mallas a diferentes resoluciones empleando un criterio basado en la resolución espacial. Por otro

lado, para limitar el espacio de geometrías, se seleccionaron cuatro de las seis formas básicas disponibles (cubo, caja, esfera, cilindro, cardioide 3D y cardioide 3D en coordenadas esféricas) y una forma más compleja (modelo de la próstata presentado en uno de los casos de estudio). Las geometrías básicas se generaron a partir de los algoritmos 3 al 8. La geometría anatómica tubular a partir del método de iso-superficies y usando la herramienta descrita en el punto 8. En la Figura 5.3 se muestran algunos ejemplos de las formas de prueba.

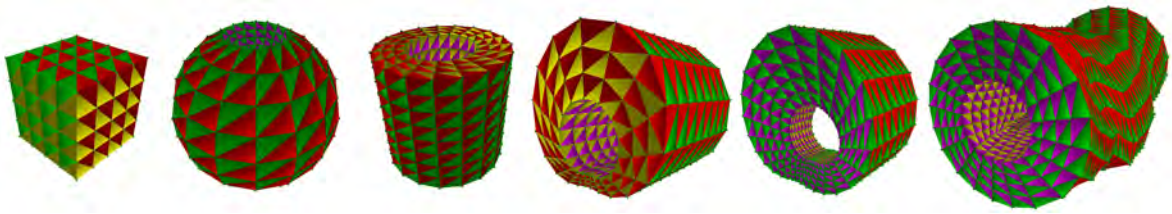


Figura 5.3. *Formas geométricas para validar la funcionalidad del software de generación de mallas y los algoritmos de reconstrucción.*

Todas las formas presentadas tienen asociadas dos propiedades: una dimensión y una resolución espacial. La primera delimita el espacio que ocupa el objeto y la segunda el nivel de detalle que le da forma. Las dimensiones del objeto generalmente no limitan su manejo computacional, debido a que pueden expresarse en términos de una terna de valores flotantes. En cambio, su resolución espacial es mucho más compleja, ya que cualquier método, háblese de deformación, corte o visualización, procesa cada elemento de manera individualizada, agregando un mayor costo computacional. Por tal motivo, el primer punto a analizar, antes de describir el criterio para generar modelos de prueba, es qué resoluciones podemos alcanzar aplicando los diferentes algoritmos.

El siguiente análisis se han dividido en tres partes: geometrías planas, geometrías tubulares básicas y geometrías tubulares anatómicas. El objetivo es fundamentar el criterio para la generación de mallas de prueba basado en la resolución espacial.

5.1.1. Análisis de geometrías planas

La dimensión de este tipo de geometrías está dada por tres cantidades: largo, ancho y alto. Además, en una configuración orientada hacia los ejes y planos coordenados, pueden discretizarse de manera uniforme a lo largo de éstos. En este grupo podemos colocar al cubo y al paralelepípedo rectangular (caja).

Dependiendo de qué parámetro de resolución espacial se manipule para discretizar un objeto plano orientado (R_x, R_y ó R_z), el número de nodos y tetraedros que formen la malla será diferente. Si se parte de una resolución mínima ($R = 1$) para dos de las resoluciones, por ejemplo R_x y R_y, y se varía el valor de R_z en una unidad, las curvas de resolución para

vértices y tetraedros de la malla serán como se muestra en la gráfica de la Figura 5.4a. Ambas corresponden a un incremento lineal. Si ahora se queda fija sólo una de las resoluciones, por ejemplo R_x, mientras que R_y y R_z sufren una variación unitaria equitativa, las curvas de resolución corresponden a un incremento cuadrático, como se muestra en la gráfica de la Figura 5.4b. Finalmente, variando equitativamente los tres valores de resolución R_x, R_y y R_z, se obtiene un incremento cúbico, como se muestra en la gráfica de la Figura 5.4c.

Aquí puede analizarse un primer punto. La distancia que guarda la curva de vértices con respecto a la curva del número de tetraedros, aumenta drásticamente al variar equitativamente las tres componentes de la resolución. Si bien es cierto, que de acuerdo con las ecuaciones para la generación de geometrías planas, hay un primer factor multiplicativo de 5 entre una y otra, no es muy notorio en la gráfica del comportamiento lineal, pero sí lo es con un incremento cúbico. Esto significa que, un primer criterio para definir el nivel de detalle de una malla se basa en la variación e independencia de las componentes de resolución. Esto es, para generar mallas de baja resolución se ajusta conforme a un comportamiento lineal. Si se desean mallas de mediana resolución, se ajusta conforme a un comportamiento cuadrático, mientras que si lo que se desea es una malla de alta resolución, se ajusta sobre un comportamiento cúbico.

5.1.2. Análisis de geometrías tubulares básicas

Las geometrías tubulares o huecas regulares, descritas en el capítulo 3, consisten en: el cilindro, la esfera hueca, el cardioide 3D cilíndrico y el cardioide 3D esférico. Todas ellas tienen la particularidad de estar definidas por un radio interno, un radio externo y en algunos casos una profundidad. Su resolución espacial, al igual que sucede con las geometrías planas, puede analizarse variando sus componentes. En este caso, las componentes de la resolución son: R_θ, R_{radial} y R_h (R_ϕ en casos esféricos). La diferencia radica en el valor inicial de variación, ya que no puede corresponder a 1 en todas las componentes, los valores correctos de inicialización son: $R(R_\theta, R_{radial}, R_h) = (3, 1, 1)$. La Figura 5.5 muestra el comportamiento lineal, cuadrático y cúbico que prevalece en este tipo de geometrías.

5.1.3. Análisis de geometrías tubulares anatómicas

Las geometrías tubulares que corresponden a modelos anatómicos, generadas mediante al algoritmo de iso-superficies, necesitan de un tratamiento distinto en cuanto a las componentes de resolución usadas en los dos análisis anteriores. En este caso la resolución no es tan explícita, pues la información proviene directamente del tratamiento de las imágenes. Así, se necesitan tres datos: el número de imágenes procesadas S, el número de muestras por contorno M y el número de capas L. La resolución espacial inicial para el análisis está dado por $R(S, M, L) = (2, 3, 1)$; que significa que una malla debe estar definida por un mínimo de 2

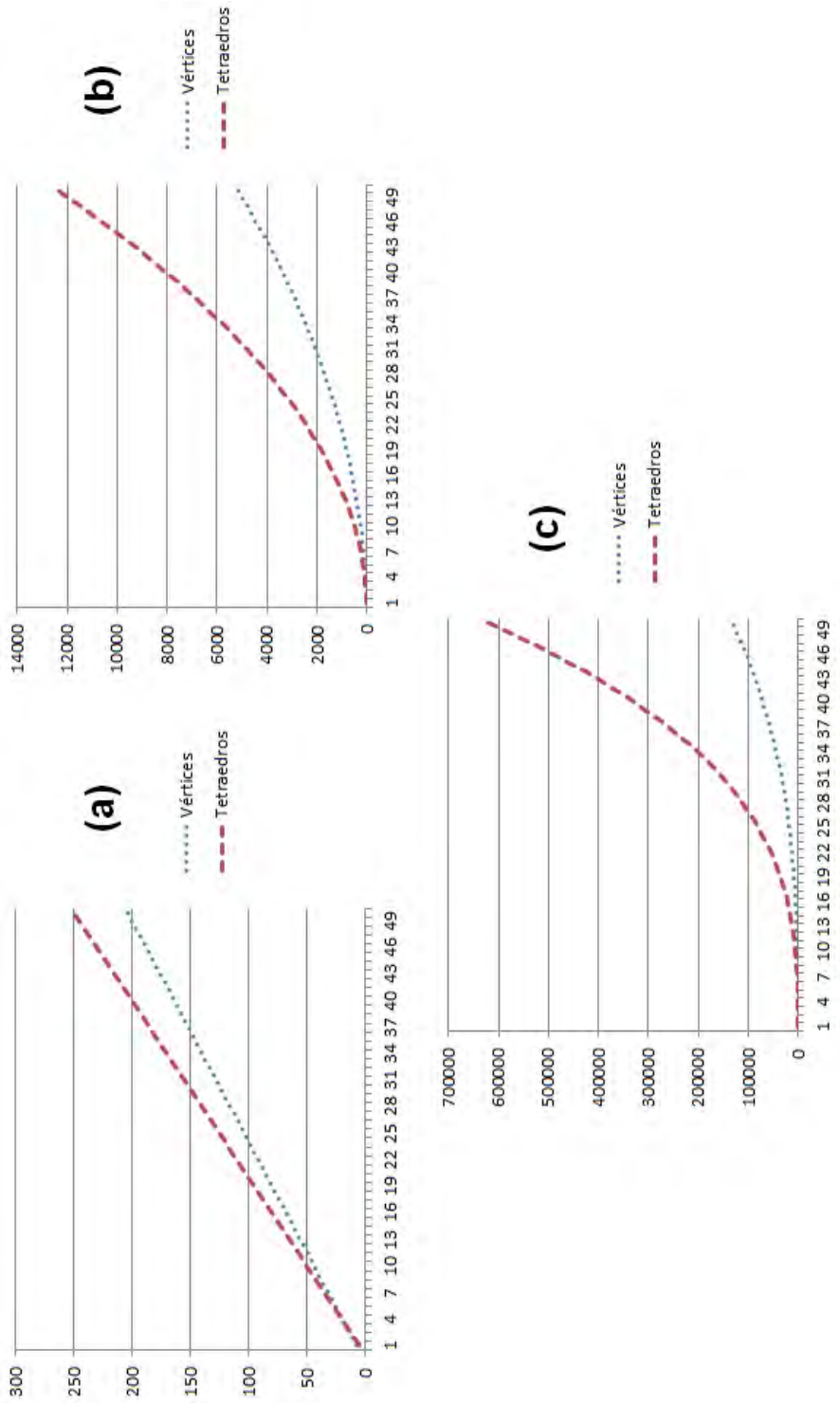


Figura 5.4. *Crecimiento en el número de vértices y tetraedros para geometrías planas. (a) Crecimiento lineal. (b) Crecimiento cuadrático. (c) Crecimiento cúbico.*

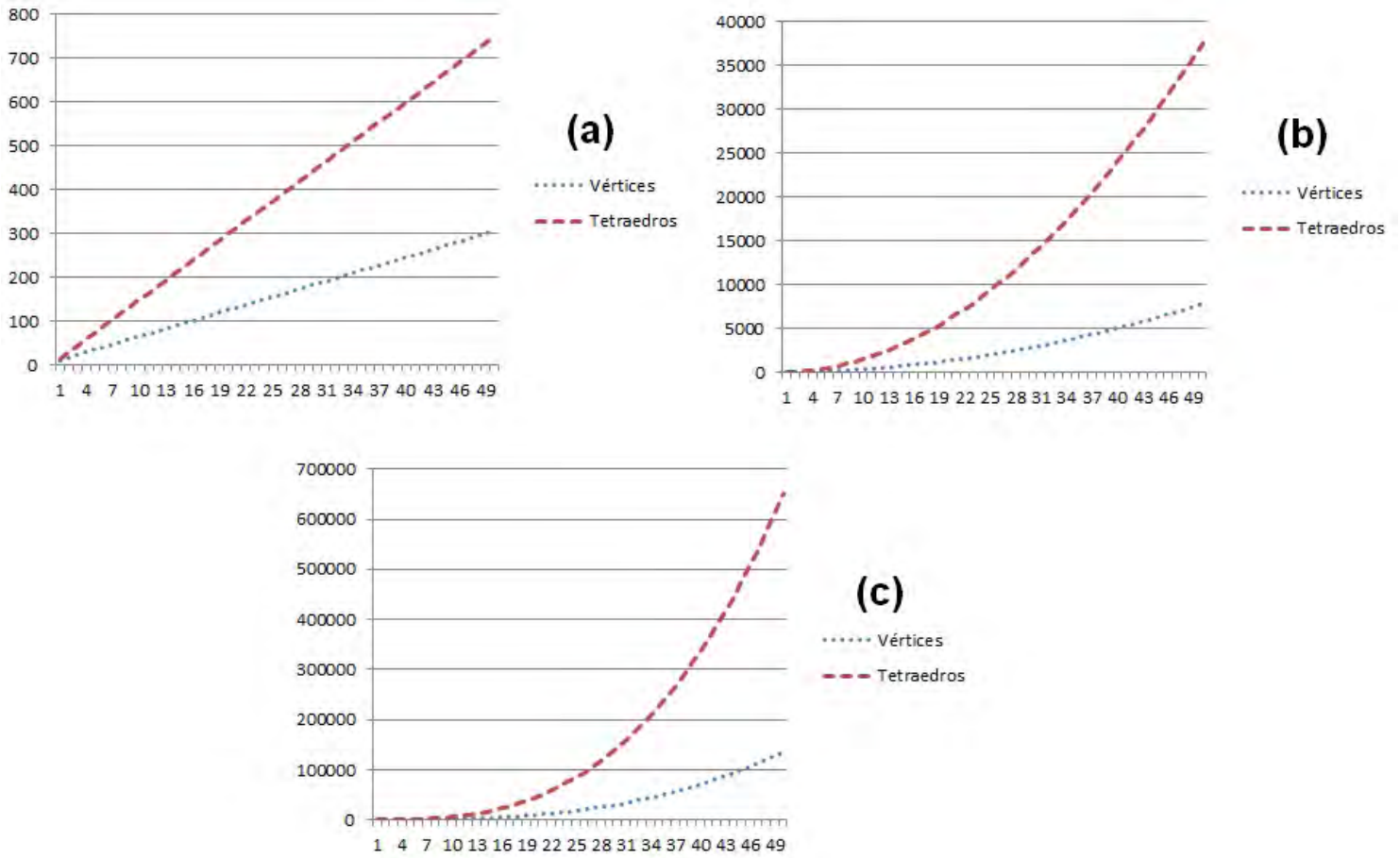


Figura 5.5. *Crecimiento en el número de vértices y tetraedros para geometrías tubulares regulares. (a) Crecimiento lineal. (b) Crecimiento cuadrático. (c) Crecimiento cúbico.*

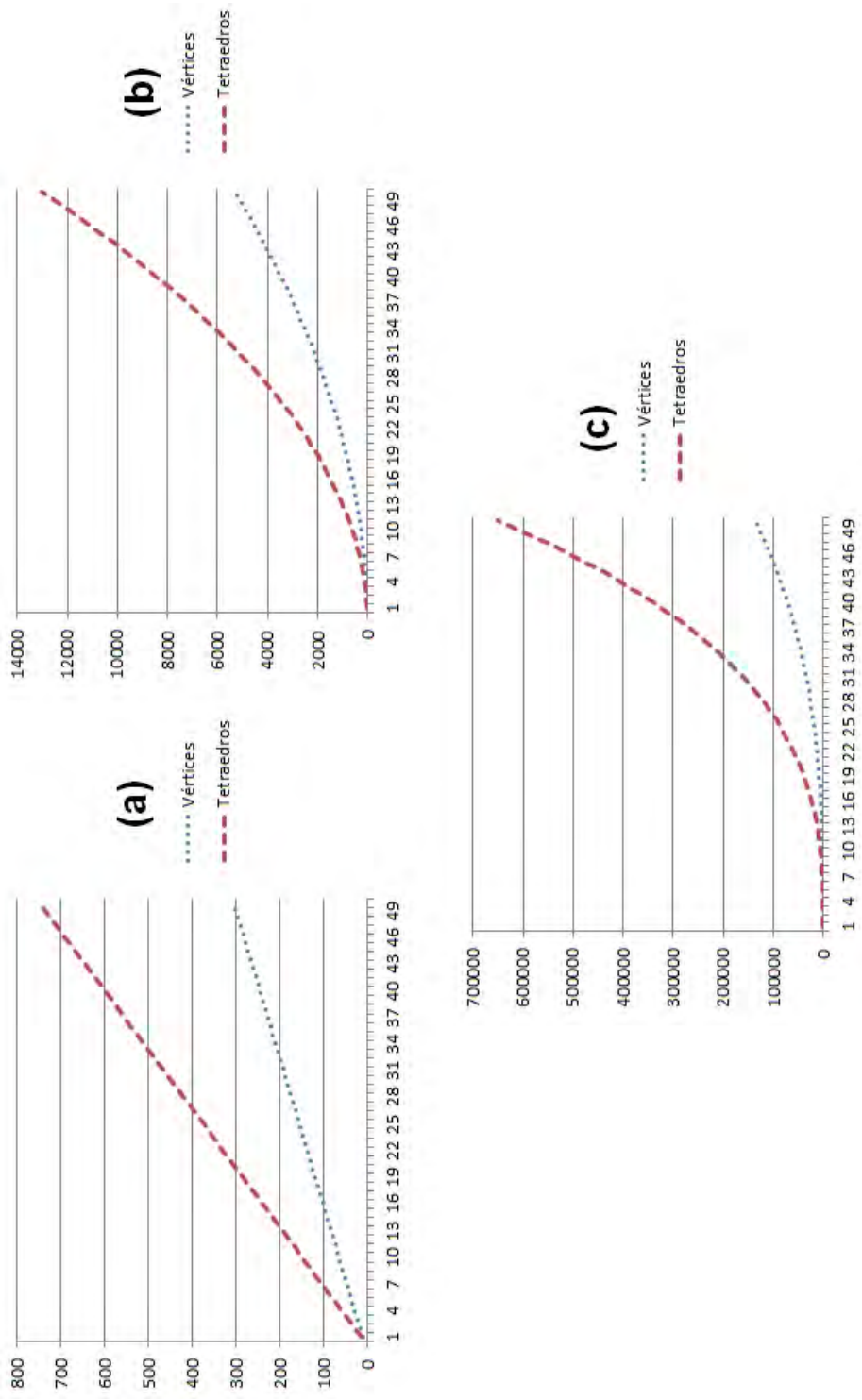


Figura 5.6. *Crecimiento en el número de vértices y tetraedros para geometrías tubulares anatómicas. (a) Crecimiento lineal. (b) Crecimiento cuadrático. (c) Crecimiento cúbico.*

imágenes, 3 muestras por contorno y una capa. Tras aplicar el mismo método descrito en las dos subsecciones anteriores, se tienen los comportamientos de las gráficas de la Figura 5.6.

5.1.4. Criterio para la generación de mallas de prueba

Tomando en cuenta las gráficas de las Figuras 5.4, 5.5 y 5.6, el análisis de la resolución da como resultado que al variar una sola componente obtenemos mallas de baja resolución, al variar dos componentes obtenemos mallas de mediana resolución y al variar todas las componentes obtenemos mallas de muy alta-extrema resolución. Sin embargo, esto no es un criterio suficiente, dado que no toma en cuenta toda la gama de combinaciones posibles, sino casos específicos. Una metodología propuesta para definir un criterio de resolución espacial, se basa en la simulación de casos a partir de valores aleatorios y su representación mediante la media geométrica. Ésta consiste en lo siguiente: por cada componente se generan n números aleatorios que van desde su valor mínimo hasta un valor incremental (inicialmente igual al valor mínimo y hasta el valor de n). Por cada incremento se calcula la media geométrica del conjunto de n valores aleatorios[1]. La media geométrica se define como:

$$\mu_G(\widetilde{x}) = (x_1 x_2 x_3 ... x_n)^{1/n} \tag{5.1}$$

donde $\widetilde{x}$ es el conjunto de valores aleatorios x_i ($i = \{1, ..., n\}$).

Por ejemplo, en el caso de mallas tubulares regulares, cuyas componentes de resolución son: $R(R_\theta, R_{radial}, R_h)$, con valores mínimos $(3, 1, 1)$, se ejecutan los siguientes pasos:

1. Generamos $n = 50$ valores aleatorios por cada componente $\widetilde{R}_1$. Inicialmente el rango de valores aleatorios por componente será $(3 : 3, 1 : 3, 1 : 3)$ (elegimos el valor mayor de la terna como cota superior inicial).
2. Calculamos la media geométrica de $\widetilde{R}_1$.
3. Incrementamos el valor de la cota superior en una unidad: $(3 : 4, 1 : 4, 1 : 4)$. Generamos un nuevo conjunto de valores aleatorios $\widetilde{R}_2$.
4. Calculamos la media geométrica de $\widetilde{R}_2$.
5. Incrementamos el valor de la cota superior en una unidad. Generamos un nuevo conjunto de valores aleatorios $\widetilde{R}_i$.
6. Calculamos la media geométrica $\widetilde{R}_i$ sobre el nuevo conjunto de valores aleatorios.
7. Repetimos los pasos 5 y 6 hasta que $i = n$.

[1] Se emplea la media geométrica porque es una medida menos sensible a los valores extremos.

8. Por cada terna $\widetilde{R}_i(R_\theta, R_{radial}, R_h)$ calculamos el número de vértices y tetraedros asociados.

El resultado de aplicar esta metodología para geometrías tubulares y planas regulares se muestra en las gráficas de la Figura 5.7. La forma de las curvas corresponden a un comportamiento no lineal. Comparando con las gráficas 5.4c y 5.5c, podemos observar que el número de vértices y tetraedros se reduce considerablemente, manteniéndose la forma de las curvas.

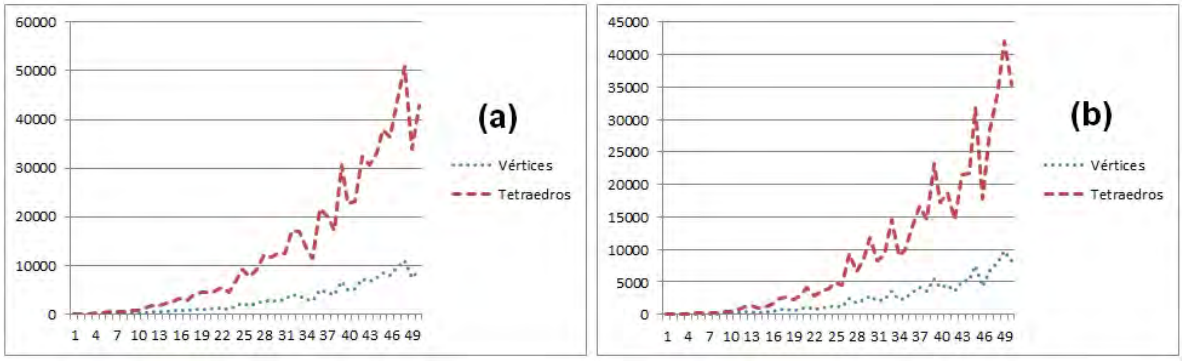


Figura 5.7. *Método de la media geométrica para clasificar mallas de baja, media y alta resolución. (a) Geometrías tubulares regulares. (b) Geometrías planas regulares.*

Podemos comparar también las gráficas de memoria mostradas en la Figura 5.8, en las cuales se observa una disminución en el tamaño de kilobytes (KB) necesario para almacenar la información de una malla. El modelo para determinar este tamaño es:

$$Mem = N_v \cdot 3 \cdot \textbf{size(double)} + N_t \cdot 4 \cdot \textbf{size(int)} \tag{5.2}$$

donde N_v es el número de vértices, N_t es el número de tetraedros, **size(double)** es el número de bytes en un dato tipo *double* (generalmente 8) y **size(int)** es el número de bytes en un dato tipo *int* (generalmente 4). El 3 corresponde a los tres valores flotantes que contienen la información de la posición de cada nodo, y el 4 son los cuatro índices que definen un tetraedro.

5.1.5. Generación de mallas de prueba

Basados en el criterio anterior, se proponen las Tablas 5.1 y 5.2 para clasificar una malla de acuerdo con su resolución espacial. En modelos tubulares anatómicos se aplicó un criterio diferente, basado también en la resolución, pero más enfocado al caso de aplicación.

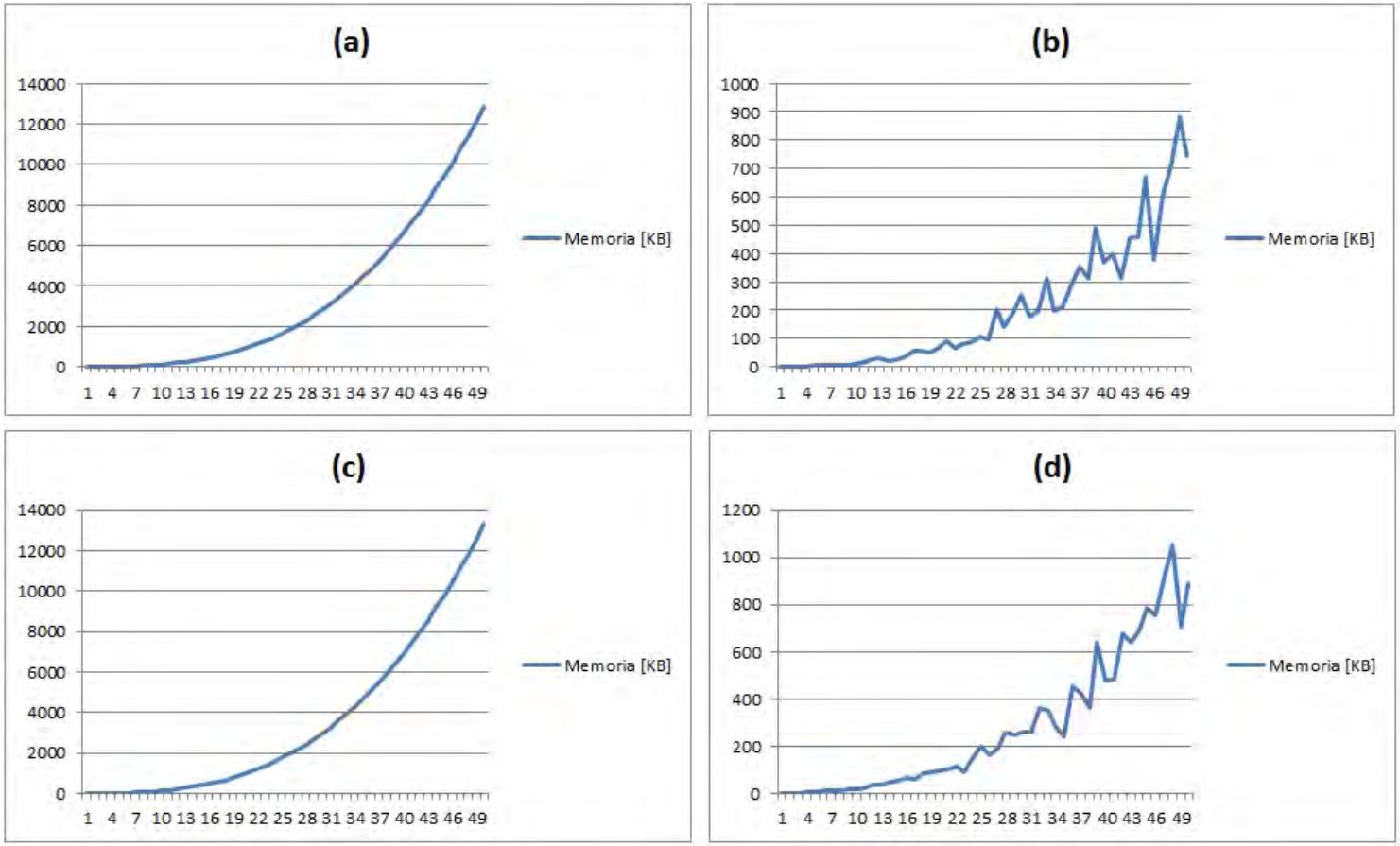


Figura 5.8. *Gráficas de memoria para mallas de tetraedros usando un criterio de resolución espacial. (a) Memoria para geometrías planas usando el método de variación incremental. (b) Memoria para geometrías planas usando el método de la media geométrica. (c) Memoria para geometrías tubulares usando el método de variación incremental. (d) Memoria para geometrías tubulares usando el método de la media geométrica.*

Resolución	Rango de cotas	Número de vértices*	Número de tetraedros*
Baja	1 – 10	8 – 200	5 – 500
Media Baja	10 – 20	200 – 1000	500 – 3000
Media	20 – 30	1000 – 3000	3000 – 10000
Media Alta	30 – 40	3000 – 5000	10000 – 20000
Alta	40 – 50	5000 – 10000	20000 – 40000

Tabla 5.1. Clasificación del nivel de detalle para mallas de geometría plana. *(Aproximaciones experimentales, el número puede ser mayor o menor sin cambios bruscos entre estos intervalos).

Resolución	Rango de cotas	Número de vértices*	Número de tetraedros*
Baja	3 – 12	12 – 300	15 – 1000
Media Baja	12 – 22	300 – 1000	1000 – 5000
Media	22 – 32	1000 – 3000	5000 – 10000
Media Alta	32 – 42	3000 – 5000	10000 – 30000
Alta	42 – 52	5000 – 10000	30000 – 50000

Tabla 5.2. Clasificación del nivel de detalle para mallas de geometría tubular. *(Aproximaciones experimentales, el número puede ser mayor o menor sin cambios bruscos entre estos intervalos).

Un experimento aplicando el método de la media geométrica, dió como resultado las componentes de resolución mostradas en la Tabla 5.3. Estos valores se usaron para generar 20 mallas de prueba, mostradas en la Figuras 5.9-5.10, y cuyas características se condensan en la Tabla 5.4.

Mallas de geometría plana					
Resolución	Cota	(R_x,R_y,R_z)	Número de vértices	Número de tetraedros	Memoria (KB)
Baja	5	$(3,3,3)$	64	135	3
Media Baja	15	$(6,6,6)$	343	1080	24
Media	25	$(10,11,9)$	1320	4950	108
Media Alta	35	$(11,15,12)$	2496	9900	213
Alta	45	$(22,17,17)$	7452	31790	671
Mallas de geometría tubular					
Resolución	Cota	$(R_\theta, R_{radial}, R_h)$	Número de vértices	Número de tetraedros	Memoria (KB)
Baja	6	$(5,3,3)$	80	225	5
Media Baja	16	$(9,8,6)$	567	2160	47
Media	26	$(14,10,10)$	1694	7000	149
Media Alta	36	$(15,12,15)$	3120	13500	284
Alta	46	$(23,17,17)$	7452	33235	693

Tabla 5.3. Experimento para determinar las resoluciones espaciales de mallas de prueba.

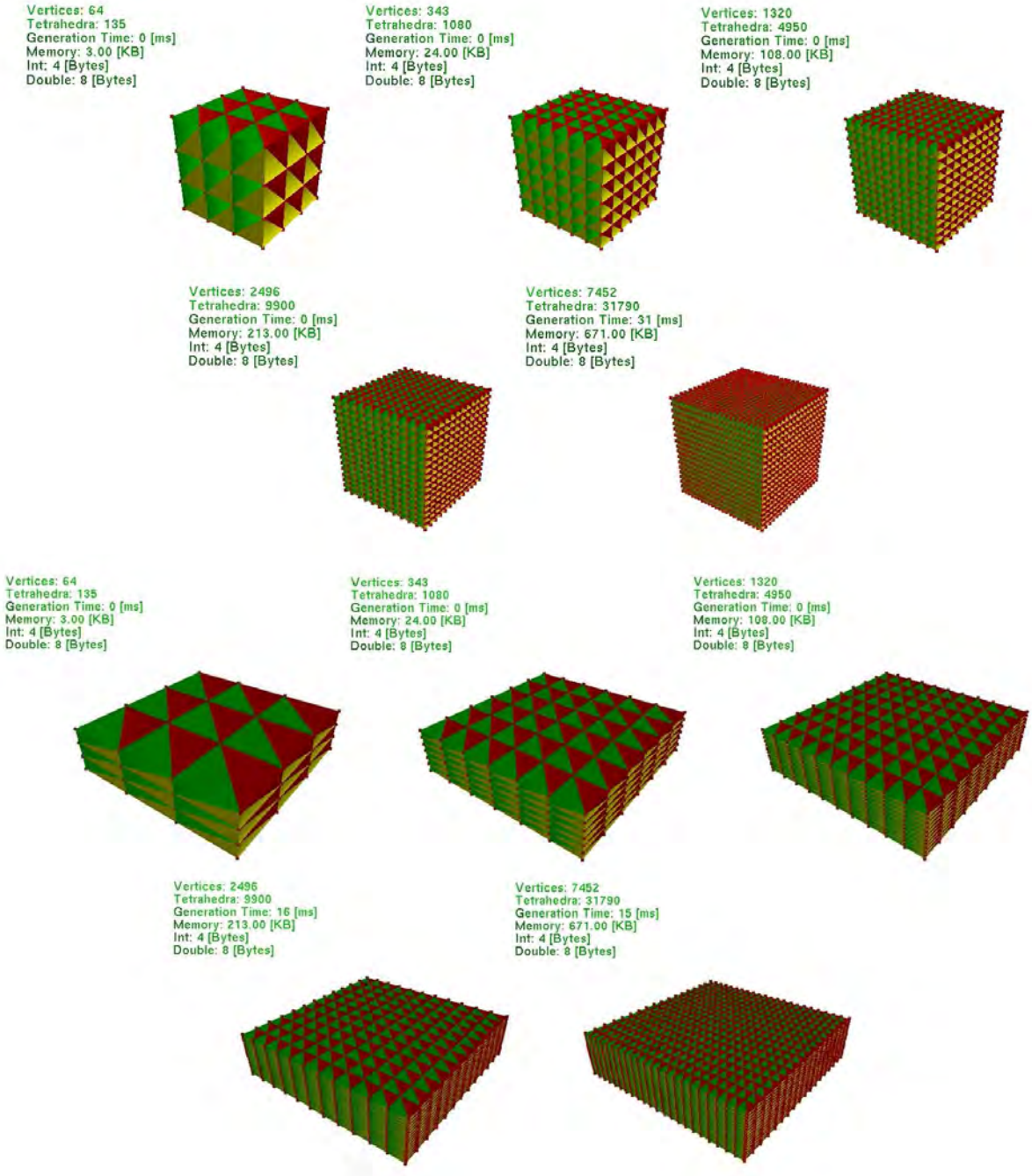


Figura 5.9. *Mallas planas de prueba.*

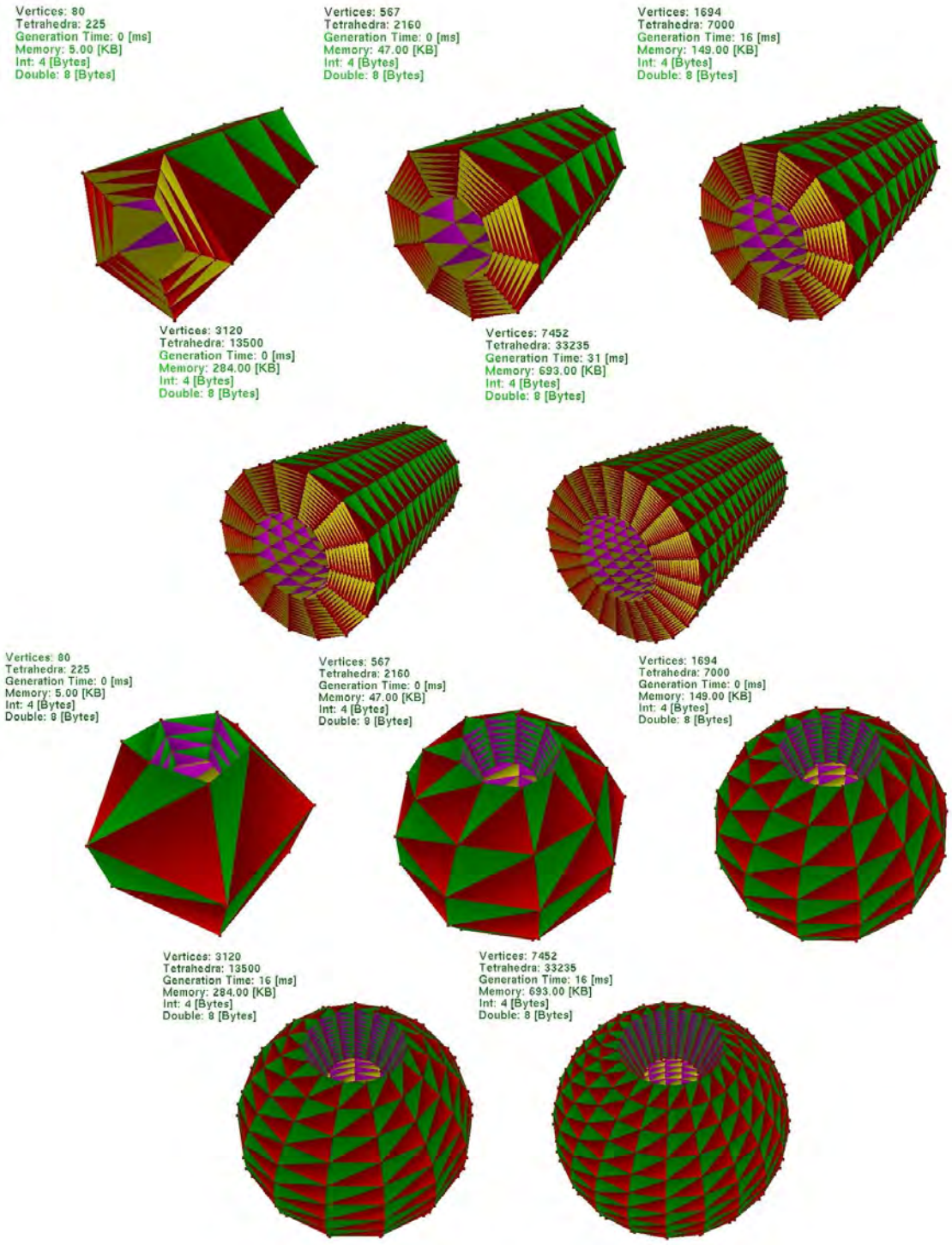


Figura 5.10. *Mallas tubulares de prueba.*

Forma	Resolución	N_v	N_t	$t_{construcción}[mseg]$	Memoria [KB]	$t_{carga}[mseg]$	Tamaño del archivo VOG [KB]
Cubo	(3, 3, 3)	64	135	< 1	3	< 1	4
Cubo	(6, 6, 6)	343	1080	< 1	24	16	29
Cubo	(10, 11, 9)	1320	4950	< 1	108	109	134
Cubo	(11, 15, 12)	2496	9900	< 1	213	249	286
Cubo	(22, 17, 17)	7452	31790	31	671	670	964
Caja	(3, 3, 3)	64	135	< 1	3	< 1	4
Caja	(6, 6, 6)	343	1080	< 1	24	16	29
Caja	(10, 11, 9)	1320	4950	< 1	108	109	134
Caja	(11, 15, 12)	2496	9900	16	213	203	286
Caja	(22, 17, 17)	7452	31790	15	671	655	964
Cilindro	(5, 3, 3)	80	225	< 1	5	16	6
Cilindro	(9, 8, 6)	567	2160	< 1	47	47	59
Cilindro	(14, 10, 10)	1694	7000	16	149	156	194
Cilindro	(15, 12, 15)	3120	13500	< 1	284	296	397
Cilindro	(23, 17, 17)	7452	33235	31	693	686	1,008
Esfera	(5, 3, 3)	80	225	< 1	5	16	7
Esfera	(9, 8, 6)	567	2160	< 1	47	47	61
Esfera	(14, 10, 10)	1694	7000	< 1	149	156	205
Esfera	(15, 12, 15)	3120	13500	16	284	281	404
Esfera	(23, 17, 17)	7452	33235	16	693	702	1,013

Tabla 5.4. Resumen de estadísticas de las mallas de prueba. N_v: Número de vértices. N_t: Número de tetraedros. $t_{construcción}$: Tiempo que toma el algoritmo de generación para construir la malla. t_{carga}: Tiempo que toma el algoritmo de carga del archivo *VOG*. Cubo: $20.0 \times 20.0 \times 20.0$ [mm]. Caja: $20.0 \times 10.0 \times 20.0$ [mm]. Cilindro: $h = 100.0$, $R_i = 5$ y $R_e = 20.0$ [mm]. Esfera: $R_i = 8$ y $R_e = 20$ [mm].

Sección 5.2

Software de simulación de tejido blando y caracterización

Una vez generado un conjunto de prueba, se requiere de un software para la simulación de objetos deformables usando uno de los métodos computacionales abordados en el Capítulo 2. Dado que *PhysX* de *nVidia* ya posee una buena implementación de un método híbrido basado en posiciones para simular cuerpos blandos, y además relaciona las ideas del modelo de masas-resortes con una técnica FEM, entre otras ventajas, se decidió experimentar con el SDK y programar el entorno gráfico mostrado en la Figura 5.11.

Las funcionalidades del software son:

- Simulación de objetos deformables con mallas de tetraedros. Permite la carga de archivos con especificación *VOG*, propuesta en el Capítulo 3.
- Configuración de parámetros físicos de la malla deformable. Permite especificar en tiempo real:
  - Rigidez del volumen (*Volume stiffness*). Puede tomar valores de entre 0 y 1. Define un coeficiente de deformación global, esto es, qué tanto se propaga la deformación sobre todo el volumen al aplicar una fuerza externa.

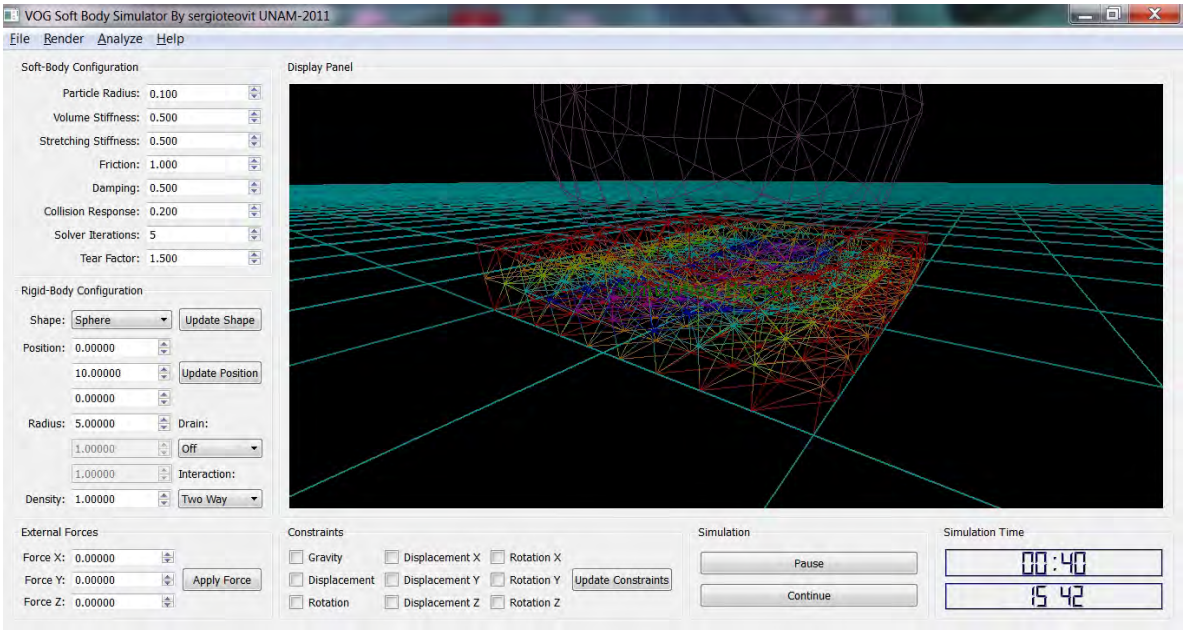


Figura 5.11. *Entorno gráfico del software de simulación de objetos deformables.*

- Rigidez del resorte (*Stretching stiffness*). Puede tomar valores de entre 0 y 1. Define un coeficiente de deformación local.
- Fricción (*Friction*). Puede tomar valores de entre 0 y 1. Define el coeficiente de fricción dinámica del material.
- Amortiguamiento (*Damping*). Puede tomar valores de entre 0 y 1. Define el coeficiente de restitución del material, es decir, qué tan rápida es la estabilización tras un proceso de deformación.

■ Configuración de parámetros de simulación de la malla deformable. Estos son:

- Radio de la partícula (*Particle radius*). Usado para cálculos de colisiones. Su valor depende del tamaño de la geometría. Se prefieren valores pequeños para visualizar mejor un contacto entre un objeto rígido y el objeto deformable.
- Coeficiente de respuesta ante una colisión (*Collision response*).
- Iteraciones (*Solver Iterations*). Número de iteraciones para resolver el sistema de ecuaciones de restricciones, descrito en el Algoritmo 1.
- Factor de rasgado (*Tear factor*). Factor que define qué tanto puede estirarse un resorte con respecto a su longitud inicial. Se emplea para la simulación de cortes usando el método de retracción.

- Simulación de objetos rígidos. Se dispone de geometrías básicas: caja, esfera, cápsula, pirámide, entre otras.
- Configuración de parámetros de objeto rígido. Se pueden definir la posición espacial, dimensiones y densidad. Además es posible configurar si el objeto funciona como un drenador (simulación de corte por el método de eliminación de elementos), o si el objeto es afectado por las fuerzas de restitución del objeto deformable cuando hay una interacción. Una interacción en una dirección únicamente afectará al cuerpo deformable cuando hay un contacto. Una interacción en dos direcciones afecta a ambos.
- Adición de fuerzas externas configurables o constantes mediante teclado.
- Restricciones globales sobre objetos rígidos. Es posible especificar que no afecten ciertas fuerzas al movimiento del objeto, como la gravedad. También es posible restringir el movimiento lineal y rotacional, en todos, o en una dirección en específico.
- Pausar o continuar la simulación, muy útil en el análisis de resultados.
- Visualización del tiempo de simulación y hora del sistema.
- Visualización gráfica con *OpenGL* de la escena.
- Modos de *render*. Al igual que la aplicación para generar mallas, se dispone del modo malla de alambre (mostrado en la figura del entorno), modo sólido y modo nube de puntos.
- Visualización de estadísticas. Activando este modo se puede obtener información de la malla deformable, como número de vértices y número de tetraedros, densidad, promedio de deformación, factor de deformación, modo de cálculo (hardware o software), entre otros. También es posible visualizar las propiedades de los objetos rígidos, como velocidad, aceleración y fuerzas actuando sobre ellos.

  El promedio del campo de deformación $\underline{U}$ se calcula sumando el módulo de cada vector de deformación $\|\underline{u}_i\| = \|\underline{x}_i^1 - \underline{x}_i^0\|$ entre el total de vértices afectados $N_{\underline{U}}$, donde $\underline{x}_i^1$ y $\underline{x}_i^0$ son la posición deformada y no deformada de cada vértice. Esto es,

$$\mu_{\underline{U}} = \frac{\sum_i \|\underline{u}_i\|}{N_{\underline{U}}} \tag{5.3}$$

  El factor de deformación η es el número de vértices afectados por la deformación entre el total de vértices de la malla.

$$\eta = \frac{N_{\underline{U}}}{N_v} \tag{5.4}$$

  Un factor de 1.0 indica que todos los vértices estás siendo afectados por la deformación.
- Visualización de gradientes de la deformación. Activando este modo, se pueden visualizar los vectores con la dirección del desplazamiento de cada vértice de la malla con respecto a su posición original.

- Visualización de cuatro modos de textura 3D. Mediante esta función se visualiza la malla con diferentes texturas volumétricas. Con cada una de ellas se tienen efectos interesantes sobre la deformación. Se definen dos texturas para geometrías planares, una para geometrías tubulares y una textura que simula tejido.

- Texturizado dinámico y estático. Mediante esta función las coordenadas de textura se calculan usando los vértices originales de la malla deformable o los vértices deformados. Usando los vértices originales, la textura se deformará con el modelo (dinámica). En el otro caso, la textura permanece estática, pudiendo hacer análisis de profundidad de la deformación combinando con alguno de los modos de visualización del punto anterior.

  Las coordenadas de textura dinámica $\underline{t}_D(r, s, t)$ ($\{r, s, t\} \in [0...1]$) están dadas por:

$$\underline{t}_D = \left(\frac{\underline{x}^1_{i.x} - \underline{B}_{min.x}}{\underline{B}_{max.x} - \underline{B}_{min.x}}, \frac{\underline{x}^1_{i.y} - \underline{B}_{min.y}}{\underline{B}_{max.y} - \underline{B}_{min.y}}, \frac{\underline{x}^1_{i.z} - \underline{B}_{min.z}}{\underline{B}_{max.z} - \underline{B}_{min.z}} \right) \tag{5.5}$$

  Las coordenadas de textura estática $\underline{t}_S(r, s, t)$ ($\{r, s, t\} \in [0...1]$) están dadas por:

$$\underline{t}_S = \left(\frac{\underline{x}^0_{i.x} - \underline{B}_{min.x}}{\underline{B}_{max.x} - \underline{B}_{min.x}}, \frac{\underline{x}^0_{i.y} - \underline{B}_{min.y}}{\underline{B}_{max.y} - \underline{B}_{min.y}}, \frac{\underline{x}^0_{i.z} - \underline{B}_{min.z}}{\underline{B}_{max.z} - \underline{B}_{min.z}} \right) \tag{5.6}$$

  donde $\underline{B}_{min}$ y $\underline{B}_{max}$ son las componentes mínima y máxima de la caja que encierra a todos los vértices de la malla, *AABB* (*Axis Aligned Bounding Box*).

Algunos ejemplos de la visualización de estas funcionalidades se muestran en la Figura 5.12.

5.2.1. Caracterización del modelo de deformación

El objetivo del software es caracterizar el modelo de deformación de *PhysX* para generar un conjunto de valores útiles en la simulación de comportamiento de tejido blando. Aquí surge un primer requisito, se requiere de un modelo teórico o experimental real para validar el método. Actualmente son pocos los grupos de investigación encargados de estudiar el comportamiento de tejido vivo, sin embargo, la mayoría de ellos concluye que se comporta de manera no lineal, es anisotrópico, viscoelástico y en algunos casos viscoplástico (para mayor detalle ver el Apéndice B). La Figura B.5 muestra el comportamiento no lineal del tejido blando.

Por lo tanto, para aproximar un objeto deformable al comportamiento no lineal de un tejido, se realizaron primero varios experimentos usando las geometrías de prueba, variando los parámetros de simulación y comparando visualmente el efecto provocado. Posteriormente se aplicó un método de calibración por compresión, consistente en aplicar una fuerza externa

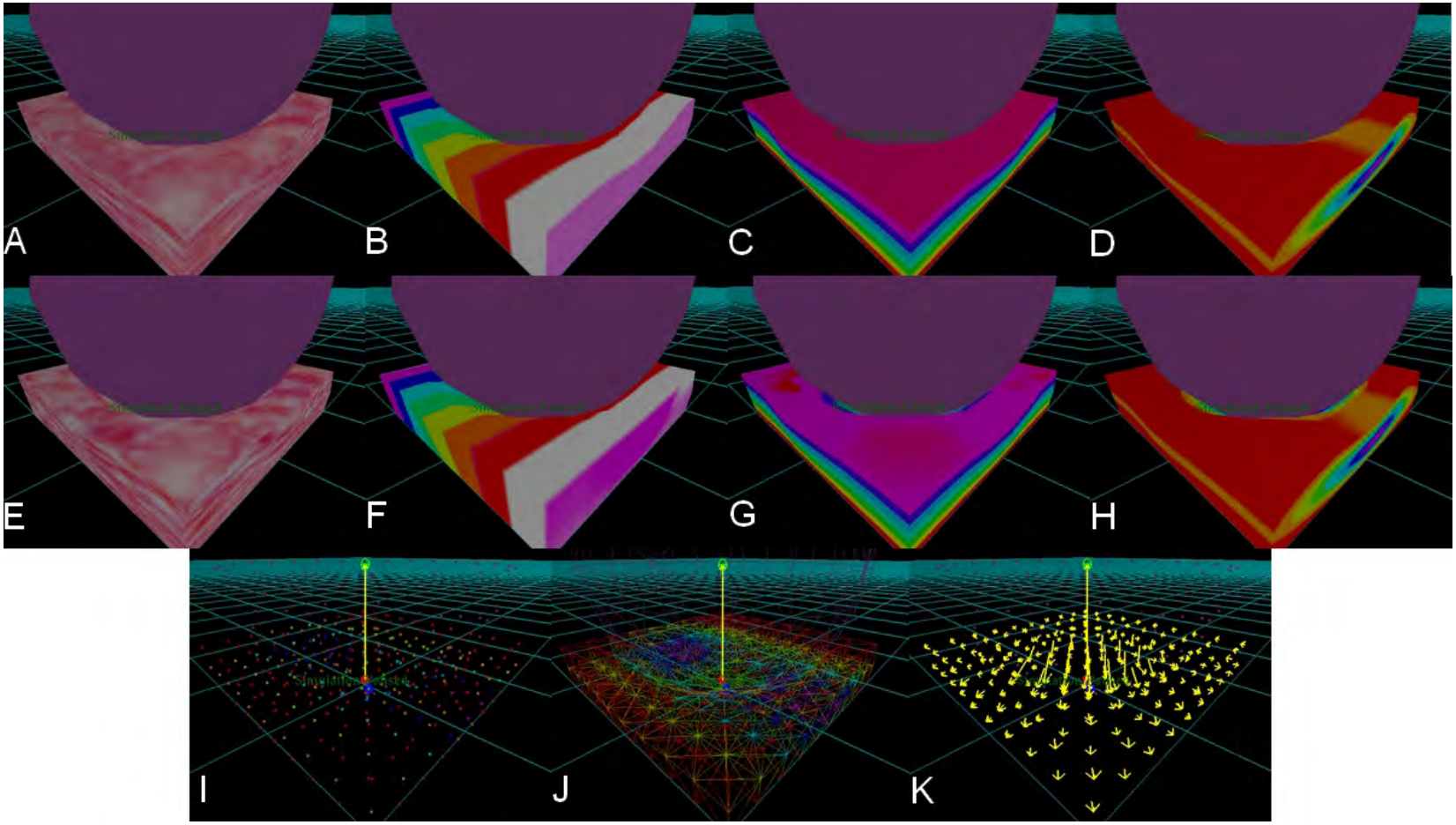


Figura 5.12. *Modos de render y visualización del entorno gráfico para simular objetos deformables. (A,E) Comparativa entre texturizado estático y dinámico con textura de tejido. (B,F) Comparativa entre texturizado estático y dinámico con textura plana tipo 1. (C,G) Comparativa entre texturizado estático y dinámico con textura plana tipo 2. (D,H) Comparativa entre texturizado estático y dinámico con textura tubular. (I) Modo nube de puntos. (J) Modo malla de alambre. (K) Visualización del campo de deformación.*

perpendicular a su superficie, y tras llegarse al equilibrio, se mide la deformación promedio. Así, se busca que al graficar la deformación con respecto al esfuerzo aplicado, se encuentre un comportamiento parecido al mostrado en la Figura B.5.

En *PhysX*, una forma de provocar la compresión de un objeto deformable es mediante la construcción de un objeto rígido a una distancia suficiente como para que, tras iniciar la simulación, éste caiga por efecto de la gravedad. Cuando el objeto rígido comprime al objeto deformable, hay un efecto de reacción que hace que el objeto deformable empuje al objeto rígido en dirección contraria de la fuerza aplicada. Así ocurre hasta que se llega a un punto de estabilización o equilibrio entre los dos objetos. Cuando esto ocurre, se dice que el objeto deformable ha llegado a su valor máximo de compresión. La Figura 5.13 muestra la idea general del método aplicado sobre el cubo deformable de $20 \times 20 \times 20$ [mm] a baja resolución.

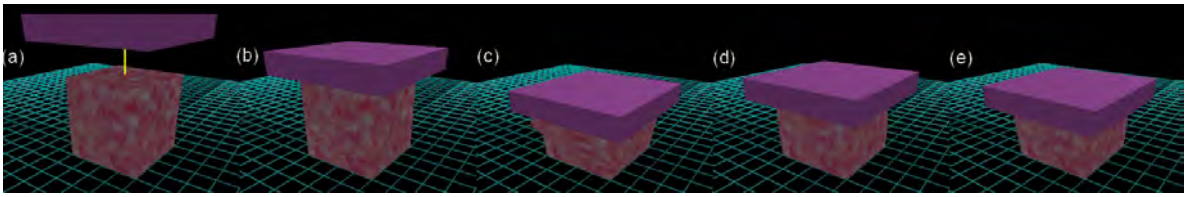


Figura 5.13. *Método de compresión de un objeto deformable. (a) No hay contacto entre el objeto rígido y el objeto deformable. (b) Ocurre un contacto, iniciando la compresión del objeto deformable. (c) Se comprime el objeto deformable hasta que sus fuerzas internas son suficientes como para empujar al objeto rígido en dirección contraria. (d) El objeto rígido es empujado en dirección opuesta a la compresión. (e) Tras un lapso de tiempo el objeto deformable llega a su compresión máxima dada la fuerza impresa.*

Dado que medir el esfuerzo aplicado sobre una superficie usando el método anterior implica mayor cantidad de cálculos durante una simulación, es posible usar un parámetro proporcional, en este caso la masa del objeto rígido; a mayor masa, mayor es la fuerza de gravedad que se imprime sobre el objeto deformable. Este valor es manipulable en *PhysX* por medio de la densidad, y dado que está implícito dentro del ciclo de simulación, no es necesario efectuar mayores cálculos. Así, como la única fuerza que se imprime es la de gravedad, la fuerza total es igual a la masa del objeto rígido (variable) por el valor de la aceleración de la gravedad (constante e igual a $\underline{g} = (0.0, -9.81, 0.0)[m/seg]$).

Por otra parte, en el objeto deformable, el parámetro que afecta de manera directa la deformación es el de rigidez del resorte (*stretching stiffness*). Al variar la rigidez global (*volume stiffness*) únicamente se afecta la propagación de la deformación desde el punto de contacto hasta los límites, por tanto un valor medio parece dar buenos resultados. En cuanto a los otros parámetros, la fricción permite que los objetos rígidos que interactúan sobre la superficie de la malla deformable se resbalen, por lo que para evitar este efecto se usó un valor

de 1.0. El coeficiente de amortiguamiento (*damping*) permite que se tenga control sobre la restitución de la malla una vez que se ha deformado, la experimentación mostró que un valor medio era suficiente para ver un efecto realista.

De las mallas de prueba, las que mostraron ser útiles para fines de calibración usando el método por compresión fueron el cubo y la caja, de baja y media baja resolución, en ambos casos. Las geometrías de media, media alta y alta resolución resultaron no muy útiles para estos efectos, debido que mostraron inestabilidades al probar con valores de rigidez promedio. En la Figura 5.14 se muestran algunos efectos observados en las diferentes mallas de prueba. Las geometrías tubulares no se usaron en este proceso debido a que su superficie no es uniforme como para poder aplicar una compresión y medir correctamente las deformaciones.

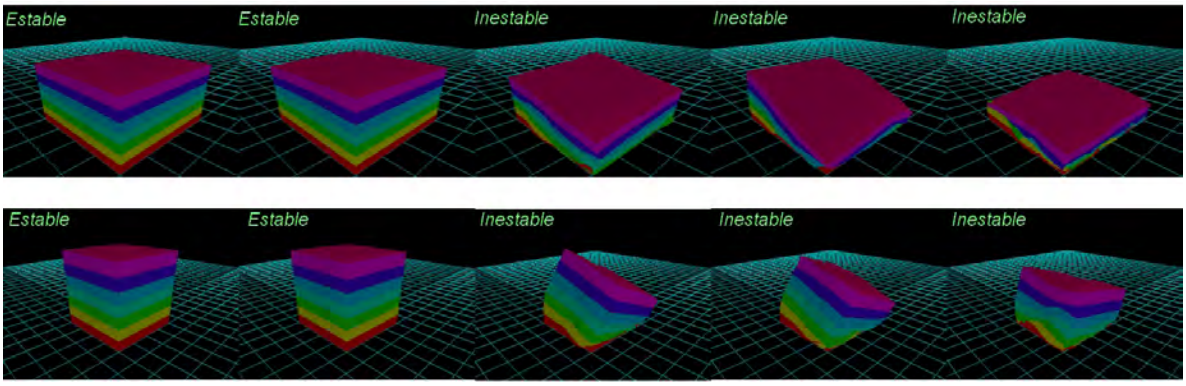


Figura 5.14. *Prueba de estabilidad con valores de rigidez medio sobre las mallas planares de prueba. De izquierda a derecha, mallas de baja a alta resolución. Rigidez del volumen*= 0.5*, Rigidez del resorte*= 0.5*, Fricción*= 1.0 *y Amortiguamiento*= 0.5.

Una vez definidas las cuatro mallas de prueba estables, se aplicó el método por compresión para valores de rigidez del resorte de entre 0.1 a 1.0, con intervalos de 0.1. La variación del esfuerzo aplicado se controló variando proporcionalmente el valor de densidad del objeto rígido. Se usó un objeto rígido con geometría de caja cuadrangular con superficie plana de 24×24 [mm] (un poco mayor a las dimensiones de la superficie de las mallas deformables; esto para asegurar cubrir toda el área de contacto), espesor de 5 [mm], posición inicial al doble del espesor del objeto deformable, por encima de la zona de contacto, densidad variable, movimiento rotacional restringido en todos los ejes, movimiento lineal sólo sobre el eje Y (dirección de la fuerza de gravedad) e interacción en ambas direcciones. Las tablas y gráficas, resultado de un total de 400 experimentos, se muestran en las Figuras 5.15 a 5.18.

De los resultados se pueden hacer las siguientes observaciones:

- A mayor resolución, el objeto deformable es más estable y menos vulnerable a penetraciones por parte del objeto rígido y de si mismo.

Objeto Rígido		Deformación promedio (Objeto Deformable) [mm]									
Densidad	Masa [g]	K=0.1	K=0.2	K=0.3	K=0.4	K=0.5	K=0.6	K=0.7	K=0.8	K=0.9	K=1.0
0.1	288	1.423	0.900	0.674	0.540	0.447	0.376	0.315	0.262	0.207	0.089
0.2	576	2.360	1.669	1.281	1.046	0.878	0.745	0.631	0.525	0.415	0.159
0.3	864	2.812	2.341	1.914	1.526	1.289	1.102	0.939	0.785	0.623	0.232
0.4	1152	3.206	2.783	2.480	2.156	1.827	1.460	1.293	1.044	0.831	0.307
0.5	1440	3.507	3.209	2.982	2.632	2.448	2.261	2.078	1.307	1.041	0.383
0.6	1728	1.517	3.587	3.249	3.030	2.841	2.620	2.270	2.062	1.259	0.459
0.7	2016	2.771	1.457	1.365	3.387	3.237	2.990	2.680	2.294	1.661	0.536
0.8	2304	2.882	1.457	1.365	1.341	1.945	3.351	3.065	2.632	2.331	0.615
0.9	2592	2.692	1.611	1.365	1.341	1.325	1.313	3.465	2.952	2.607	0.694
1.0	2880	2.551	1.457	1.365	1.341	1.325	1.313	1.607	3.348	2.859	0.775

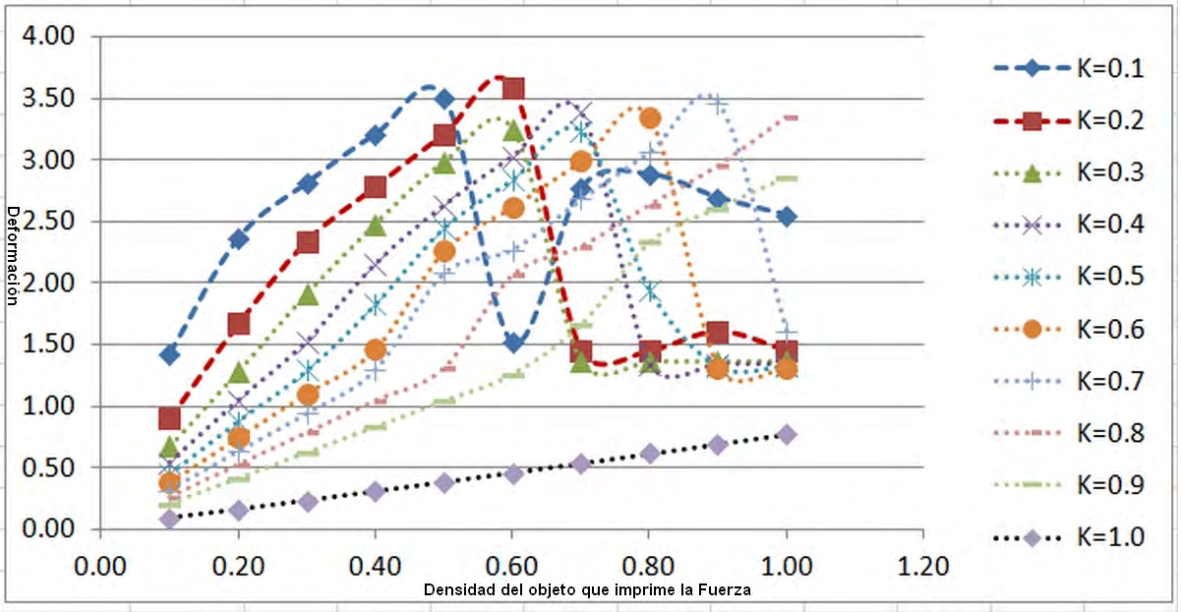


Figura 5.15. *Resultados de una prueba de compresión sobre una caja de* $20 \times 10 \times 20$ *[mm], con resolución* $R(3, 3, 3)$*, densidad: 1.0, radio de la partícula: 0.1, rigidez del volumen: 0.5, rigidez del resorte variable (K), fricción dinámica: 1.0, amortiguamiento: 0.5, respuesta a la colisión: 0.2, número de iteraciones: 5 y plano de mundo activado. Se observan dos curvas principales de comportamiento no lineal (*$K = 0.1, K = 0.2$*). Los picos indican la máxima compresión alcanzada, siendo los valores consecuentes, estados de inestabilidad (visualmente se observa que el objeto rígido penetra la malla deformable). También se observa una transición entre comportamiento no lineal a lineal suave entre los valores* $K = 0.3$ *a* $K = 0.9$*, relativo al rango seleccionado. El tiempo de estabilización para alcanzar el equilibrio entre los dos objetos osciló entre 1 y 1.5 minutos, aproximadamente.*

Objeto Rígido		Deformación promedio (Objeto Deformable) [mm]									
Densidad	Masa [g]	K=0.1	K=0.2	K=0.3	K=0.4	K=0.5	K=0.6	K=0.7	K=0.8	K=0.9	K=1.0
0.1	288	0.804	0.533	0.413	0.339	0.286	0.244	0.207	0.174	0.139	0.065
0.2	576	1.282	0.886	0.700	0.581	0.492	0.420	0.357	0.298	0.236	0.096
0.3	864	1.701	1.215	0.978	0.820	0.700	0.600	0.511	0.426	0.337	0.129
0.4	1152	2.077	1.522	1.247	1.058	0.909	0.781	0.667	0.557	0.440	0.165
0.5	1440	2.430	1.810	1.504	1.292	1.119	0.967	0.826	0.690	0.546	0.201
0.6	1728	2.761	2.090	1.752	1.520	1.331	1.157	0.992	0.828	0.653	0.238
0.7	2016	3.049	3.750	1.998	1.745	1.538	1.350	1.163	0.972	0.764	0.275
0.8	2304	3.707	2.649	2.249	1.973	1.748	1.544	1.341	1.124	0.880	0.313
0.9	2592	3.979	3.121	2.493	2.205	1.964	1.744	1.523	1.284	1.003	0.351
1.0	2880	4.349	3.420	2.736	2.426	2.181	1.951	1.715	1.453	1.136	0.390

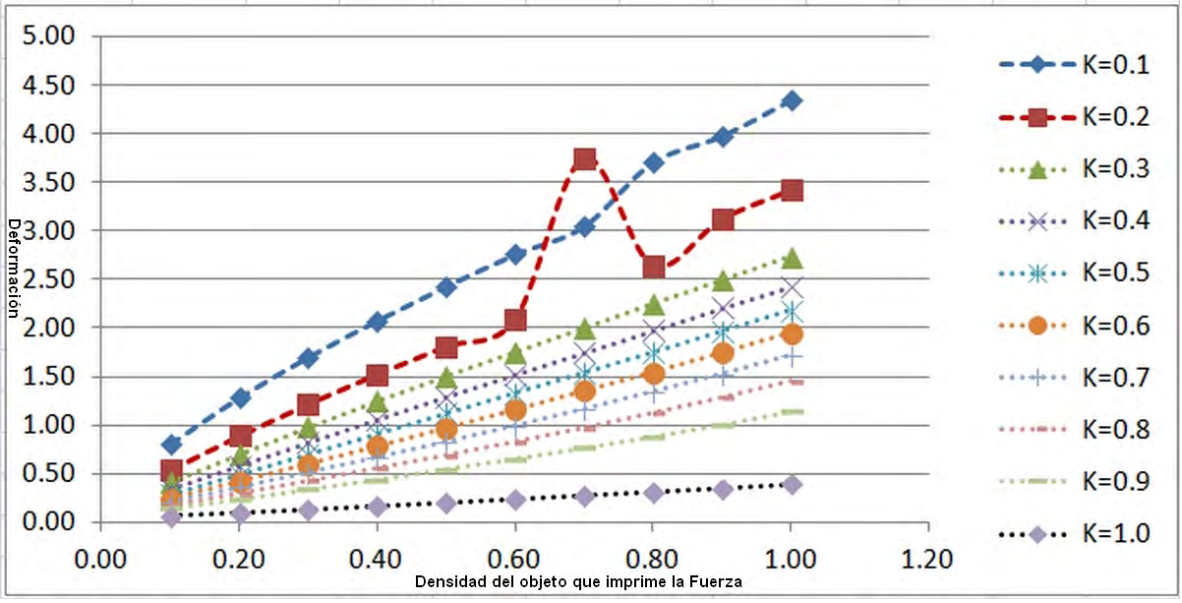


Figura 5.16. *Resultados de una prueba de compresión sobre una caja de* $20 \times 10 \times 20$ *[mm], con resolución* $R(6, 6, 6)$*, densidad: 1.0, radio de la partícula: 0.1, rigidez del volumen: 0.5, rigidez del resorte variable (K), fricción dinámica: 1.0, amortiguamiento: 0.5, respuesta a la colisión: 0.2, número de iteraciones: 5 y plano de mundo activado. Se observan dos curvas principales de comportamiento cuasi-no-lineal (*$K = 0.1, K = 0.2$*), con mayor resistencia ante cargas grandes; esto es, su comportamiento es más estable (visualmente hay menor penetración entre el objeto rígido y la malla). Hay una transición rápida a un comportamiento lineal a partir de* $K = 0.3$*, relativo al rango de seleccionado. El tiempo de estabilización para alcanzar el equilibrio entre los dos objetos osciló entre 30 segundos y 1.5 minutos, aproximadamente.*

Objeto Rígido		Deformación promedio (Objeto Deformable) [mm]									
Densidad	Masa [g]	K=0.1	K=0.2	K=0.3	K=0.4	K=0.5	K=0.6	K=0.7	K=0.8	K=0.9	K=1.0
0.1	288	2.350	1.655	1.054	0.787	0.613	0.940	0.396	0.319	0.251	0.133
0.2	576	5.018	2.838	2.342	2.047	1.201	0.952	0.759	0.600	0.455	0.196
0.3	864	8.594	6.315	4.566	2.243	1.879	1.987	1.460	0.886	0.664	0.262
0.4	1152	10.034	5.893	6.643	3.649	2.397	2.381	2.261	1.855	1.251	0.329
0.5	1440	11.169	10.726	7.295	6.647	3.798	2.762	2.559	2.067	1.683	0.397
0.6	1728	7.199	0.048	8.880	7.159	6.172	3.976	2.879	2.525	2.001	0.466
0.7	2016	8.742	0.048	11.279	10.473	6.356	6.000	4.020	2.787	2.381	0.536
0.8	2304	10.616	0.048	6.513	11.705	7.731	6.158	4.803	4.608	2.748	0.606
0.9	2592	10.658	0.048	0.046	0.046	11.165	10.566	5.241	6.163	3.385	1.596
1.0	2880	10.696	0.048	0.046	0.048	0.050	11.195	5.411	6.488	4.872	1.811

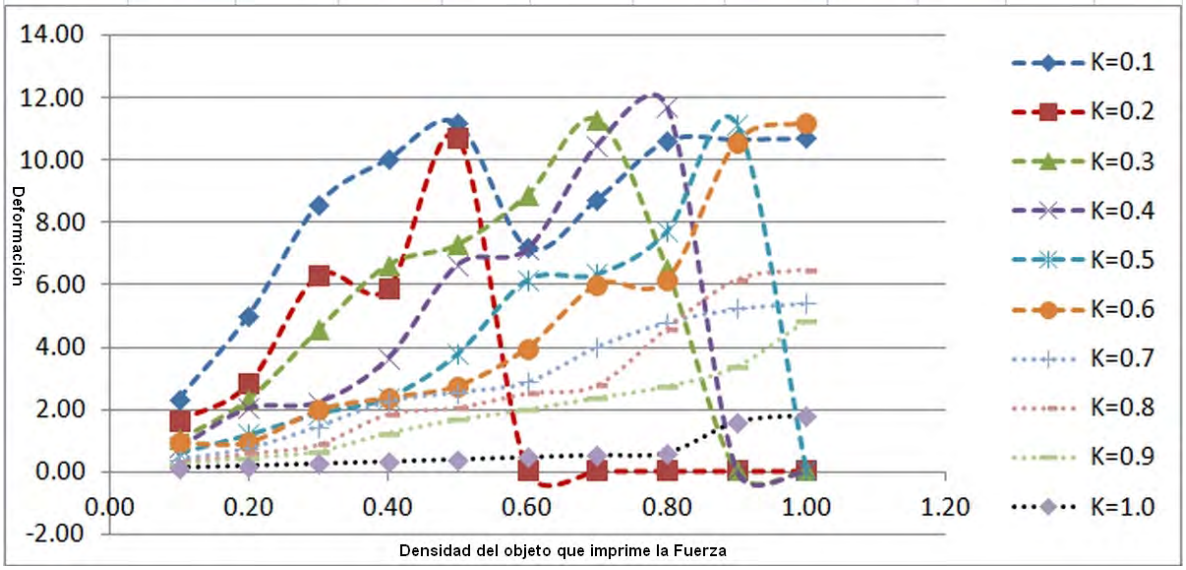


Figura 5.17. *Resultados de una prueba de compresión sobre un cubo de* $20 \times 20 \times 20$ *[mm], con resolución* $R(3, 3, 3)$*, densidad: 1.0, radio de la partícula: 0.1, rigidez del volumen: 0.5, rigidez del resorte variable (K), fricción dinámica: 1.0, amortiguamiento: 0.5, respuesta a la colisión: 0.2, número de iteraciones: 5 y plano de mundo activado. Se observan seis curvas de comportamiento no lineal (*$K = [0.1...0.6]$*), con mayor estabilidad relativa con* $K = 0.1$ *ante cargas grandes. Visualmente se observa una menor penetración del objeto rígido, en comparación con los experimentos con la caja, pero un incremento en el tiempo de estabilización, siendo éste de entre 1.5 a 2 minutos. La transición entre comportamientos no lineal a lineal es muy lenta.*

Objeto Rígido		Deformación promedio (Objeto Deformable) [mm]									
Densidad	Masa [g]	K=0.1	K=0.2	K=0.3	K=0.4	K=0.5	K=0.6	K=0.7	K=0.8	K=0.9	K=1.0
0.1	288	1.694	1.023	0.728	0.557	0.443	0.361	0.297	0.245	0.197	0.115
0.2	576	2.810	1.797	1.287	0.980	0.771	0.618	0.499	0.400	0.310	0.149
0.3	864	3.781	2.586	1.880	1.432	1.121	0.891	0.712	0.563	0.428	0.185
0.4	1152	5.589	3.356	2.507	1.920	1.498	1.183	0.937	0.734	0.551	0.223
0.5	1440	7.931	4.316	3.225	2.526	1.911	1.499	1.178	0.914	0.678	0.261
0.6	1728	8.395	7.856	3.971	3.198	2.463	1.981	1.547	1.105	0.811	0.300
0.7	2016	8.840	8.203	6.591	3.709	2.996	2.247	1.723	1.308	0.950	0.340
0.8	2304	9.090	8.241	7.152	5.879	3.533	2.824	2.050	1.530	1.095	0.381
0.9	2592	9.271	8.541	7.495	6.980	4.498	3.343	2.440	1.777	1.250	0.422
1.0	2880	9.500	8.606	7.710	7.202	6.257	4.778	2.986	2.063	1.416	0.464

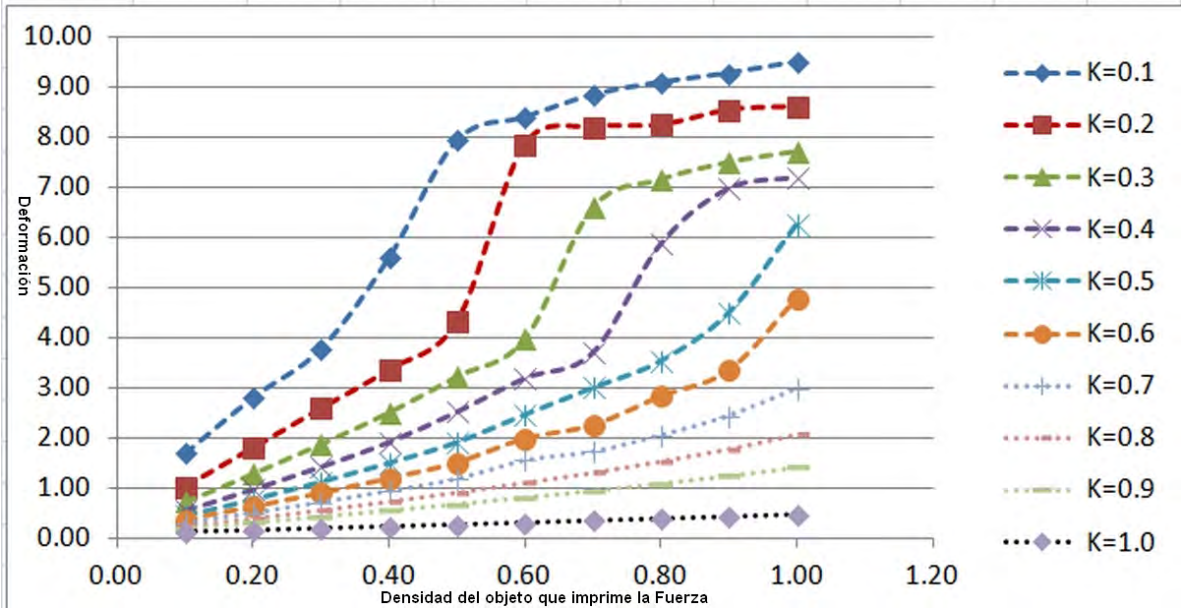


Figura 5.18. *Resultados de una prueba de compresión sobre un cubo de* $20 \times 20 \times 20$ *[mm], con resolución* $R(6, 6, 6)$*, densidad: 1.0, radio de la partícula: 0.1, rigidez del volumen: 0.5, rigidez del resorte variable (K), fricción dinámica: 1.0, amortiguamiento: 0.5, respuesta a la colisión: 0.2, número de iteraciones: 5 y plano de mundo activado. Se observan seis curvas con comportamiento no lineal, relativo al intervalo seleccionado. Visualmente se observa una mayor estabilidad de la deformación, en comparación con todos los experimentos anteriores, ya que hay muy poca o nula penetración del objeto rígido dentro del modelo deformable, aún con cargas grandes. El tiempo de estabilización se reduce, con respecto al experimento con el cubo de menor resolución, siendo éste de entre 30 segundos a un minuto. Finalmente, hay una transición suave y bien definida entre comportamientos no lineal a lineal.*

- En todos los experimentos, excepto en casos aislados, el objeto deformable regresó a su forma original tras eliminar la carga (deformación puramente elástica). Los casos aislados corresponden a inestabilidades en el cálculo de auto-colisión entre elementos de la propia malla de tetraedros.

- Aunque el factor de rigidez del resorte es uno de los principales causantes de la deformación de la malla, otro parámetro puede resultar igual de sensible, éste corresponde al radio de colisíón; que si no se configura adecuadamente puede provocar mayores inestabilidades. La experimentación con este parámetro mostró ser estable en el rango de 0.1 a 0.2 en geometrías con dimensiones menores a 10 cm. Para tamaños mayores, este valor debe ser caracterizado de acuerdo con la geometría.

- El comportamiento no lineal parece suceder en valores menores a 0.5, para el parámetro de rigidez del resorte. Por lo cual, cualquier valor por debajo de éste, visualmente puede ser usado para simular tejido de manera realista, en sus diferentes consistencias. Para consistencias muy suaves, un valor de 0.1 a 0.2 es suficiente. Valores menores a 0.1 provocan que la malla se vuelva demasiado viscosa y se deforme, aún sin aplicar algún tipo de fuerza externa.

5.2.2. Pruebas de simulación de tejido

Con el fin de probar una relativa generalidad de los valores obtenidos durante la caracterización para la simulación de tejido blando usando el SDK de *PhysX*, se usaron las mallas cilíndricas y esféricas construidas. En éstas se observó el realismo visual de la deformación y su estabilidad durante la interacción con un objeto rígido. En la experimentación con cilindros se usó una esfera de radio 1.5 veces mayor a su radio interno. En geometrías esféricas se usó una cápsula de radio 1.5 veces mayor al radio interno y altura de 10 [mm]. Se desactivó el piso del mundo para disponer del área positiva y negativa del espacio. Ejemplos de estas pruebas se muestran en las Figuras 5.19 y 5.20 .

En general, la experimentación con las geometrías mostró buen realismo visual en la simulación de tejido, obteniendo efecto interesantes. Por ejemplo, se observó que las geometrías de media y alta resolución, al contener mayor cantidad de elementos interaccionando con el objeto rígido a la vez, ejercen una fuerza sobre éste último que lo empujan a lo largo de la cavidad, provocando un desplazamiento involuntario. Este efecto puede ser muy útil en la simulación de esfínteres de manera muy realista.

Las geometrías que fallaron para la prueba de simulación de tejido fueron las de alta resolución, debido a que se presentaron inestabilidades en la deformación, aún sin aplicar fuerza externa alguna. Esto puede deberse al exceso de resolución radial, que en cierto modo debilita las fuerzas internas por el exceso de resortes de poca longitud.

Finalmente, los experimentos mostraron que los valores más estables para simular

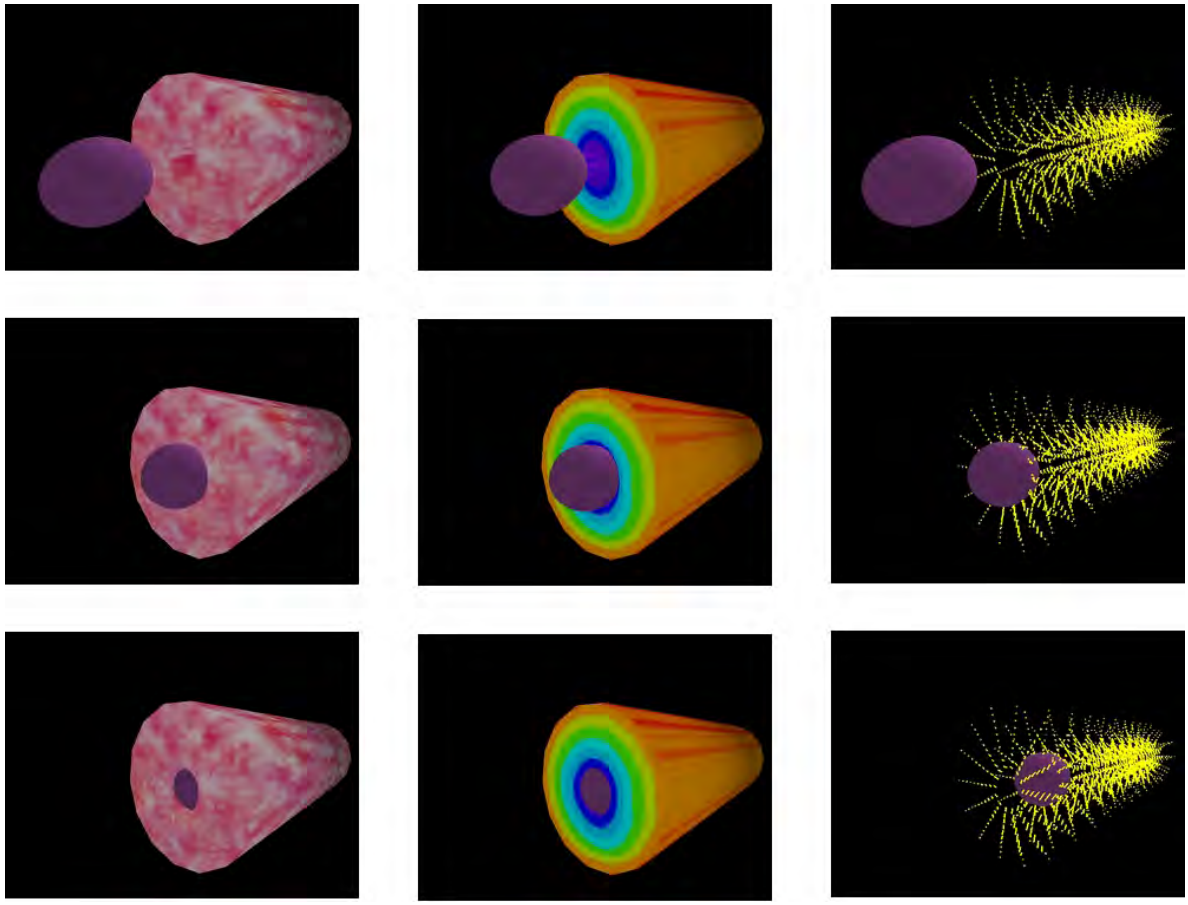


Figura 5.19. *Prueba de simulación de tejido en una geometría cilíndrica de resolución media. De izquierda a derecha, diferentes modos de visualización. De arriba a abajo, simulación de la entrada de una esfera 1.5 veces mayor al radio interno.*

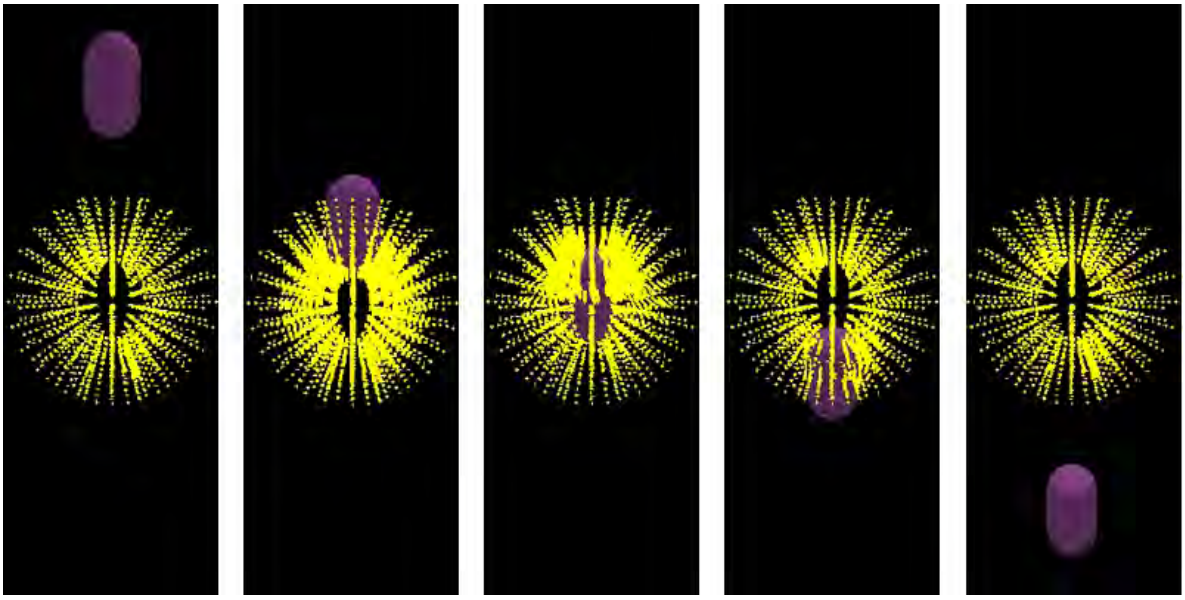


Figura 5.20. *Prueba de simulación de tejido en una geometría esférica de resolución media alta. De izquierda a derecha, se observa el campo de vectores de deformación, el cual se va volviendo más denso conforme el objeto rígido interacciona con el tejido.*

tejido, en sus diferentes geometrías y resoluciones (sin tomar en cuenta a las de alta resolución), son:

- Rigidez del volumen: $<$ 0.5 para propagar mucho la deformación, 0.5 para una propagación media, 1.0 para no propagar la deformación. Un valor pequeño ayuda a simular movimientos del tejido mientras un objeto rígido se abre paso sobre éste.
- Rigidez del resorte: 0.1 a 0.5 (con las observaciones descritas en la subsección anterior).
- Fricción dinámica: 1.0 para incluir fuerzas de rozamiento en paredes y 0.0 para un desplazamiento fluido. En algunas ocasiones se prefiere eliminar el rozamiento ya que puede inestabilizar la deformación al estirar demasiado los vértices de la malla.
- Factor de amortiguamiento: 0.5. No hay una razón aparente para no usar otros valores, ya que no se observaron cambios significativos en la deformación al variar este parámetro.
- Respuesta a la colisión: 0.2.
- Radio de la partícula: 0.1 a 0.2, dependiendo del tamaño de la geometría. Recordemos que es usado para cálculos de colisiones.
- Iteraciones: 5.

Sección 5.3

Simulación de cortes de tejido blando

Una vez teniendo los dos elementos principales, una geometría y un método de deformación para simular tejido blando, el siguiente paso fue la prueba con los diferentes métodos computacionales para realizar una acción de corte. Los métodos que se probaron fueron: separación de nodos, retracción, deformación plástica y eliminación de elementos. Para los primeros tres se programó una aplicación muy básica y menos interactiva que las aplicaciones anteriores, ya que el objetivo fue explorar las diferentes opciones para finalmente concentrarnos en una aplicación para simular resección, en la que el último método parece ser más efectivo. Por otra parte, la mayoría de los experimentos realizados extienden la idea de los ejemplos mostrados en el Capítulo 4, con un poco más de detalle y con la inclusión de modelos de superficie de las herramientas de corte. Las observaciones experimentales sirvieron para plantear la Tabla 4.3, del mismo capítulo sobre cortes.

Como se planteó en su momento, el uso de herramientas de corte es específico para cada técnica quirúrgica y el mundo de posibilidades es grande. Por ello, para no dejar a un lado la importancia que tiene la herramienta de corte durante una simulación, se modelaron un conjunto de mallas triangulares para representar objetos que visualmente dan la impresión de ser una herramienta de corte. Este trabajo de modelado 3D se llevó a cabo a partir de imágenes de herramientas reales, usando el software de modelado *Blender*, y almacenando la información en formato *Wavefront OBJ*. La Figura 5.21 muestra algunos ejemplos de estos modelos.

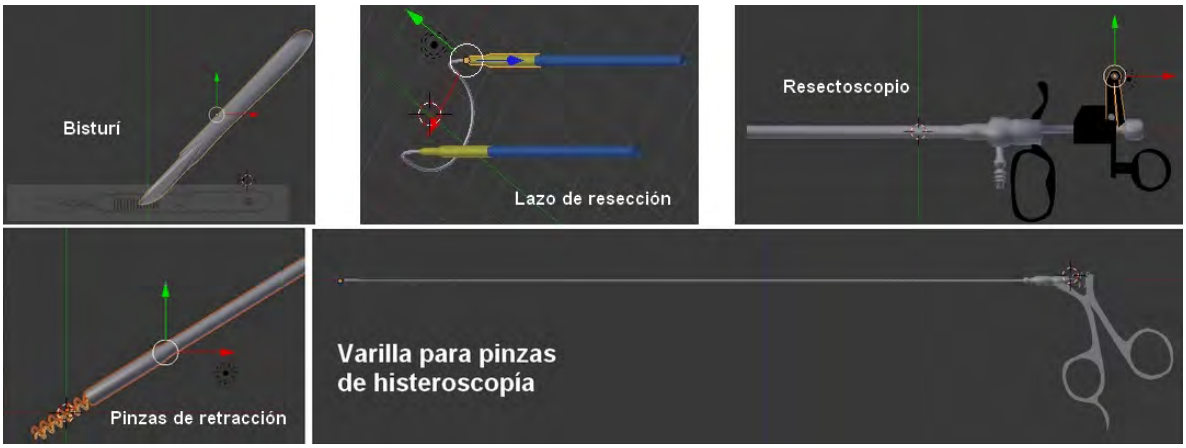


Figura 5.21. *Conjunto de modelos virtuales usados en la simulación de cortes de tejido.*

5.3.1. Prueba de separación de nodos

En esta prueba se usó una malla de tetraedros rectangular, en configuración alternada, diferente a las mallas creadas en la primera sección de este capítulo, debido a que sólo necesitamos de probar la separación de un conjunto de nodos para simular cortes con bisturí sobre tejido. La resolución de la malla fue de $11\times2\times11$, y se usaron como herramientas virtuales al bisturí y las pinzas de retracción. En cuanto a parámetros de deformación se usaron los siguientes valores:

- Rigidez del volumen: 0.5
- Rigidez del resorte: 0.5
- Fricción dinámica: 1.0
- Factor de amortiguamiento: 0.5
- Densidad: 1.0.

La metodología consistió en tocar varios puntos en la superficie de la malla con un bisturí virtual, siguiendo una trayectoria en línea recta sobre el eje X, activando un modo de corte. Una vez finalizado, pasando a un modo de interacción con pinzas, se efectúa una deformación del tejido para verificar que el corte se haya efectuado. La Figura 5.22 muestra este procedimiento.

Respecto a varias pruebas realizadas, se hacen las siguientes observaciones:

- Visualmente es claro que es útil para simular incisiones o disección de tejidos. En limitados casos es posible usarse para extirpación o resección; sin embargo, en tales casos debe agregarse un control secuencial sobre la separación, ya que generalmente en este tipo de procedimientos la herramienta interactúa con varios puntos a la vez.
- Se trabaja sobre nodos de la malla (posiciones), por lo que computacionalmente es simple de manejar en cuanto a procesamiento y memoria.
- Aunque el efecto con mallas de tetraedros es muy llamativo, generalmente siempre se trabaja con la superficie del objeto, por tanto es recomendable usar mallas triangulares para optimizar cálculos. Todo depende de la aplicación.
- No se requieren altas resoluciones, dado que los cortes definen trayectorias rectas. En casos donde no aplica, es en cortes diagonales a la disposición de la malla. En tales casos, se debe tomar en cuenta durante la construcción de la malla, la orientación geométrica de los tetraedros.

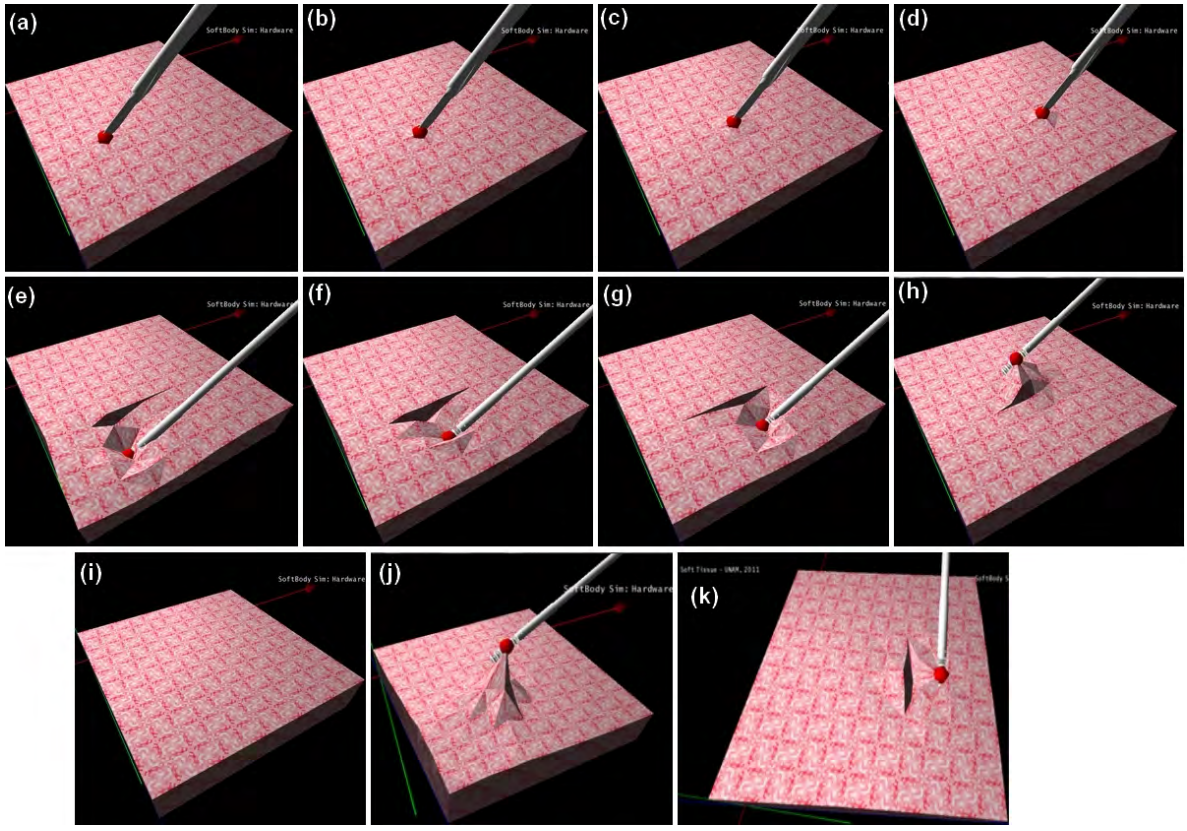


Figura 5.22. *Procedimiento de corte sobre una malla de tetraedros por el método de separación de nodos. (a-d) Trayectoria del corte con la herramienta virtual. El punto rojo indica el vértice de contacto. (e-h) Prueba de separación del vértice de acuerdo con la trayectoria. Se observa el cambio de herramienta por unas pinzas para separar tejido. (i) Estado inicial de la malla. (j) Prueba de deformación usando pinzas de retracción. (k) Prueba de corte una vez finalizado el procedimiento.*

5.3.2. Prueba de retracción

Se usó una malla de tetraedros rectangular, en configuración alternada, con resolución $11 \times 2 \times 11$. La herramienta usada fue la pinza de retracción, como se muestra en la Figura 5.23.

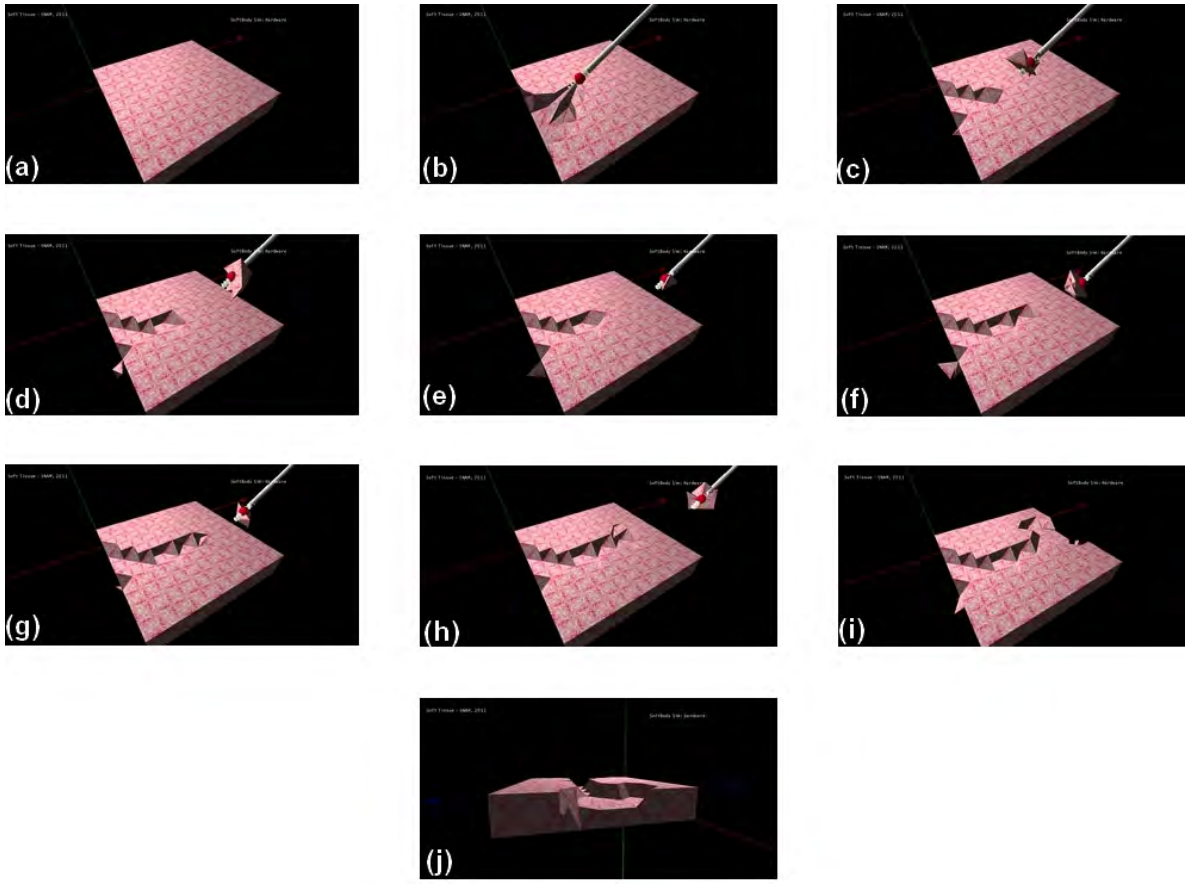


Figura 5.23. *Procedimiento de corte sobre una malla de tetraedros por el método de retracción. (a) Estado inicial de la malla. (b) Primer contacto de la herramienta con uno de los puntos. Se fija el punto de contacto con la punta. El punto rojo indica la posición de agarre. (c) Las aristas llegan a su elongación máxima y se desprende un pedazo de tejido. (d-i) Se repite el mismo procedimiento para eliminar pedazos de otras regiones de la malla. (j) Vista de perfil del hueco formado tras eliminar parte del tejido.*

Los parámetros de deformación empleados fueron:

- Rigidez del volumen: 0.5
- Rigidez del resorte: 0.5

- Fricción dinámica: 1.0
- Factor de amortiguamiento: 0.5
- Densidad: 1.0.

Respecto a varias pruebas realizadas, se hacen las siguientes observaciones:

- Es ideal para simular extirpación de tejido. No así para incisión o resección, ya que se necesita necesariamente de un punto que provoca la elongación de los resortes asociados.
- Por lo anterior, el elemento de trabajo es el nodo o vértice.
- Aunque es aplicable a mallas triangulares, el efecto es más interesante con mallas de tetraedros, dado que se eliminan porciones que pueden incluir el interior del objeto y no sólo su superficie.
- Una resolución media parece tener buen resultado. Mallas de baja resolución pueden eliminar porciones muy grandes y producir mallas visualmente mordidas. Mallas de muy alta resolución requieren muchas interacciones para eliminar porciones grandes de tejido.

5.3.3. Prueba de deformación plástica

Se usó una malla de tetraedros rectangular, en configuración alternada, con resolución 11×2×11. Dado que en esta versión de prueba del software se trabaja con nodos para provocar una deformación plástica, el experimento consistió en tocar varios puntos de la malla, deformándolos plásticamente usando un enfoque basado en la dinámica de las partículas. El resultado se muestra en la Figura 5.24.

Los parámetros de deformación empleados fueron:

- Rigidez del volumen: 0.5
- Rigidez del resorte: 0.5
- Fricción dinámica: 1.0
- Factor de amortiguamiento: 0.5
- Densidad: 1.0.

Respecto a varias pruebas realizadas, se hacen las siguientes observaciones:

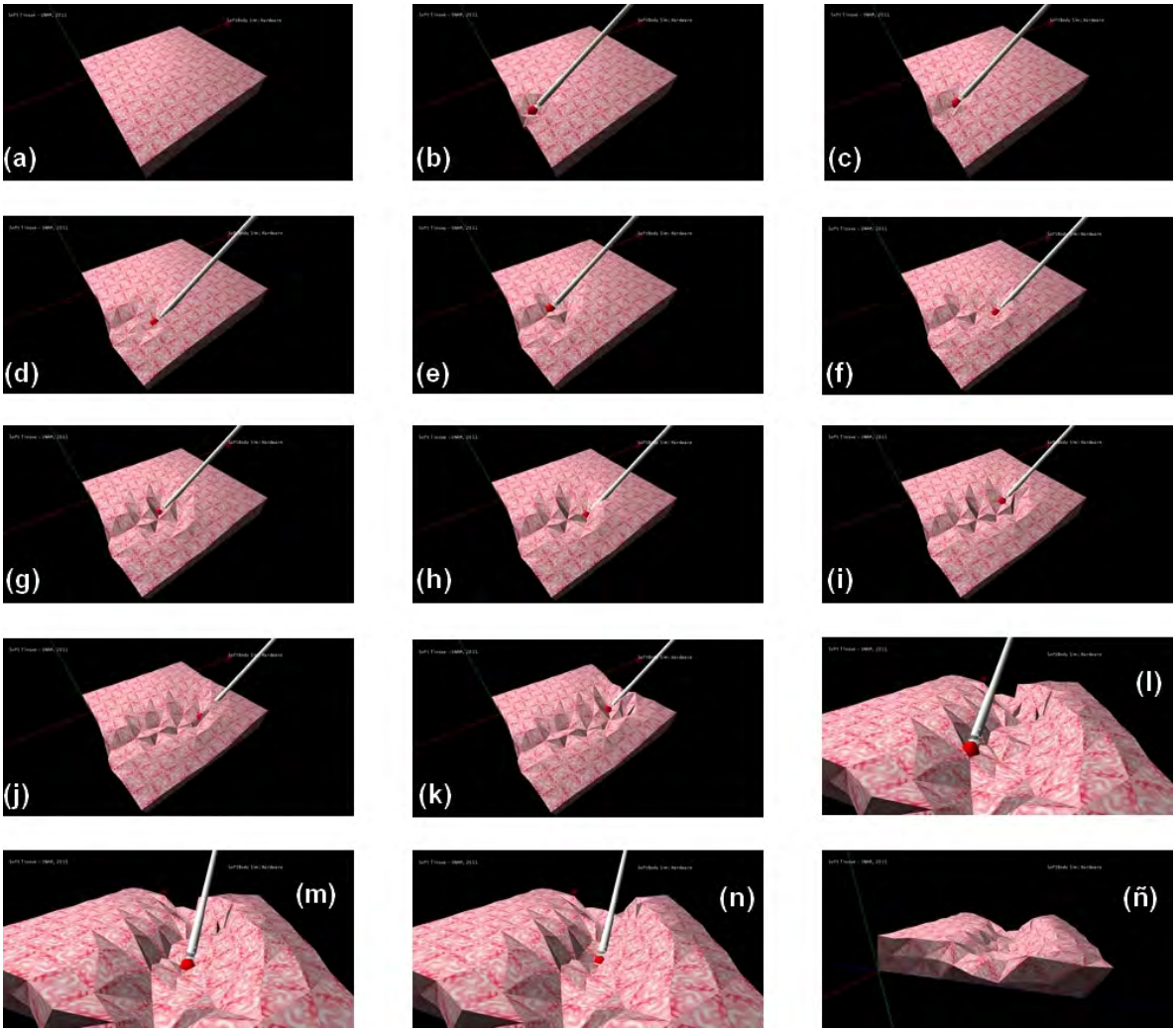


Figura 5.24. *Procedimiento de corte sobre una malla de tetraedros por el método de deformación plástica. (a) Estado inicial de la malla. (b) Primer contacto con una varilla. El punto rojo indica el vértice de contacto. Se efectúa una primera deformación plástica. (c) La región deformada queda con la forma anterior y se continúa con la interacción con un nuevo vértice. (d-k) Se repite el procedimiento, tocando varios puntos a lo largo de la superficie. (l-n) Se deforman puntos del interior del hueco para probar la continuidad de la deformación. (ñ) Malla final.*

- Visualmente se forman huecos, por lo que su aplicación principal es la simulación de cortes por resección, donde una herramienta deforma plásticamente varios puntos a la vez, y deja a su paso un espacio. No es aplicable para incisiones o extirpación de tejido.
- El elemento de interacción es el nodo o vértice, lo que facilita su procesamiento y manejo en memoria.
- Dado que trabaja en conjunto con un modelo deformable elástico para conservar la forma de la malla, se prefiere el uso de mallas de tetraedros. Sin embargo, la experimentación con superficies también tiene un resultado visual interesante. Un ejemplo de este experimento puede verse en la Figura 4.15 en el Capítulo 4.
- Una resolución media parece ser suficiente para lograr deformaciones suaves. Mallas de baja resolución requieren de un buen método de cálculo de normales para suavizar la transición entre vértices deformados. Mallas de alta resolución tienen un mejor resultado visual pero demandan más recursos de cómputo.

5.3.4. Prueba de drenado

Se realizaron pruebas de drenado sobre geometrías planas y tubulares. El experimento consistió en hacer pasar una geometría conocida entre el tejido simulado y se observó el proceso de drenado. Ejemplos sobre varias resoluciones se muestran en las Figuras 5.25 y 5.26.

Los parámetros de deformación empleados fueron:

- Rigidez del volumen: 0.5 (en resoluciones de $10 \times 11 \times 9$ se usó un valor de 1.0 para lograr la estabilidad).
- Rigidez del resorte: 0.5 (en resoluciones de $10 \times 11 \times 9$ se usó un valor de 1.0 para lograr la estabilidad).
- Fricción dinámica: 1.0
- Factor de amortiguamiento: 0.5
- Densidad: 1.0.

Respecto a varias pruebas realizadas, se hacen las siguientes observaciones:

- Es ideal para simular procesos de resección, ya que el hueco simulado, con una adecuada resolución puede agregar buen realismo. No es útil para simular incisiones ya que se generarían surcos muy grandes. Puede ayudar a la simulación de extirpación,

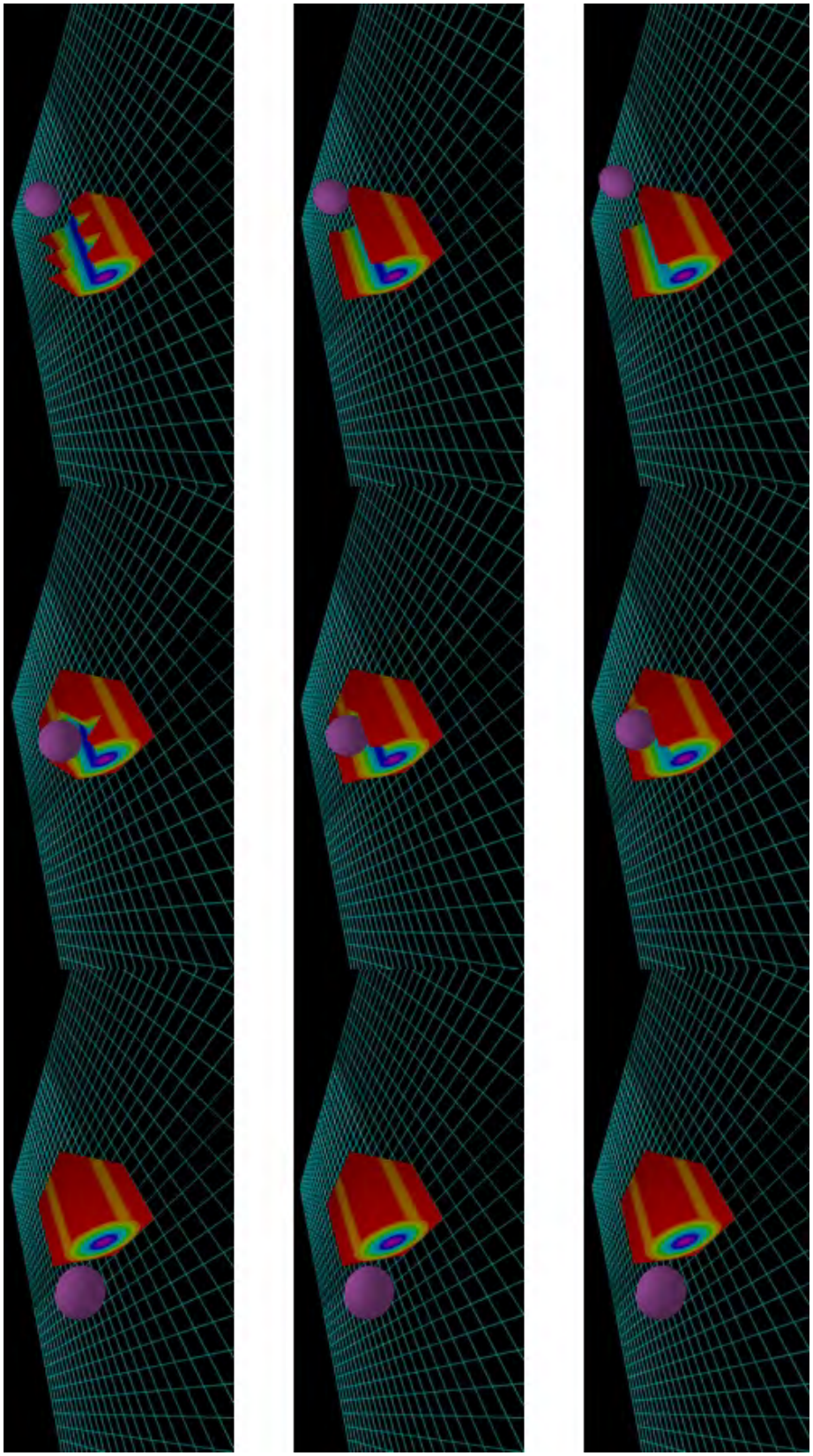


Figura 5.25. *Prueba de drenado sobre geometrías en forma de caja con resoluciones:* $3 \times 3 \times 3$, $6 \times 6 \times 6$ y $10 \times 11 \times 9$ *(De arriba hacia abajo). De izquierda a derecha: estado inicial, drenado y forma final. Visualización en modo textura estática tubular.*

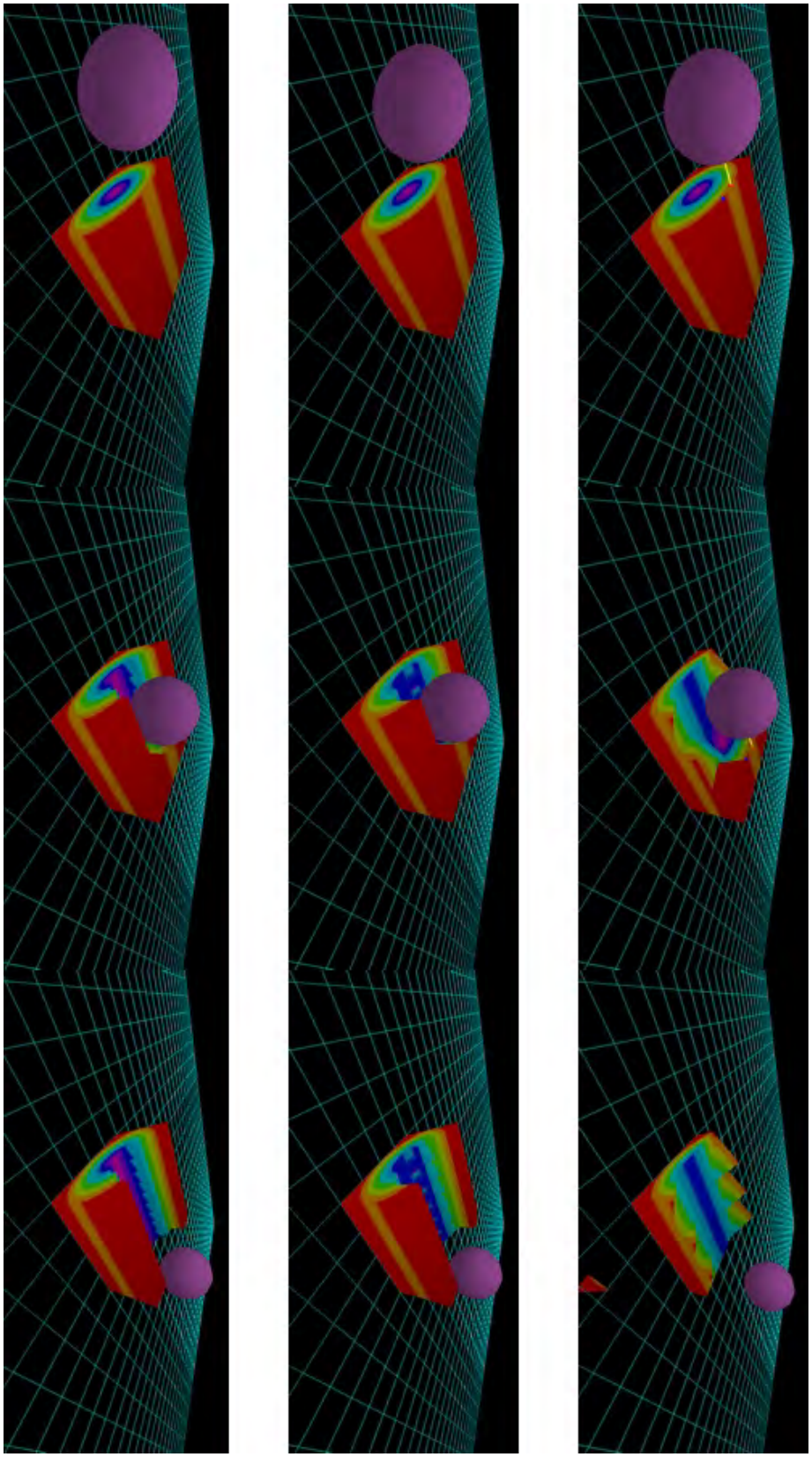


Figura 5.26. *Prueba de drenado sobre geometrías en forma de cubo con resoluciones:* $3 \times 3 \times 3$, $6 \times 6 \times 6$ *y* $10 \times 11 \times 9$ *(De arriba hacia abajo). De izquierda a derecha: estado inicial, drenado y forma final. Visualización en modo textura estática tubular.*

agregando el pedazo removido como una animación, aunque se prefiere el método por retracción.

- Dado que se eliminan tetraedros completos, el método trabaja sobre estos elementos y puede presentarse un efecto de mordido, como se aprecia en las pruebas de baja resolución.
- Requiere de mallas de alta resolución para evitar el efecto anterior, lo cual significa que los parámetros de deformación deben calibrarse para lograr la estabilidad.

Sección 5.4

Aplicación: Resección en un simulador de cirugía de la próstata

Finalmente, el software y todos los experimentos realizados en las secciones anteriores sirvieron para el desarrollo de un simulador de cirugía de la próstata, usando como plataforma de desarrollo el SDK de *nVidia PhysX*. Este desarrollo se ha venido tratando a lo largo de todo el trabajo, presentándose como caso de estudio, sin embargo, no se han tocado varios detalles que se presentan a continuación.

En general, los principales aportes al simulador son:

- Incorporación de un motor de física para cálculos de deformaciones en mallas de tetraedros.
- Incorporación de bibliotecas para la carga de mallas de tetraedros usando el formato *VOG*.
- Calibración del modelo virtual de la próstata para comportarse como tejido blando, de acuerdo con los resultados de la sección 5.2.
- Generación de diferentes modelos de la próstata para fines de prueba con médicos y residentes, a quienes va dirigido el uso del simulador.
- Incorporación de la técnica de eliminación de elementos para simular resección de tejido blando.
- Modelado e incorporación de una nueva herramienta de corte, más realista visualmente.

Los modelos de la próstata que se generaron usando la metodología descrita en el Capítulo 3, son los mostrados en la Figura 5.27. Las imágenes fuente provinieron del

trabajo de *Soriano[72]* (2008), que corresponden a imágenes ya procesadas de la forma de la próstata. A éstas se incorporaron dos secciones, una para definir la entrada tubular de la uretra peneana y otra para definir la transición con el cuello vesical. Así, los modelos finales muestran circunferencias bien definidas en sus extremos. Los tamaños, en dimensiones de la simulación, corresponden a modelos de aproximadamente 20 [mm] de diámetro externo, 4 [mm] de radio interno y 40 [mm] de longitud (incluidas las zonas de transición).

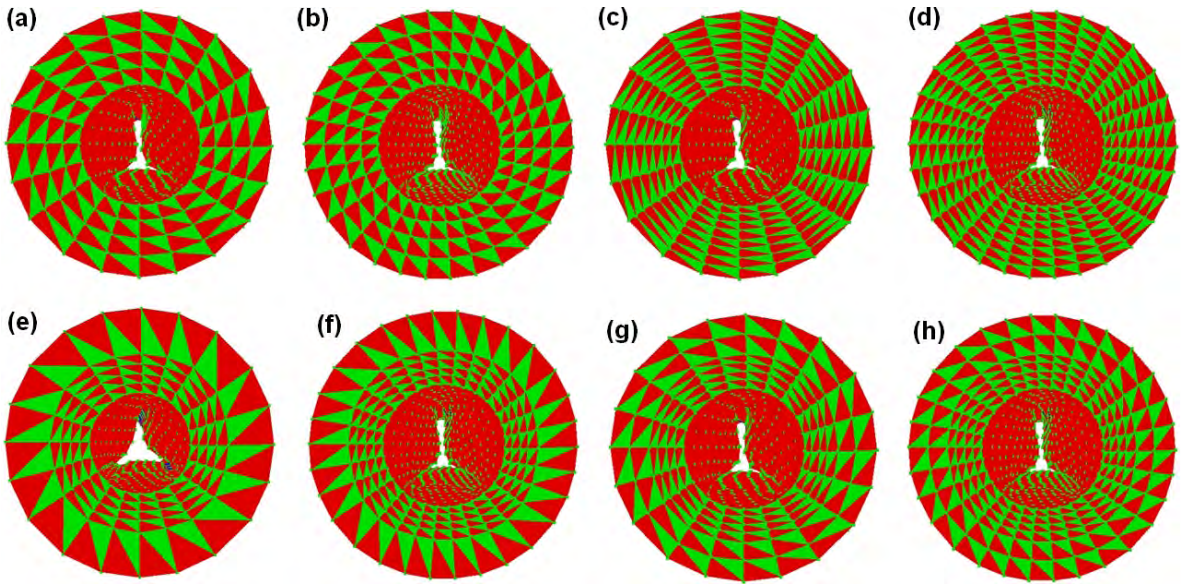


Figura 5.27. *Mallas de la próstata a diferentes resoluciones. (a) 4 capas uniformes, 20 muestras por contorno. (b) 4 capas uniformes, 30 muestras por contorno. (c) 8 capas uniformes, 20 muestras por contorno. (d) 8 capas uniformes, 30 muestras por contorno. (e) 5 capas no uniformes, 20 muestras por contorno. (f) 5 capas no uniformes, 30 muestras por contorno. (g) 6 capas no uniformes, 20 muestras por contorno. (h) 6 capas no uniformes, 30 muestras por contorno.*

La uretra peneana, como se mencionó anteriormente, se creó como forma tubular a partir de un cilindro de 4 [mm] de radio interno, 10 [mm] de radio externo (diferente a la próstata, ya que en este caso no nos interesa un espesor) y 10 [cm] de longitud. Para la uretra no fue necesario la construcción de varios modelos, uno fue suficiente para fines de visualización.

Tanto el modelo de la próstata, como el de la uretra, se texturizaron usando una pila de imágenes que definen una textura 3D, generadas por *García[24]* (2009), mediante una técnica procedural. Estas texturas también se usaron en el software de simulación de tejido blando, antes descrito. El mapeo se manejó en las dos modalidades: dinámico y estático, cada uno aportando efectos interesantes.

Para simular la deformación de tejido, se realizó una prueba de realismo visual sobre

una malla anatómica de la próstata, usando los rangos de valores obtenidos en la sección 5.2. Los valores con mejor realismo visual fueron: rigidez del volumen: 0.1, rigidez del resorte: 0.2, fricción dinámica: 0.0, factor de amortiguamiento: 0.5 y densidad: 1.0. El resultado se muestra en la Figura 5.28.

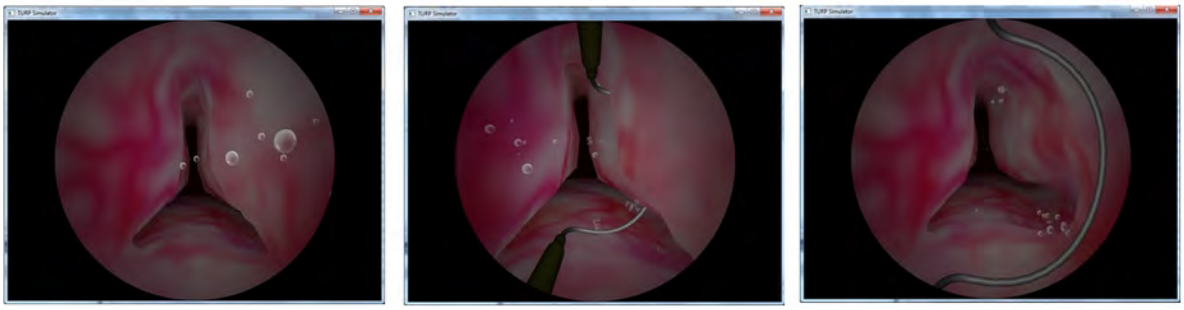


Figura 5.28. *Simulación de deformaciones sobre una malla de tetraedros que representa la anatomía de la próstata.*

Dado que la interacción entre el tejido simulado y la herramienta de resección no mostraron una deformación fluida, en parte por la alta resolución en triángulos del asa, se recurrió a una geometría más básica, formada por cubos. Esta técnica se denomina *composición*, y es muy empleada en el SDK de *PhysX* para aumentar el rendimiento en aplicaciones que hacen uso intensivo de las colisiones. La composición se efectúa definiendo un conjunto de primitivas, como cubos o esferas, que dispuestas en cierta configuración forman una nueva figura geométrica, más compleja. Cada elemento del nuevo objeto (rígido) procesa las colisiones de manera individualizada, del tal forma que al sumar los efectos de cada uno, se obtiene una respuesta a la colisión más fluida sobre un objeto deformable. Además, dado que todos los elementos que componen la nueva forma pertenecen a ella misma, las colisiones entre ellos están desactivadas. En la Figura 5.29 se muestra la incorporación de esta idea en el simulador de cirugía. En ella se observan dos características, la primera, el asa de resección se compone de 20 cubos de 0.5 [mm] de espesor, dispuestos en forma de U; la segunda, cada mango se forma de 10 cubos de 1 [mm] de espesor, sin espaciamientos entre ellos. Esto ayudó a que se mejore la fluidez de la deformación y se evita en lo posible el efecto de penetración en el tejido cuando no se trata de un corte.

Seguramente se preguntará, ¿y el bloque situado a la mitad del lazo, qué función tiene?. Dado que el método que se implementa sobre el simulador para realizar cortes en el tejido, es el de eliminación de elementos (que como recordará, se trata de eliminar tetraedros completos de la malla), un efecto no deseado que se presenta cuando se usan lazos de corte, es que no todos los tetraedros situados por encima del lazo se detectan en la colisión, produciendo eliminaciones incompletas y la aparición de residuos dentados, algo muy similar a lo que ocurre con el efecto de *aliasing* en el despliegue 2D. Para ello, el bloque horizontal sobre el sistema de coordenadas de la herramienta, tiene la función de minimizar el efecto, detectando la mayor cantidad de elementos que lo provocan. Una

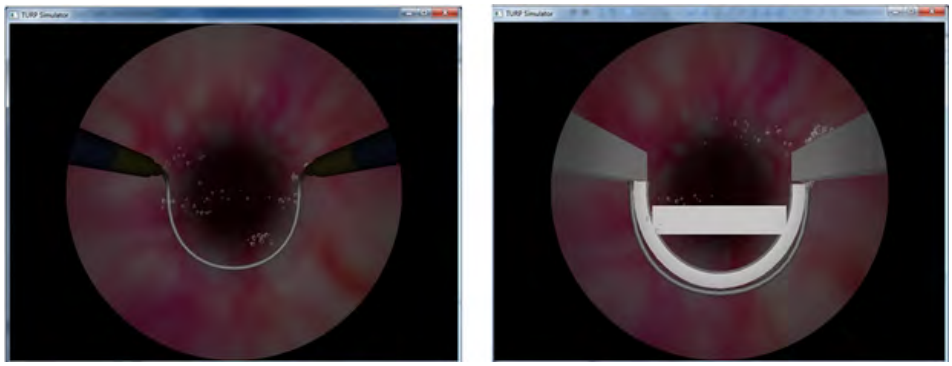


Figura 5.29. *Técnica de composición sobre el asa de resección para aumentar el rendimiento del simulador de cirugía de la próstata.*

solución alternativa consiste en suavizar el modelo cada vez que se efectúa una acción de corte, para eliminar protuberancias; sin embargo, se ha que tener en cuenta que esto implica agregar un mayor costo computacional. La experimentación con esta última alternativa no se aborda en este trabajo. Como conclusión a este capítulo, la Figura 5.30 muestra el efecto de corte aplicado sobre una malla de la próstata. En la última revisión por parte del Dr. Sergio Durán, médico Urólogo del Instituto Nacional de Rehabilitación, se observó un realismo visual de la deformación de tejido y de corte adecuados.

Figura 5.30. *Simulación de resección usando el método de eliminación de elementos. La banda rectangular del asa se emplea para eliminar tetraedros no detectados por los cubos en el lazo de resección.*

CAPÍTULO 6

Conclusiones y trabajo a futuro

Tras el estudio, experimentación y análisis sobre el tema de este trabajo, apegado a los objetivos y perspectivas en el planteamiento, se presentan las siguientes conclusiones:

1. El tejido blando, presente en una gran cantidad de procedimientos quirúrgicos, puede ser modelado física, matemática y computacionalmente. Físicamente se interpreta de acuerdo con sus propiedades estáticas y dinámicas, formulando ecuaciones que desde el punto de vista matemático pueden ser escritas en un modelo. Computacionalmente, el tejido blando puede ser modelado como un objeto deformable; y resuelto como tal, usando un método heurístico, un método basado en la mecánica de los medios continuos o un método híbrido. Actualmente ya existen paquetes que tienen implementados una gran cantidad de métodos de deformación, algunos de código abierto y otros comerciales. El SDK de *nVidia PhysX* posee una muy buena implementación de un método híbrido para la simulación realista de objetos blandos.

2. La importancia de simular tejido blando en una computadora, radica en el potencial que ofrece para el desarrollo de sistemas para entrenamiento médico, donde un aprendiz puede sumergirse en un entorno gráfico e interactuar con los elementos geométricos, que representan de manera realista el comportamiento de tejidos u órganos en el cuerpo humano. El reto para el desarrollador, desde este punto de vista, implica el pleno conocimiento de lo que se tiene que simular, asesorándose con ayuda de un médico y de la literatura. En general, se concluye que cualquier sistema que pretenda simular el comportamiento de una estructura, debe tener una base teórica que lo sustente.

3. La base para la simulación de objetos deformables es su geometría, ya que a partir de ella dependerá el funcionamiento del modelo físico, matemático y computacional. La creación y aplicación de metodologías para la construcción de geometrías es un paso

importante en la simulación computacional, ya que enmarcan el tipo de espacio en el cual se desenvolverá un modelo. En este trabajo se trataron las geometrías formadas por tetraedros, que necesariamente nos ubican en un espacio tridimensional. Los métodos de generación de mallas tratados van desde objetos muy básicos como: un cubo, una caja, un cilindro y una esfera, hasta modelos anatómicos complejos, cuya información proviene de imágenes o inclusive de superficies. Con respecto a estos métodos, el producto principal fueron las mallas de prueba de tejido y mallas que representan una próstata virtual, útiles para un simulador de cirugía de próstata.

4. Entendiendo la estructura espacial y el comportamiento de un objeto deformable, como el tejido, es posible definir un conjunto de acciones que operan sobre éste, como cortar, suturar, extirpar, etc. Este trabajo se centró en la acción de corte. Además, limitado a tres tipos: incisión, retracción y resección. Para su simulación computacional, se presentaron cinco metodologías: separación de nodos, retracción, eliminación de elementos, deformación plástica y cambio topológico; cada una aplicable a una u otra acción quirúrgica. Se puede concluir que el mejor método para simular incisiones sobre tejido es el de cambio topológico, para simular extirpación el de retracción y para simular resección el de eliminación de elementos.

5. La experimentación con el SDK de *NVIDIA PhysX* mostró que, con la configuración adecuada, puede lograrse simular tejido de manera realista. El software de generación de mallas fue lo suficiente robusto como para generar una gran variedad de geometrías aleatorias para fines de prueba. Y la aplicación culminante, la incorporación de las metodologías en un simulador de cirugía de próstata, donde la acción de corte se vió beneficiada por el método de eliminación de elementos, representan una contribución significativa para el desarrollo de simuladores computarizados que involucran tejido blando.

6. Finalmente, desde el punto de vista de aplicación en las ciencias médicas, este trabajo brinda un marco teórico y experimental suficiente para la creación de simuladores computarizados capaces de aportar beneficios en el modelo de enseñanza de la cirugía. En particular, en el campo de la Urología Mexicana, el poder ofrecer un desarrollo tecnológico en el marco de la cirugía virtual, permite la modernización y la competencia internacional en términos de investigación de nuestro país, aportando un beneficio a la sociedad en un sector cada vez más demandante, como es el sector salud.

Trabajo a futuro

En un proyecto como éste, donde intervienen una gran cantidad de conceptos y tareas de programación, es muy difícil plantear una solución única a los problemas presentados; por ello, aquí algunos puntos que son susceptibles a mejoras, cambios u optimizaciones:

- Aplicando algunas adaptaciones en los algoritmos de separación de nodos y de cambio topológico, es posible la implementación de efectos de resección sobre mallas de volumen. O bien, complementando el método de drenado, se pueden adaptar métodos de refinamiento local sobre el área de corte.

- Los algoritmos de generación de mallas de tetraedros, propuestos en el Capítulo 3, son totalmente optimizables. Además, un análisis formal de la regularidad de los tetraedros en las mallas generadas, ayudaría a adicionar parámetros que permitan una mejor estabilidad al momento de simular deformaciones.

- Sobre el software desarrollado, es posible fusionar todos ellos para formar uno solo, lo cual tendría la ventaja de poder generar, deformar y cortar una malla de forma serializada.

- La caracterización de objetos deformables para simular tejido, usando el método por compresión, podría completarse además con el diseño e implementación de un método de caracterización por elongación. Ambos métodos eventualmente podrían ajustarse a curvas reales de comportamiento de tejido, obteniendo valores más exactos sobre la deformación.

- Aunque se trató de tomar en cuenta la mayor cantidad de propiedades físicas de los objetos durante las simulaciones, no se incorporó el cálculo de volumen en estructuras complejas. De incorporarse, ayudaría a conocer qué tanto volumen pierde un objeto en los métodos de retracción, eliminación de elementos o inclusive deformación plástica.

- Incorporar las metodologías en el desarrollo de otros simuladores, ayudaría a validar aún más todo el trabajo.

APÉNDICE A

Programación de *NVIDIA PhysX* y *Qt*

En el Capítulo 2 se abordaron los tres modelos fundamentales para el modelado biomecánico y computacional de cuerpos físicos: el modelo de partículas, el modelo de objetos rígidos y el modelo de objetos deformables. Además se mencionaron los principales paquetes de desarrollo usados para la simulación de éstos, siendo *NVIDIA PhysX*[64] el paquete que por sus características de *middleware*, ofrece los mejores resultados para aplicaciones en tiempo real. En este apartado abordaremos la programación de software de simulación usando la especificación 2.8.1 del API de *PhysX* [1], *OpenGL*[60] como plataforma de visualización y *Qt*[65] 4.7.3 para la incorporación de GUI's de control[2]. Todos ellos desarrollados bajo lenguaje C/C++.

Comenzaremos analizando el ciclo de simulación del SDK de *PhysX* para poder entender la forma en que se lleva a cabo la tarea de cálculos físicos, continuaremos con el análisis de una aplicación gráfica de usuario completa para manejar este tipo de simulaciones, y finalmente abordaremos los elementos necesarios para configurar cada objeto que forma una escena de *PhysX*. Mucho del contenido obvia que se tiene un conocimiento básico sobre la programación de *OpenGL*, por lo que se omite la explicación de esta biblioteca y de las funciones usadas. Así también, se omiten las funciones para dibujar objetos geométricos ya que el código sobrepasa el objetivo de este espacio; sin embargo, la API de *PhysX* proporciona información muy completa para esta acción.

[1] Parte del contenido de este apartado es un extracto del programa de entrenamiento para el uso de la API de *PhysX*, disponible en el sitio Web de *NVIDIA Developers*[58], también incluido como parte del software de distribución. El curso completo del SDK 2.8.1 consta de 12 capítulos y 92 tutoriales.

[2] Es importante mencionar que este material no pretende ser un manual o una guía para la programación de aplicaciones de simulación, sino únicamente un elemento de referencia para entender un programa que usa estas bibliotecas. Para mayor detalle refiérase a la bibliografía correspondiente.

Sección A.1

El ciclo de simulación

PhysX posee una arquitectura multiprocesador para asegurar un mayor rendimiento en aplicaciones que involucran *rendering*, animación, inteligencia artificial, física, sonido u otros elementos en tiempo real. Por esta razón, la API desde su versión 2.2 incorpora un enfoque multihilos. Generalmente la aplicación principal corre bajo un hilo, mientras que los cálculos de la simulación corren en otro. Para lograr la sincronía entre ambos implementa un sistema de doble buffer, evitando un conflicto entre hilos.

El elemento principal en una aplicación de *PhysX* es la escena: *NxScene*, la cual contiene todos los elementos de la simulación física; desde la fuerza de gravedad hasta la geometría de cada uno de los actores del mundo virtual. Toda escena tiene asociado un intervalo de subdivisión del tiempo Δt para ejecutar cálculos de posiciones y velocidades, que se asigna mediante el método **setTiming**.

```
NxScene::setTiming();
```

Por cada intervalo de tiempo se ejecutan los siguientes pasos:

1. Se inicia la simulación.
2. Se verifica que todos los datos se encuentran en el hilo de simulación.
3. Se verifica continuamente si la simulación ha finalizado y todos los datos se encuentran en el buffer de cálculos.
4. Se intercambian los buffers para que los datos, resultado del cálculo, sean usados por el programa principal. Se procede al siguiente paso de la simulación regresando al paso 1.

Este proceso coincide con el ciclo de *render* de *OpenGL* y *Qt*, lo que facilita el acoplamiento entre SDK's. Un ciclo principal de simulación física dentro de un ciclo de *render* se vería como el siguiente programa:

```
NxScene      *gScene;

void NvidiaPhysX::RenderCallback()
{
...
    if (gScene)
    {
        GetPhysicsResults();
        ProcessInputs();
```

```
        StartPhysics();
    }
...
}

void NvidiaPhysX::GetPhysicsResults()
{
    // Verifica los resultados de gScene->simulate(deltaTime)
    while (!gScene->fetchResults(NX_RIGID_BODY_FINISHED, false));
}

void NvidiaPhysX::StartPhysics(){
          // Actualiza el incremento de tiempo
          NxReal gDeltaTime = UpdateTime();
          // Inicia cálculos de física
          // desde el último frame
          gScene->simulate(gDeltaTime);
          gScene->flushStream();
}
```

El método **GetPhysicsResults** permite asegurar que se han finalizado todos los cálculos del frame anterior antes de continuar con el siguiente paso. Cuando el buffer está listo se continúa con la simulación del intervalo *gDeltaTime* mediante **simulate**. La función **UpdateTime** obtiene el intervalo de tiempo transcurrido entre el frame anterior y el actual. Finalmente con **flushStream** se intercambian los buffers para que el programa principal use los resultados de la simulación.

El método **ProcessInputs** se encarga de procesar los datos del usuario, que consisten en modos de visualización o aplicación de fuerzas a los actores de la escena. Un ejemplo de programa para procesar entradas es el siguiente:

```
void NvidiaPhysX::ProcessInputs()
{
    ProcessKeys();

    // Muestra visualización en malla de alambre
    if (bDebugWireframeMode)
    {
        glDisable(GL_LIGHTING);
        gScene->visualize();
        gPhysicsSDK->visualize(gDebugRenderer);
        glEnable(GL_LIGHTING);
    }
}

void NvidiaPhysX::ProcessKeys()
{
    // Procesa teclas
    for (int i = 0; i < MAX_KEYS; i++)
```

```
    {
        if (!gKeys[i])  { continue; }

        switch (i)
        {
            ...
            // Aplica diferentes fuerzas a un actor
            // dependiendo de la tecla presionada
            case 'i':
              gForceVec = ApplyForceToActor(box,NxVec3(0,0,1),
                                            gForceStrength);
            break;
            case 'k':
              gForceVec = ApplyForceToActor(box,NxVec3(0,0,-1),
                                            gForceStrength);
            break;
            case 'j':
              gForceVec = ApplyForceToActor(box,NxVec3(+1,0,0),
                                            gForceStrength);
            break;
            case 'l':
              gForceVec = ApplyForceToActor(box,NxVec3(-1,0,0),
                                            gForceStrength);
            break;
            case 'u':
              gForceVec = ApplyForceToActor(box,NxVec3(0,+1,0),
                                            gForceStrength);
            break;
            case 'm':
              gForceVec = ApplyForceToActor(box,NxVec3(0,-1,0),
                                            gForceStrength);
            break;

            // Reposiciona al objeto en su posición
            // original
            case 't':
              box->setGlobalPosition(NxVec3(0,5,0));
            break;
        }
    }
}

NxVec3 NvidiaPhysX::ApplyForceToActor(NxActor* actor,
             const NxVec3& forceDir, const NxReal forceStrength)
{
    NxVec3 forceVec = forceStrength*forceDir;
    // Agrega una fuerza al objeto
    actor->addForce(forceVec);
    return forceVec;
}
```

Las entradas provienen de un dispositivo de interacción, en el ejemplo del teclado. El método **ProcessKeys** verifica qué tecla ha sido especificada por el usuario y aplica una fuerza sobre un objeto del mundo. Adicionalmente en el método **ProcessInputs** se llevan a cabo acciones de visualización, combinando entre métodos propios del SDK de *PhysX* e instrucciones de *OpenGL*.

Generalmente la comunicación entre el programa y dispositivos como el teclado, la brindan bibliotecas en un nivel más alto de encapsulamiento, que mediante métodos de *callback* establecen banderas a estados fácilmente reconocibles. Por ejemplo, cuando se presiona la tecla 'A' o algún clic proveniente del ratón. Algunas bibliotecas que implementan esta funcionalidad son: *Win32* (API de Windows), *GLUT*[27] y *Qt*.

Sección A.2

Programación de *Qt* con *OpenGL*

Qt[65] es un conjunto de bibliotecas de código abierto y herramientas diseñadas para el desarrollo de aplicaciones que requieren una interfaz de usuario (GUI), portable para plataformas Windows, MAC OS X y Linux. Actualmente se distribuye gratuitamente bajo una licencia no comercial.

El SDK contiene todo un conjunto de clases escritas en C/C++ y bibliotecas dinámicas con lo necesario para construir una aplicación basada en *widgets* o pequeñas vantanas con funcionalidad propia. Los *widgets* son los elementos básicos de toda aplicación con interfaz gráfica de usuario (GUI). Cada componente del GUI (por ejemplo, botones, etiquetas, editores de texto, páneles o regiones de despliegue) es un *widget* que tiene una posición y una funcionalidad en la aplicación principal.

La estructura básica de un programa en *Qt* para desplegar una simple ventana es la siguiente:

```
#include <QtGui>
#include <QApplication>
#include <QDesktopWidget>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    // Configuración y despliegue de widgets

    return app.exec();
}
```

Una de las funcionalidades más atractivas de *Qt* es el soporte que ofrece para desplegar gráficos acelerados con *OpenGL* usando su propia clase: **QGLWidget**. Esta clase implementa métodos abstractos y métodos para el manejo de *buffers*, así como subclases que permiten llevar a cabo la mayoría de las tareas necesarias para un render con *OpenGL*.

Los métodos más importantes de la clase **QGLWidget** son:

- **paintGL()**: Contiene todas las instrucciones en *OpenGL* para el despliegue. Tiene su similitud con el *callback* especificado con **glutDisplayFunc()** de la librería GLUT.
- **initializeGL()**: Contiene las instrucciones básicas para configurar iluminación y modos de *render*.
- **resizeGL()**: Contiene las instrucciones necesarias para readaptar dinámicamente el contenido gráfico ante un cambio de tamaño de la ventana. Tiene su similitud con el *callback* especificado con **glutReshapeFunc()** de la librería GLUT.
- **updateGL()**: Actualiza el panel de dibujo invocando a la función **glDraw()**. Tiene su similitud con la función **glutPostRedisplay()** de la librería GLUT.
- **swapBuffers()**: Intercambia los buffers de visualización en modo de *buffer* doble. Tiene su similitud con la función **glutSwapBuffers()** de la librería GLUT.

En *Qt*, una alternativa a la técnica de *callbacks*[3], o llamadas a procedimientos, es el uso de señales (*signals*) y receptores o *slots*. Una señal se emite cuanto sucede un evento particular. Un receptor o *slot* es una función que se ejecuta en respuesta a una señal. La forma de especificar qué señal corresponde a cada receptor se lleva a cabo mediante métodos `connect` de la clase *QObject*, inherente a todo *widget*.

El método `connect` tiene la siguiente especificación:

```
bool  connect ( const QObject * sender,
                const char * signal,
                const QObject * receiver,
                const char * method,
                Qt::ConnectionType type = Qt::AutoConnection );
```

El método asocia un objeto que envía una señal con otro objeto que la recibe, procesando el método asociado a ésta última. Un diagrama del funcionamiento de esta técnica es el mostrado en la Figura A.1. En este se observan cuatro objetos, cada uno de ellos define un conjunto de métodos de tipo `signals` y `slots`. El Objeto 1 enlaza sus señales con los receptores del Objeto 2 y el Objeto 4 mediante métodos `connect`. De esta forma, cuando se activa la senal 1 del Objeto 1, se ejecutan los métodos *slot* del Objeto 2. Cuando se activa la señal 2 del Objeto 1, se ejecuta el código del *slot* 1 del Objeto 4. De igual manera, cuando se activa la señal 1 del Objeto 3, se ejecuta el código del *slot* 3 en el Objeto 4.

[3]Punteros a funciones que se ejecutan en un programa cuando sucede un evento específico.

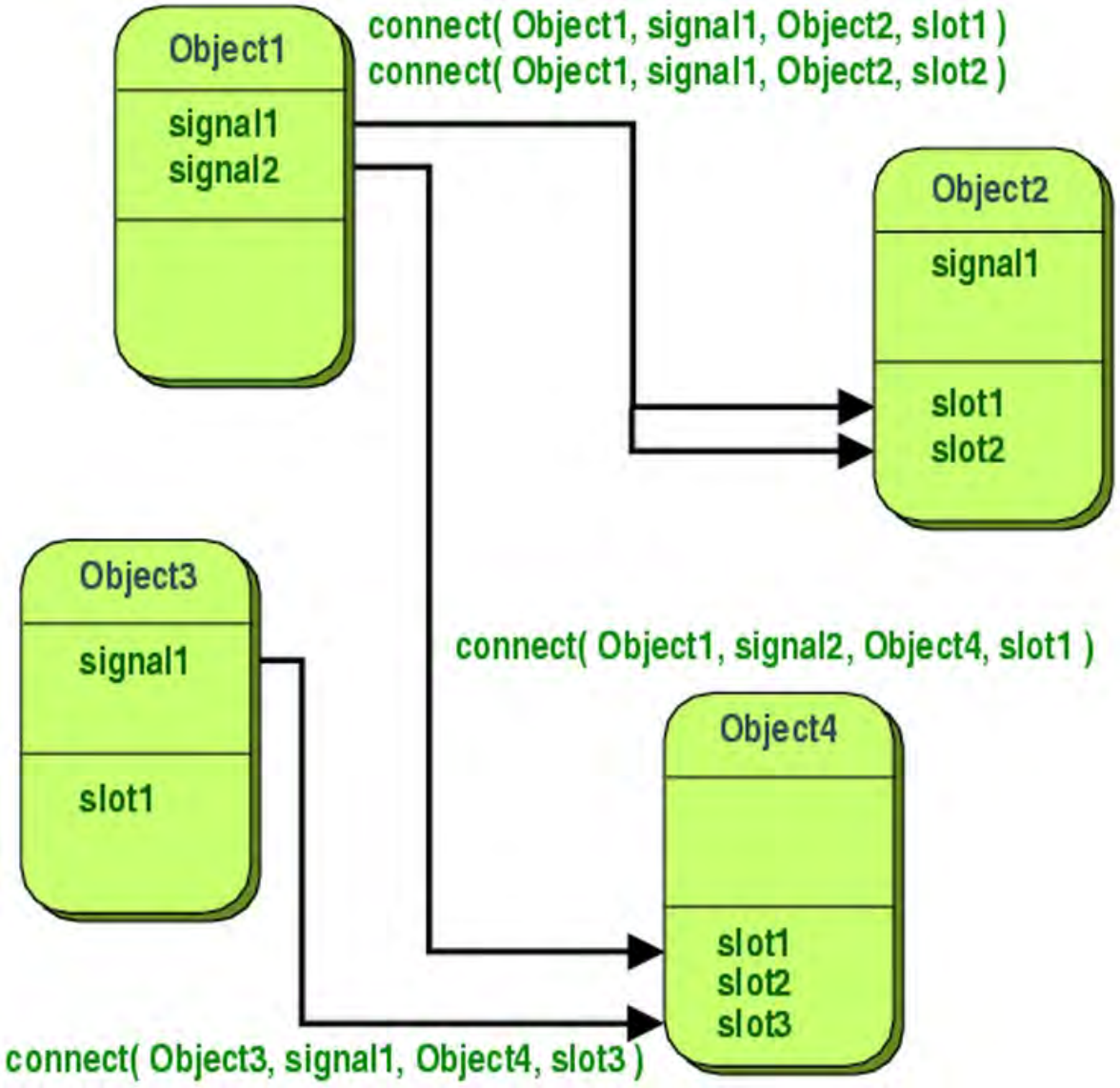


Figura A.1. *Conexión entre objetos mediante la técnica de señales y receptores en Qt (Qt 4.7: Signals and slots, 2008-2010 Nokia Corporation).*

Cuando se trata de comunicar objetos de una aplicación creada con *Qt* y la clase que maneja el código de *OpenGL* se puede usar fácilmente la técnica de señales y receptores. De este modo todo *widget* presente en la interfaz (como botónes o campos de texto) puede intercambiar información con la escena gráfica, procurando transformaciones en los objetos gráficos, habilitar o deshabilitar modos de render, manipular la iluminación, entre muchas otras funcionalidades.

Un ejemplo de aplicación que incluye *OpenGL* y *PhysX* se muestra en la Figura A.2. Ésta se compone por una ventana principal (*QMainWindow*), un menú (*QMenu*) y una región de trabajo (administrada por una clase manejadora de widgets: *MainWidget*. Dentro de la ventana podemos encontrar los *widgets* separados por páneles. El panel de la izquierda se compone por *widgets* de control y contiene botones, etiquetas de texto, cajas de texto, etc., que se comunican con la parte gráfica por medio de señales y *slots*. El panel derecho se compone de una región de despliegue administrada por la clase *QGLWidget*.

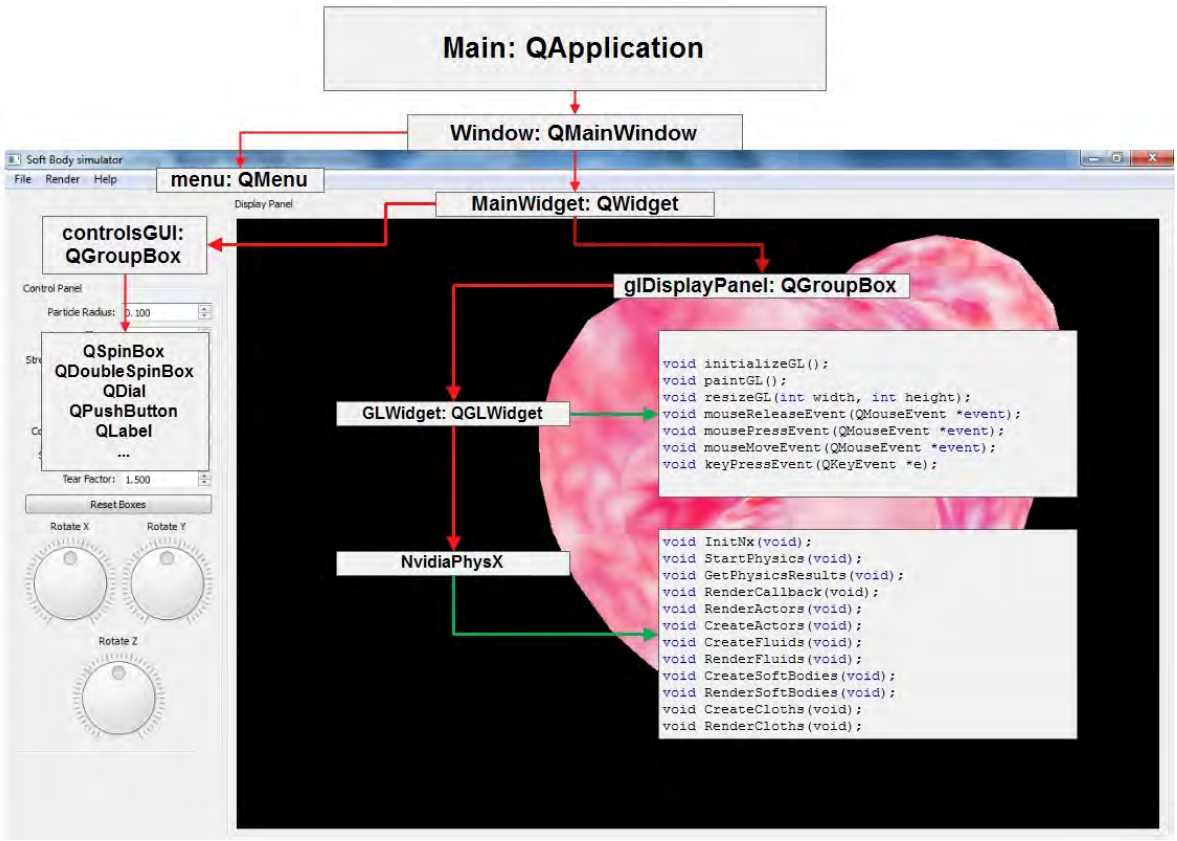


Figura A.2. *Estructura de software de un programa que incorpora Qt, OpenGL y simulaciones con PhysX.*

Sección A.3

Elementos de la programación de *PhysX*

Inicialización

El primer paso para la construcción de una aplicación en *PhysX* es la inicialización del SDK que se encargará de manejar toda la simulación y la creación de una escena. Para ello se crea una clase de tipo *NvidiaPhysX* cuyo constructor inicializa un objeto de tipo *NxPhysicsSDK* y otro de tipo *NxScene*, como se muestra en el siguiente código:

```
class NvidiaPhysX{
public:
    NvidiaPhysX();
    ~NvidiaPhysX();
    void                InitNx(void);
    ...
private:
    ...
    // SDK de PhysX y escena
    NxPhysicsSDK*       gPhysicsSDK;
    NxScene*            gScene;
    ...
};

void NvidiaPhysX::InitNx()
{
    ...
    // Inicialización del SDK
    gPhysicsSDK = NxCreatePhysicsSDK(NX_PHYSICS_SDK_VERSION);
    if (!gPhysicsSDK)  return;

    // Creación de la escena
    NxSceneDesc sceneDesc;
    sceneDesc.gravity.set(0.0, -9.81, 0.0);  // Aceleración de la
                                             // gravedad

    gScene = gPhysicsSDK->createScene(sceneDesc);
    ...
}

void GLWidget::initializeGL()
{
    ...
    physXwrl = new NvidiaPhysX();
    physXwrl->InitNx();
    ...
}
```

Configuración de parámetros del mundo

El segundo paso es la configuración de los parámetros del SDK y de la escena. Básicamente se requieren configurar dos tipos de parámetros: de visualización y de física. Los primeros permiten la depuración del programa en tiempo real y son muy útiles para el programador. Los segundos definen restricciones o condiciones para el cálculo de variables físicas. El parámetro más importante de este segundo grupo es el `NX_SKIN_WIDTH`, el cual permite establecer un umbral de penetración entre objetos durante una colisión. Si deseamos que este valor sea de 1 mm, por ejemplo, configuramos de la siguiente manera:

```
gPhysicsSDK->setParameter(NX_SKIN_WIDTH, 0.001); // 0.001 m = 1 mm
```

Algunos otros parámetros configurables son los siguientes:

```
gPhysicsSDK->setParameter(NX_CONTINUOUS_CD, true); // Colisión Continua
gPhysicsSDK->setParameter(NX_CCD_EPSILON, 0.01);   // Tolerancia
...
// Parámetros de visualización
gPhysicsSDK->setParameter(NX_VISUALIZATION_SCALE, true);
gPhysicsSDK->setParameter(NX_VISUALIZE_COLLISION_SHAPES, true);
gPhysicsSDK->setParameter(NX_VISUALIZE_ACTOR_AXES, true);
gPhysicsSDK->setParameter(NX_VISUALIZE_JOINT_LIMITS, true);
gPhysicsSDK->setParameter(NX_VISUALIZE_JOINT_LOCAL_AXES, true);
```

Materiales de la escena

Un material define cómo reaccionará un objeto ante una colisión y propiedades de las superficies. Esto permite que todos los objetos del mundo puedan interaccionar de forma predeterminada unos con otros. Se definen tres propiedades básicas de un material, su coeficiente de restitución, el coeficiente de fricción estática y el coeficiente de fricción dinámica.

```
NxMaterial* defaultMaterial = gScene->getMaterialFromIndex(0);

defaultMaterial->setRestitution(0.5);      // Restitution in the impact
defaultMaterial->setStaticFriction(0.5);   // Static Friction
defaultMaterial->setDynamicFriction(0.5); // Dinamic Friction
```

Creación de actores

Los actores son los elementos principales que interactúan en una escena y generalmente se crean en el método de inicialización o dinámicamente durante la simulación. Son

de tres tipos: dinámicos, cinemáticos y estáticos. Los primeros dos pueden considerarse como dinámicos, ya que son afectados por fuerzas externas, mientras que los últimos únicamente pueden cambiar al variar directamente su posición.

Todos los actores tienen asociado un descriptor de actor y un descriptor de cuerpo, que contienen información útil para que el motor de física calcule posiciones, rotaciones, velocidades, aceleraciones, etc. En el caso de actores estáticos el descriptor de cuerpo es nulo ya que no hay cálculos sobre el objeto.

Los actores se componen de una geometría que define su forma. Las formas básicas proporcionadas por el SDK son: plano, caja, esfera, cápsula, pirámide y pirámide truncada. Otras geometrías más complejas se construyen a partir de estas básicas o bien a partir de modelos en formato OBJ.

Tres funciones para crear actores dinámicos, cinemáticos y estáticos, respectivamente, son las siguientes:

```
NxActor* CreateDynamicActor()
{
    NxActorDesc actorDesc;
    NxBodyDesc  bodyDesc;

    // Se define una forma geométrica
    NxBoxShapeDesc boxDesc;
    boxDesc.dimensions  = NxVec3(1.5f, 1.5f, 1.5f);
    actorDesc.shapes.pushBack(&boxDesc);

    // Se asignan propiedades físicas
    actorDesc.density           = 1.0f;
    actorDesc.body              = &bodyDesc;
    actorDesc.globalPose.t      = NxVec3(3.0f, 0.0f, 0.0f);

    // Se inserta el actor en la lista de actores
    // de la escena
    NxActor *pActor = gScene->createActor(actorDesc);
    assert(pActor);

    return pActor;
}

NxActor* CreateKinematicActor()
{
    NxActorDesc actorDesc;
    NxBodyDesc  bodyDesc;
    bodyDesc.flags              |= NX_BF_KINEMATIC;

    // Se define una forma geométrica
    NxCapsuleShapeDesc capsuleDesc;
    capsuleDesc.radius  = 1.0f;
    capsuleDesc.height  = 1.5f;
```

```
    actorDesc.shapes.pushBack(&capsuleDesc);

    // Se asignan propiedades físicas
    actorDesc.density           = 1.0f;
    actorDesc.body              = &bodyDesc;
    actorDesc.globalPose.t      = NxVec3(0.0f, 2.0f, 0.0f);

    // Se inserta el actor en la lista de actores
    // de la escena
    NxActor *pActor = gScene->createActor(actorDesc);
    assert(pActor);

    return pActor;
}


NxActor* CreateStaticActor()
{
    NxActorDesc actorDesc;
    NxBodyDesc  bodyDesc;

    // Se define una forma geométrica
    NxSphereShapeDesc sphereDesc;
    sphereDesc.radius   = 1.5f;
    actorDesc.shapes.pushBack(&sphereDesc);

    // Se asignan propiedades físicas
    actorDesc.body              = NULL;
    actorDesc.globalPose.t      = NxVec3(-3.0f, 3.0f, 0.0f);

    // Se inserta el actor en la lista de actores
    // de la escena
    NxActor *pActor = gScene->createActor(actorDesc);
    assert(pActor);

    return pActor;
}
```

Colisiones

En *PhysX* las colisiones se manejan por grupos *NxCollisionGroup*. Todos los objetos pertenecientes a un grupo interaccionan únicamente entre ellos y de manera predeterminada todos los objetos pertenecen al grupo 0, y pueden cambiar de grupo mediante el método **NxShape::setGroup**. Si no hay restricciones en cuanto a colisiones no es necesario configurar colisiones por cada objeto.

Sección A.4

Transformaciones

Las transformaciones son una de las operaciones más importantes sobre los objetos de una escena gráfica. Las transformaciones básicas son: escalamiento, rotación y traslación, que generalmente operan sobre puntos de coordenadas (x, y, z) (*NxVec3*) o matrices de orientación (*NxMat33*), y se representan mediante matrices de transformación (*NxMat33*, *NxMat34*) y cuaterniones (*NxQuat*).

Para simular partículas se toma la posición de cada una de ellas. Sobre un objeto rígido se usa su centro de masa y su sistema de ejes local para determinar su orientación; mientras que para simular objetos deformables se toma cada nodo de la malla y su sistema de ejes local. La combinación de posición y orientación de un objeto se denomina Pose.

Todo objeto tiene asociado un espacio de transformación local y un espacio de transformación global; el primero nos permite realizar operaciones con respecto al origen del objeto, mientras que el segundo nos permite realizar operaciones sobre el origen de coordenadas del mundo. Para calcular una pose relativa con respecto a uno u otro espacio se siguen las siguientes implementaciones:

- Pose local:

```
NxActor* actor;
...
NxVec3   localPose, worldPose;
NxMat34  transfMat, invMat;

worldPose = NxVec3(0,0,1);
transfMat = actor->getGlobalPose();
transfMat.getInverse(invMat);
localPose = invMat * worldPose;
```

- Pose global:

```
NxActor* actor;
...
NxVec3   localPose, worldPose;
NxMat34  transfMat;

localPose = NxVec3(0,0,1);
transfMat = actor->getGlobalPose();
worldPose = transfMat * localPose;
```

Para determinar la orientación de un objeto, es decir, qué tanto está inclinado con respecto a cada uno de los ejes coordenados del mundo, se emplea una matriz compuesta por

tres vectores axiales orientados. Cada fila de la matriz representa el eje X, Y ó Z relativo a dicho objeto. Se pueden obtener los ejes de orientación de un objeto mediante el siguiente código:

```
NxMat33 orientation;
NxVec3 xAxis, yAxis, zAxis;

orientation = actor->getGlobalOrientation();
orientation.getRow(0, xAxis);
orientation.getRow(1, yAxis);
orientation.getRow(2, zAxis);
```

Para orientar un objeto basta especificar los ángulos de inclinación para cada eje y operar sobre la matriz de orientación del objeto, como se muestra en el siguiente código (generalmente se usan cuateriones para efectuar operaciones de rotación; sin embargo, también puede obtenerse el mismo resultado usando matrices homogéneas):

```
NxActor* actor;
...
// Ángulos eulerianos
NxVec3 eulerAngle = NxVec3(90,90,30);

// Cuaternión
NxQuat qX, qY, qZ;
qX.fromAngleAxis(eulerAngle.x, NxVec3(1,0,0));
qY.fromAngleAxis(eulerAngle.y, NxVec3(0,1,0));
qZ.fromAngleAxis(eulerAngle.z, NxVec3(0,0,1));

// Matriz de orientación
NxMat33 orientX, orientY, orientZ;
orientX.fromQuat(qX);
orientY.fromQuat(qY);
orientZ.fromQuat(qZ);

NxMat33 orient;
orient = orientZ*orientY*orientX;  // Usa ejes globales
// orient = orientX*orientY*orientZ;  // Usa ejes locales

// Asigna la orientación calculada al objeto
actor->setGlobalOrientation(orient);
```

Sección A.5

Simulación de objetos dinámicos

El SDK 2.8.1 tiene la capacidad de simulación para estructuras más complejas basadas en física; estas son: telas, fluidos, cuerpos blandos y campos de fuerzas (Figura A.3). Estos objetos más complejos pueden interaccionar con los demás elementos de la escena mediante interacciones en dos direcciones (*two-way*) o en una dirección (*one-way*). Esto significa que pueden reaccionar ante una colisión con una fuerza en dirección contraria, afectando al objeto que lo colisionó; o bien, reaccionar ante la colisión pero no afectar al objeto que colisionó.

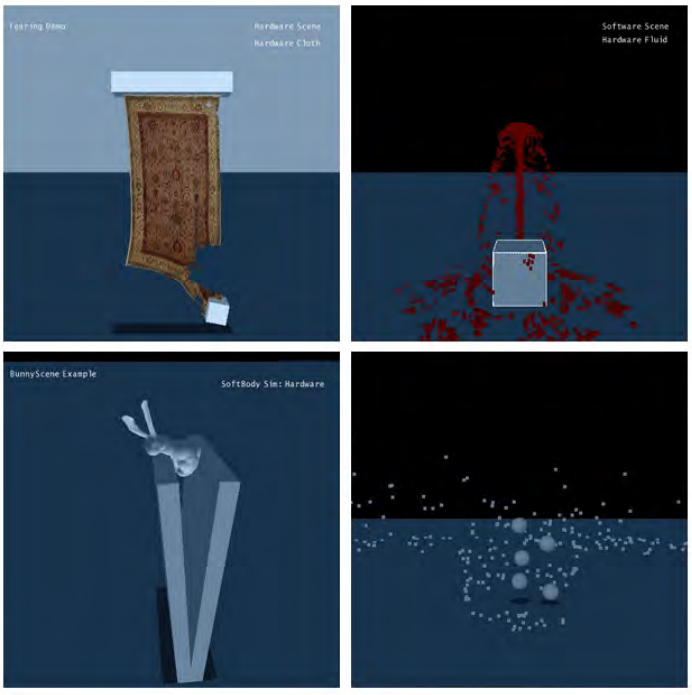


Figura A.3. *Simulación de estructuras complejas usando NVIDIA PhysX (NVIDIA PhysX training Program 2008).*

A.5.1. Simulación de telas

La simulación de telas permite la representación de banderas, ropa, cortinas, alfombras, etc. La simulación se efectúa sobre mallas de superficie triangulares mediante un método de masas y resortes, abordado en el Capítulo 2. El siguiente código muestra la configuración básica de una tela, en el que se crea un descriptor con propiedades físicas y de colisión. La creación de la geometría con la clase *Cloth* es un proceso más extenso, por lo que no se aborda aquí.

```
void NvidiaPhysX::InitNx()
{
    ...
    CreateCloth();
    ...
}

void NvidiaPhysX::CreateCloth()
{

    // Creamos el descriptor
    NxClothDesc clothDesc;

    // Propiedades espaciales
    clothDesc.globalPose.t = NxVec3(0,10,0);
    clothDesc.globalPose.M.rotX(-NxHalfPiF32);
    clothDesc.thickness = 0.2;

    // Propiedades físicas
    clothDesc.bendingStiffness = 0.5;
    clothDesc.windAcceleration = NxVec3(-20, 12, -1);

    // Banderas de interacción
    clothDesc.flags |= NX_CLF_BENDING;
    clothDesc.flags |= NX_CLF_COLLISION_TWOWAY | NX_CLF_VISUALIZATION;

    // Modo de cálculos por PPU
    if (gHardwareCloth)
        clothDesc.flags |= NX_CLF_HARDWARE;

    // Creación de la geometría
    Cloth* clothGeometry= new Cloth(gScene, clothDesc);

    // Se guarda la geometría y sus propiedades en
    // una lista de telas.
    gCloths.push_back(clothGeometry);

}
```

A.5.2. Simulación de fluidos

La simulación de fluidos permite la representación de líquidos y gases usando sistemas de partículas y emisores. Se pueden crear cascadas, contenedores de agua, sangrados, etc. Cuando una partícula de un fluido colisiona con un objeto rígido se agrega un impulso a la dinámica de esa partícula, así se crean efectos muy realistas. De forma predeterminada las colisiones entre formas estáticas y dinámicas están activas. Un ejemplo

de código para la creación de fluidos con *PhysX* se muestra a continuación:

```
void NvidiaPhysX::InitNx()
{
    ...
    CreateFluid();
    ...
}

void NvidiaPhysX::CreateFluid(void){

    // Datos de la partícula
    NxParticleData particles;
    particles.numParticlesPtr = &gParticleBufferNum;
    particles.bufferPos       = &gParticleBuffer[0].x;
    particles.bufferPosByteStride       = sizeof(NxVec3);

    // Descriptor del fluido
    NxFluidDesc fluidDesc;
    fluidDesc.maxParticles             = gParticleBufferCap;
    fluidDesc.kernelRadiusMultiplier   = KERNEL_RADIUS_MULTIPLIER;
    fluidDesc.restParticlesPerMeter    = REST_PARTICLES_PER_METER;
    fluidDesc.motionLimitMultiplier    = MOTION_LIMIT_MULTIPLIER;
    fluidDesc.packetSizeMultiplier     = PACKET_SIZE_MULTIPLIER;

    // Propiedades físicas
    fluidDesc.stiffness      = 50;
    fluidDesc.viscosity      = 22;
    fluidDesc.restDensity    = 1000;
    fluidDesc.damping        = 0;
    fluidDesc.restitutionForStaticShapes      = 0.4;
    fluidDesc.dynamicFrictionForStaticShapes = 0.3;

    // Propiedades de la simulación
    fluidDesc.collisionResponseCoefficient    = 0.5f;
    fluidDesc.collisionDistanceMultiplier     = 0.1f;
    fluidDesc.simulationMethod    = NX_F_SPH;

    fluidDesc.particlesWriteData = particles;
    fluidDesc.flags &= ~NX_FF_HARDWARE;
    fluidDesc.flags |= NX_FF_COLLISION_TWOWAY;

    // Se crea el fluido
    NxFluid* fl = gScene->createFluid(fluidDesc);
}
```

A.5.3. Simulación de cuerpos blandos

La simulación de cuerpos blandos permite la representación de objetos deformables de geometría compleja. Así, pueden simularse tejidos blandos, terrenos deformables, materiales elásticos, etc. Un fragmento de código para la implementación de un cuerpo deformable en *PhysX* es el mostrado a continuación:

```
void NvidiaPhysX::InitNx()
{
    ...
    CreateSoftBody();
    ...
}

void NvidiaPhysX::CreateSoftBodie(void){

    // Descriptor de cuerpo deformable
    NxSoftBodyDesc softBodyDesc;

    // Propiedades espaciales
    softBodyDesc.globalPose.t = NxVec3(0,0,0);
    softBodyDesc.particleRadius = 0.1f;

    // Propiedades físicas
    softBodyDesc.volumeStiffness = 0.5f;
    softBodyDesc.stretchingStiffness = 0.5f;
    softBodyDesc.friction = 1.0f;
    softBodyDesc.dampingCoefficient = 0.5f;
    softBodyDesc.density = 1.0;

    // Propiedades de simulación
    softBodyDesc.collisionResponseCoefficient = 0.2f;
    softBodyDesc.solverIterations = 5;

    softBodyDesc.flags |= NX_SBF_VOLUME_CONSERVATION;
    softBodyDesc.flags |= NX_SBF_SELFCOLLISION;
    softBodyDesc.flags |= NX_SBF_COLLISION_TWOWAY;
    softBodyDesc.flags |= NX_SBF_DAMPING;

    // Modo de cálculo con PPU
    if (gHardwareSoftBodySimulation)
        softBodyDesc.flags |= NX_SBF_HARDWARE;

    // Creamos la geometría del objeto
    SoftBody *softBody = new SoftBody(gScene, softBodyDesc);

    // Guardamos el objeto deformable
    gSoftBodies.push_back(softBody);

}
```

A.5.4. Simulación de campos de fuerzas

Un campo de fuerzas se define como un cuerpo con dos propiedades principales: un volumen de actividad con elementos de forma (similares a los actores) y una función que define el campo de fuerzas que actúan sobre éstos. Cuando un objeto entra dentro del volumen del campo de fuerzas se aplica una fuerza al objeto. Un fragmento de código para configurar un campo de fuerzas en *PhysX* se muestra a continuación:

```
void NvidiaPhysX::InitNx()
{
    ...
    CreateVortex();
    ...
}

void NvidiaPhysX::CreateVortex(const NxVec3& pos,
                               NxActor* actor, NxScene* scene)
{

    // Descriptor del campo de fuerzas
    NxForceFieldDesc ffDesc;

    // Kernel
    NxForceFieldLinearKernelDesc        lKernelDesc;
    NxForceFieldLinearKernel*           linearKernel;

    // Propiedades espaciales
    ffDesc.coordinates = NX_FFC_CYLINDRICAL;
    ffDesc.actor       = actor;
    lKernelDesc.constant       = NxVec3(-30, 4.0f, 0);
    lKernelDesc.positionTarget = NxVec3(3,0,0);

    NxMat33 m;
    m.zero();
    m(0,0) = 10;
    m(0,1) = -5;
    m(0,2) = 0;
    // Propiedades físicas
    lKernelDesc.positionMultiplier  = m;
    lKernelDesc.noise               = NxVec3(5,5,5);
    lKernelDesc.velocityTarget      = NxVec3(0,0,30);
    m.diagonal(NxVec3(1,1,1));
    lKernelDesc.velocityMultiplier  = m;

    lKernelDesc.falloffLinear     = NxVec3(5.0f, 0, 0);
    lKernelDesc.falloffQuadratic  = NxVec3(5.0f, 0, 0);
    linearKernel    = scene->createForceFieldLinearKernel(lKernelDesc);
    ffDesc.kernel   = linearKernel;
    ffDesc.flags    = 0;
```

```
    // Se crea el campo de fuerzas
    m_forceField = scene->createForceField(ffDesc);

    // Se configura el volumen del campo de fuerzas
    NxForceFieldShape* s = NULL;
    NxBoxForceFieldShapeDesc b;
    b.dimensions = NxVec3(5, 7, 5);
    b.pose.t     = NxVec3(0, 3.5f, 0);
    s = m_forceField->getIncludeShapeGroup().createShape(b);

    NxForceFieldShapeGroupDesc sgDesc;
    sgDesc.flags    = NX_FFSG_EXCLUDE_GROUP;
    m_excludeGroup = scene->createForceFieldShapeGroup(sgDesc);

    NxBoxForceFieldShapeDesc exclude;
    exclude.dimensions = NxVec3(2.25f, 1.5f, 1.75f);
    exclude.pose.t     = NxVec3(8.85f, 1.5f, -10.3f);
    m_excludeShape     = m_excludeGroup->createShape(exclude);

    m_forceField->addShapeGroup(*m_excludeGroup);
}
```

APÉNDICE B

Modelo de elasticidad para objetos deformables

Dependiendo del objeto deformable de estudio, la teoría de la elasticidad puede formularse en términos de muchos tipos de variables, escalares o vectoriales. Ejemplos de cantidades escalares encontramos a la masa, la densidad, la temperatura, entre otras; mientras que el desplazamiento y la rotación son ejemplos de cantidades vectoriales. Adicionalmente, la teoría de la elasticidad requiere de formulaciones matriciales o de variables matriciales para el planteamiento de un modelo. En el modelado matemático, a estas últimas cantidades se les conoce como tensores [1]. Para el análisis de materiales deformables son de gran importancia: el tensor de presión o de *stress* y el tensor de deformación (*strain*). A continuación analizaremos algunos conceptos principales para el planteamiento de un modelo biomecánico desde un punto de vista más formal, con el objetivo de adaptarlo a un modelo que represente la deformación de un objeto, y en su forma aplicable en la simulación de tejido blando.

Sección B.1

Fuerzas internas, externas y de frontera

En general, un objeto se encuentra bajo la acción de tres tipos de fuerzas: internas, externas y de frontera. En la mecánica de los medios continuos, las primeras dos se distribuyen de manera continua dentro del cuerpo del objeto; mientras que las externas se encuentran fuera de él, como consecuencia de su interacción con el exterior.

Las fuerzas internas o del cuerpo, son proporcionales a su masa y reaccionan

[1] Para conocer más acerca del álgebra de tensores, consultar: Sadd [67] y Allen [3].

directamente ante un agente externo mediante la propagación de fuerzas en la frontera. Ejemplos de este tipo de fuerzas son el peso debido a la gravedad, las fuerzas magnéticas, y las fuerzas de inercia. La Figura B.1 muestra un ejemplo de la acción que provocan las fuerzas internas sobre un objeto deformable.

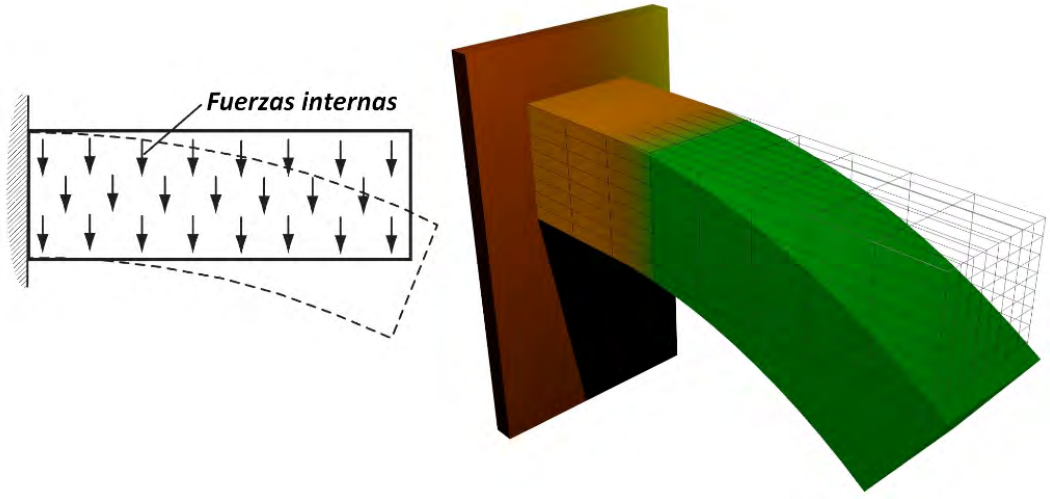


Figura B.1. *Deformación debida a fuerzas internas, tales como el peso* $\underline{W} = m\underline{g}$ *(*$\underline{g} = 9.81\,j[m/seg^2]$*).*

Las fuerzas de frontera o de superficie siempre actúan sobre la superficie del objeto y son el resultado del contacto físico con otro cuerpo. Ejemplo de este tipo de fuerzas es la fricción ($\underline{F}_r = \mu\underline{N}$, donde μ es el coeficiente de fricción y $\underline{N}$ es el vector normal a la superficie de contacto). La fricción también es llamada fuerza de rozamiento. La Figura B.2 muestra dos ejemplos de fuerzas en la frontera.

En el cuerpo humano, es muy sencillo distinguir este tipo de fuerzas, considerando que éste mismo se encuentra formado por una gran variedad de tejidos y que interaccionan constantemente con fuerzas de todo tipo. Supongamos entonces que se lleva a cabo la aplicación de una inyección intravenosa, la aguja imprime una fuerza externa sobre un tejido blando, en este caso la piel que reacciona con una deformación y al mismo tiempo se opone a la introducción del cuerpo extraño por medio de las fuerzas internas. En otro ejemplo, durante una intervención no invasiva con endoscopio, uno de los obstáculos es la fricción entre el tejido y la herramienta, por lo cual se hace uso de sustancias que provocan el libre deslizamiento y evitan la fricción con el tejido. Al llevarse a cabo el rozamiento entre la herramienta y el tejido, las fuerzas que se generan se encuentran en la superficie del tubo o tracto, identificándose rápidamente fuerzas en la frontera.

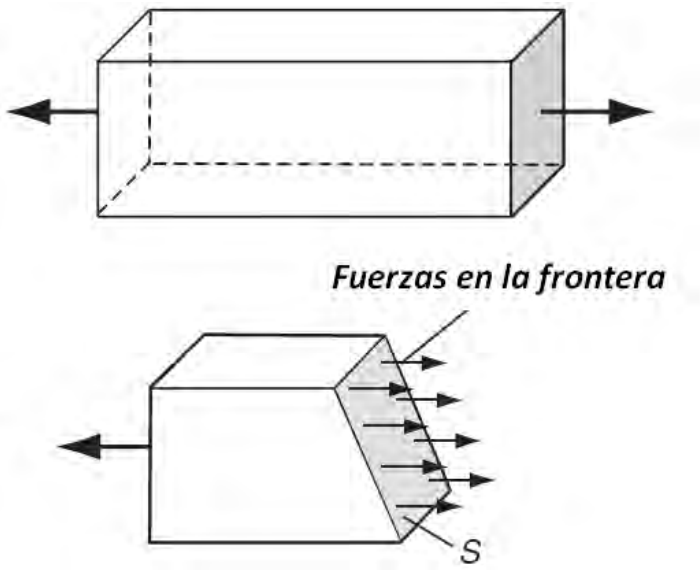


Figura B.2. *Fuerzas de frontera o de superficie sobre un objeto. Arriba, fuerza perpendicular a la superficie. Abajo, fuerzas no perpendiculares a la superficie [67].*

Sección B.2

Vector de Tracción y el Tensor de *stress*

Cuando un objeto se encuentra bajo la influencia de fuerzas externas que se aplican sobre su frontera, se dice que se genera un esfuerzo de tracción, que tiende a deformarlo en forma de compresión o estiramiento [67], modificando su energía interna. Así, sea el objeto mostrado en la Figura B.3. El vector de tracción o de *stress* se define como:

$$\underline{T}^n(\underline{x}, \underline{n}) = \lim_{\Delta A \to 0} \frac{\Delta \underline{F}}{\Delta A}$$

donde $\underline{x}$ es un punto del objeto, $\underline{n}$ es el vector normal unitario a una sección de área A y $\underline{F}$ el vector de fuerza en el punto $\underline{x}$.

Supongamos ahora, que la superficie ΔA coincide con cada uno de los tres planos coordenados y la normal apunta en la dirección positiva de cada eje coordenado, como se muestra en la Figura B.4. En este caso, el vector de tracción en la dirección de cada vector unitario normal a cada cara del cubo infinitesimal, puede escribirse como:

$$\begin{aligned}
\underline{T}^n(\underline{x}, \underline{n} = i) &= \sigma_x i + \tau_{xy} j + \tau_{xz} k \\
\underline{T}^n(\underline{x}, \underline{n} = j) &= \tau_{yx} i + \sigma_y j + \tau_{yz} k \\
\underline{T}^n(\underline{x}, \underline{n} = k) &= \tau_{zx} i + \tau_{zy} j + \sigma_z k
\end{aligned}$$

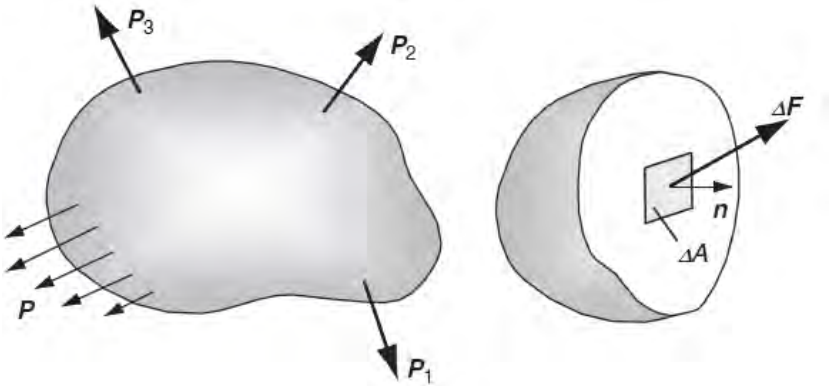


Figura B.3. *Objeto influenciado por fuerzas externas. A la izquierda, carga externa sobre el cuerpo. A la derecha, cambio de la fuerza en una sección de área infinitesimal [67].*

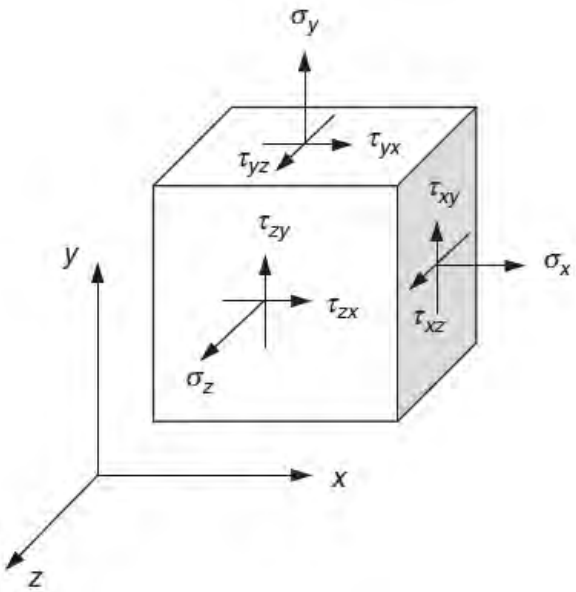


Figura B.4. *Componentes del Tensor de stress, ejemplificados en un cubo infinitesimal [67].*

Donde $\{i, j, k\}$ son los vectores unitarios, en la dirección de cada eje coordenado; y $\{\sigma_x, \tau_{xy}, \tau_{xz}, \tau_{yx}, \sigma_y, \tau_{yz}, \tau_{zx}, \tau_{zy}, \sigma_z\}$ son las nueve componentes del tensor de stress, definido como:

$$\underline{\underline{\sigma}} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} = \sigma_{ij}$$

Sección B.3

Caracterización del modelo

B.3.1. Ecuaciones del equilibrio

El campo de esfuerzos de tracción en un sólido elástico se encuentra continuamente distribuido dentro del cuerpo y únicamente determinado por las cargas aplicadas [67]. Suponiendo que el cuerpo se inicia en estado de equilibrio, se debe satisfacer que la suma de fuerzas y momentos es igual a cero. Si el cuerpo entero se encuentra en equilibrio, entonces también lo estarán cada una de sus partes que lo constituyen. Por lo tanto, podemos partir cualquier sólido en subdominios para aplicar el principio de equilibrio en cada región.

Siguiendo este enfoque, las ecuaciones de equilibrio serán válidas para cualquier punto en el material y en cualquier región finita de cada uno de sus subdominios. Entonces, se cumple que:

$$\begin{aligned} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} + F_x &= 0 \\ \frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} + F_y &= 0 \\ \frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + F_z &= 0 \end{aligned}$$

O bien,

$$\nabla \cdot \underline{\underline{\sigma}} + \underline{F} = \underline{0}$$

donde $\underline{F}$ es el vector de fuerzas necesario para lograr el equilibrio del cuerpo.

B.3.2. Ecuaciones constitutivas

Las relaciones que caracterizan las propiedades físicas de un material se les conoce como *ecuaciones constitutivas* [67] [29]. Generalmente, estas relaciones expresan el esfuerzo como una función de la deformación (*stress-strain*).

Un modelo describe un objeto deformable cuando éste incluye el proceso de estabilización del objeto; esto es, cuando el objeto recupera su configuración original al eliminan las cargas que provocan la deformación, una especie de amortización.

Los objetos elásticos pueden comportarse de manera lineal o no lineal. Muchos materiales elásticos muestran un comportamiento lineal, incluyendo los metales, plásticos, cerámicas, madera, roca, concreto. Otros, como el caso de tejido biológico, las propiedades elastodinámicas son más complejas. En términos mecánicos, el tejido orgánico se comporta de manera no lineal, es anisotrópico, viscoelástico (se comporta como un fluido y un sólido deformable) no homogéneo, y en algunos casos visco-plástico (se comporta como un fluido y un sólido semi-deformable) [40]. En la Figura B.5 puede observarse el comportamiento lineal y no lineal de algunos materiales elásticos.

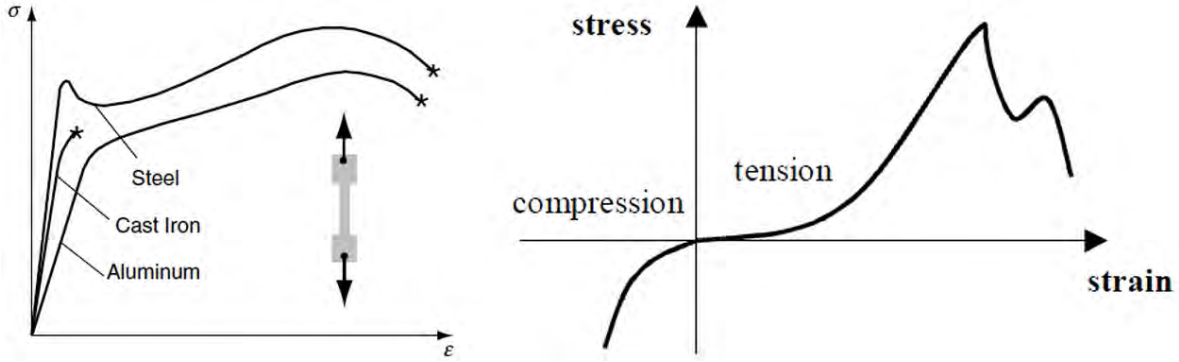


Figura B.5. *Comportamiento elástico de algunos materiales. A la izquierda, comportamiento elastodinámico relativamente lineal [67] (los asteriscos indican el punto de quiebre o falla). A la derecha, comportamiento elastodinámico no lineal del Tejido orgánico[40].*

Para caracterizar experimentalmente un material, generalmente se le somete a una prueba simple de tensión. La deformación se obtiene por el cambio en la longitud original del material al ser sometido a una carga experimental. Para el caso de tejido vivo, las pruebas se realizan con tejidos animales, como por ejemplo, en tejido bovino[40]. Los estudios en tejido humano actualmente son muy escasos y dada la dependencia de los parámetros conforme a la muestra, no permite obtener un conjunto de valores representativo. Algunos métodos, más sofisticados, usan el formalismo matemático con estos resultados empíricos para general métodos híbridos que permiten describir el comportamiento de estructuras elásticas [53].

B.3.3. Ley de Hooke

Para obtener un modelo de tejido blando, supondremos que en cierto rango, éste se comporta de manera lineal; lo cual nos permitirá aplicar algunos conceptos de la teoría

de la elasticidad. Así, podemos calcular cada componente de *stress* en términos de cada componente de deformación. Entonces:

$$\begin{aligned}
\sigma_x &= C_{11}e_x + C_{12}e_y + C_{13}e_z + 2C_{14}e_{xy} + 2C_{15}e_{yz} + 2C_{16}e_{zx} \\
\sigma_y &= C_{21}e_x + C_{22}e_y + C_{23}e_z + 2C_{24}e_{xy} + 2C_{25}e_{yz} + 2C_{26}e_{zx} \\
\sigma_z &= C_{31}e_x + C_{32}e_y + C_{33}e_z + 2C_{34}e_{xy} + 2C_{35}e_{yz} + 2C_{36}e_{zx} \\
\tau_{xy} &= C_{41}e_x + C_{42}e_y + C_{43}e_z + 2C_{44}e_{xy} + 2C_{45}e_{yz} + 2C_{46}e_{zx} \\
\tau_{yz} &= C_{51}e_x + C_{52}e_y + C_{53}e_z + 2C_{54}e_{xy} + 2C_{55}e_{yz} + 2C_{56}e_{zx} \\
\tau_{zx} &= C_{61}e_x + C_{62}e_y + C_{63}e_z + 2C_{64}e_{xy} + 2C_{65}e_{yz} + 2C_{66}e_{zx}
\end{aligned}$$

donde los coeficientes C_{ij} son los parámetros del material,

$$e_x = \frac{\partial u}{\partial x} \qquad e_y = \frac{\partial v}{\partial y} \qquad e_z = \frac{\partial w}{\partial z}$$

$$e_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)$$

$$e_{yz} = \frac{1}{2}\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)$$

$$e_{zx} = \frac{1}{2}\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)$$

y los factores de 2 representan la simetría de la deformación. En forma matricial, este último sistema de ecuaciones puede escribirse como:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} & C_{15} & C_{16} \\ C_{21} & C_{22} & C_{23} & C_{24} & C_{25} & C_{26} \\ C_{31} & C_{32} & C_{33} & C_{34} & C_{35} & C_{36} \\ C_{41} & C_{42} & C_{43} & C_{44} & C_{45} & C_{46} \\ C_{51} & C_{52} & C_{53} & C_{54} & C_{55} & C_{56} \\ C_{61} & C_{62} & C_{63} & C_{64} & C_{65} & C_{66} \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_z \\ 2e_{xy} \\ 2e_{yz} \\ 2e_{zx} \end{bmatrix}$$

Las componentes C_{ij} son llamadas *módulo elástico* y tienen unidades de presión (fuerza/área).

Si el material es homogéneo, el comportamiento elástico no varia espacialmente, y el módulo elástico es constante. Asimismo, si el material es isotrópico, se pueden introducir dos constantes: μ y λ. La primera se le conoce como *módulo de cizalla* o *módulo de rigidez*. A la segunda se le conoce como *constante de Lamè*. Así, el tensor de *stress* puede escribirse como:

$$\sigma_{ij} = \lambda e_{kk}\delta_{ij} + 2\mu e_{ij}$$

Entonces, el vector $\underline{\sigma}$ puede ser expresado como:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{bmatrix} = \begin{bmatrix} \lambda+2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \begin{bmatrix} e_x \\ e_y \\ e_z \\ 2e_{xy} \\ 2e_{yz} \\ 2e_{zx} \end{bmatrix}$$

que en forma de sistemas de ecuaciones:

$$\begin{aligned} \sigma_x &= \lambda(e_x+e_y+e_z)+2\mu e_x \\ \sigma_y &= \lambda(e_x+e_y+e_z)+2\mu e_y \\ \sigma_z &= \lambda(e_x+e_y+e_z)+2\mu e_z \\ \tau_{xy} &= 2\mu e_{xy} \\ \tau_{yz} &= 2\mu e_{yz} \\ \tau_{zx} &= 2\mu e_{zx} \end{aligned}$$

A estas últimas relaciones se les conoce como las *ecuaciones de la Ley de Hooke para sólidos elásticos lineales e isotrópicos*.

B.3.4. Formulación del desplazamiento: Ecuaciones de Navier

Empleando las ecuaciones de la Ley de Hooke, el problema de la formulación del desplazamiento se reduce a la determinación de los coeficientes del vector $\underline{\sigma}$. Entonces, si calculamos el cambio en cada una de las direcciones del vector $\underline{U} = (u, v, w)$, tenemos que:

$$\begin{aligned} \sigma_x &= \lambda\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right)+2\mu\frac{\partial u}{\partial x} \\ \sigma_y &= \lambda\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right)+2\mu\frac{\partial v}{\partial y} \\ \sigma_z &= \lambda\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}\right)+2\mu\frac{\partial w}{\partial z} \\ \tau_{xy} &= \mu\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right) \\ \tau_{yz} &= \mu\left(\frac{\partial v}{\partial z}+\frac{\partial w}{\partial y}\right) \\ \tau_{zx} &= \mu\left(\frac{\partial w}{\partial x}+\frac{\partial u}{\partial z}\right) \end{aligned}$$

Empleando estas expresiones en las ecuaciones de equilibrio:

$$\begin{aligned}
\frac{\partial}{\partial x}\sigma_x + \frac{\partial}{\partial y}\tau_{xy} + \frac{\partial}{\partial z}\tau_{xz} + F_x &= 0 \\
\frac{\partial}{\partial x}\tau_{yx} + \frac{\partial}{\partial y}\sigma_y + \frac{\partial}{\partial z}\tau_{yz} + F_y &= 0 \\
\frac{\partial}{\partial x}\tau_{zx} + \frac{\partial}{\partial y}\tau_{zy} + \frac{\partial}{\partial z}\sigma_z + F_z &= 0
\end{aligned}$$

finalmente obtenemos:

$$\begin{aligned}
\mu\nabla^2 u + (\lambda+\mu)\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + F_x &= 0 \\
\mu\nabla^2 v + (\lambda+\mu)\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + F_y &= 0 \\
\mu\nabla^2 w + (\lambda+\mu)\frac{\partial}{\partial z}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) + F_z &= 0
\end{aligned}$$

donde el Laplaciano está dado por:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$$

Finalmente, este sistema puede expresarse en forma vectorial como:

$$\mu\nabla^2 \underline{U} + (\lambda+\mu)\nabla(\nabla\cdot\underline{U}) + \underline{F} = \underline{0}$$

A esta expresión se le conoce como *ecuación de Navier* y representa el modelo formal para describir el comportamiento elástico de un material desde el punto de vista de la mecánica de los medios continuos.

B.3.5. Demostración del modelo de elasticidad de Navier-Stokes

$$\begin{aligned}
\sigma_x &= \lambda(u_x + v_y + w_z) + 2\mu u_x \\
\sigma_y &= \lambda(u_x + v_y + w_z) + 2\mu v_y \\
\sigma_z &= \lambda(u_x + v_y + w_z) + 2\mu w_z \\
\tau_{xy} &= \mu(u_y + v_x) \\
\tau_{yz} &= \mu(v_z + w_y) \\
\tau_{zx} &= \mu(w_x + u_z)
\end{aligned}$$

$$
\begin{aligned}
\frac{\partial}{\partial x}\sigma_x + \frac{\partial}{\partial y}\tau_{xy} + \frac{\partial}{\partial z}\tau_{xz} + F_x &= 0 \\
\lambda(u_{xx} + v_{yx} + w_{zx}) + 2\mu u_{xx} + \mu(u_{yy} + v_{xy}) + \mu(w_{xz} + u_{zz}) + F_x &= 0 \\
\lambda(u_{xx} + v_{yx} + w_{zx}) + \mu u_{xx} + \mu v_{xy} + \mu w_{xz} + \mu(u_{xx} + u_{yy} + u_{zz}) + F_x &= 0 \\
\lambda(u_{xx} + v_{yx} + w_{zx}) + \mu(u_{xx} + v_{xy} + w_{xz}) + \mu\nabla^2 u + F_x &= 0 \\
(\lambda + \mu)(u_{xx} + v_{yx} + w_{zx}) + \mu\nabla^2 u + F_x &= 0 \\
(\lambda + \mu)\nabla_x(u_x + v_y + w_z) + \mu\nabla^2 u + F_x &= 0 \\
(\lambda + \mu)\nabla_x(\nabla \cdot \underline{U}) + \mu\nabla^2 u + F_x &= 0
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial x}\tau_{yx} + \frac{\partial}{\partial y}\sigma_y + \frac{\partial}{\partial z}\tau_{yz} + F_y &= 0 \\
\mu(u_{yx} + v_{xx}) + \lambda(u_{xy} + v_{yy} + w_{zy}) + 2\mu v_{yy} + \mu(v_{zz} + w_{yz}) + F_y &= 0 \\
\lambda(u_{xy} + v_{yy} + w_{zy}) + \mu u_{yx} + \mu v_{yy} + \mu w_{yz} + \mu(v_{xx} + v_{yy} + v_{zz}) + F_y &= 0 \\
\lambda(u_{xy} + v_{yy} + w_{zy}) + \mu(u_{yx} + v_{yy} + w_{yz}) + \mu\nabla^2 v + F_y &= 0 \\
(\lambda + \mu)(u_{xy} + v_{yy} + w_{zy}) + \mu\nabla^2 v + F_y &= 0 \\
(\lambda + \mu)\nabla_y(u_x + v_y + w_z) + \mu\nabla^2 v + F_y &= 0 \\
(\lambda + \mu)\nabla_y(\nabla \cdot \underline{U}) + \mu\nabla^2 v + F_y &= 0
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial x}\tau_{zx} + \frac{\partial}{\partial y}\tau_{zy} + \frac{\partial}{\partial z}\sigma_z + F_z &= 0 \\
\mu(w_{xx} + u_{zx}) + \mu(v_{zy} + w_{yy}) + \lambda(u_{xz} + v_{yz} + w_{zz}) + 2\mu w_{zz} + F_z &= 0 \\
\lambda(u_{xz} + v_{yz} + w_{zz}) + \mu u_{zx} + \mu v_{zy} + \mu w_{zz} + \mu(w_{xx} + w_{yy} + w_{zz}) + F_z &= 0 \\
\lambda(u_{xz} + v_{yz} + w_{zz}) + \mu(u_{zx} + v_{zy} + w_{zz}) + \mu\nabla^2 w + F_z &= 0 \\
(\lambda + \mu)(u_{xz} + v_{yz} + w_{zz}) + \mu\nabla^2 w + F_z &= 0 \\
(\lambda + \mu)\nabla_z(u_x + v_y + w_z) + \mu\nabla^2 w + F_z &= 0 \\
(\lambda + \mu)\nabla_z(\nabla \cdot \underline{U}) + \mu\nabla^2 w + F_z &= 0
\end{aligned}
$$

En resumen:

$$
\begin{aligned}
(\lambda + \mu)\nabla_x(\nabla \cdot \underline{U}) + \mu\nabla^2 u + F_x &= 0 \\
(\lambda + \mu)\nabla_y(\nabla \cdot \underline{U}) + \mu\nabla^2 v + F_y &= 0 \\
(\lambda + \mu)\nabla_z(\nabla \cdot \underline{U}) + \mu\nabla^2 w + F_z &= 0
\end{aligned}
$$

En forma vectorial:

$$
\mu\nabla^2\underline{U} + (\lambda + \mu)\nabla(\nabla \cdot \underline{U}) + \underline{F} = \underline{0}
$$

q.e.d.

APÉNDICE C

Métodos de integración numérica para la simulación de objetos deformables

Con el objetivo de calcular el comportamiento dinámico de un objeto deformable, se hacen presentes las ecuaciones del movimiento de Newton que se aplican a cada uno de los puntos que conforman el objeto (*Teschner, 2004 [78]*). Suponiendo un modelo genérico en el cual se conocen las posiciones $\underline{x}_i^0$ y velocidades iniciales $\underline{v}_i^0$ de todos los puntos, un incremento en el tiempo h y las fuerzas resultantes $\underline{F}_i$ sobre los puntos de masa m_i, resultado de considerar parámetros de elasticidad, fuerzas internas, externas y de frontera, podemos calcular su aceleración lineal, velocidad lineal y posición en un tiempo t de la siguiente forma:

$$\underline{a}_i(t) = \frac{1}{m_i}\underline{F}_i \qquad \underline{v}_i(t) = \int \underline{a}_i(t)dt \qquad \underline{x}_i(t) = \int \underline{v}_i(t)dt$$

(Ecuaciones (2.4), (2.5) y (2.6)). Mismas que tienen su representación inversa como diferencial de la siguiente forma:

$$\underline{v}_i(t) = \frac{d\underline{x}_i}{dt} \qquad \underline{a}_i(t) = \frac{d\underline{v}_i}{dt}$$

.

De manera general y para fines computacionales, la forma de resolver las integrales es expresando el modelo en forma diferencial en una dimensión (para después aplicar el mismo método en cada una de las componentes para espacios de más de una dimensión), como una función del tiempo y el valor x, de la figuiente forma:

$$f(x,t) = \frac{dx}{dt}$$

Así, conocido un valor inicial $x = x_0$ en un instante inicial $t = t_0$, la solución al valor de x en cualquier instante t estará dado por:

$$x(t) = \int_{t_0}^{t} f(x,t)dt$$

.

Los método numéricos más usados en la resolución de este tipo de planteamientos son: el *Método de Integración del Euler*, el *Método de Integración de Runge-Kutta* y el *Método de Integración de Verlet*.

Sección C.1

Método de Integración de Euler

El método más simple para resolver la integral es el llamado *Método de Euler* [86][20]. El método consiste en ir aproximando el valor de la pendiente $\frac{dx}{dt}$ en el siguiente instante, t_1:

$$t_1 = t_0 + h$$

donde h es la longitud del paso, suficientemente pequeño para alcanzar la convergencia. La aproximación está dada por:

$$x_1 \approx x_0 + h \cdot tan(\psi) = x_0 + h\left(\frac{dx}{dt}\right)_{t=t_0} = x_0 + hf(x_0, t_0)$$

y de manera general:

$$t_{n+1} = t_n + h$$

$$x_{n+1} = x_n + hf(x_n, t_n)$$

Una método mejorado es el llamado *Método de Euler Predictor-Corrector (EPC)*, en el cual se calcula un promedio entre dos valores de la pendiente en dos puntos extremos en un intervalo h. Esto es, una aproximación por series de Taylor, trancada a dos términos por la izquierda, de x_1 está dada por:

$$x_1 \approx x_0 + h\left(\frac{dx}{dt}\right)_0 = x_0 + hf(x_0, t_0)$$

De igual manera, una aproximación por la derecha de x_0 es:

$$x_0 \approx x_1 - h\left(\frac{dx}{dt}\right)_1 = x_1 - hf(x_1, t_1)$$

De estas últimas dos ecuaciones, la pendiente de la curva está dada por:

$$\frac{x_1 - x_0}{h} = \frac{1}{2}\left(f(x_0, t_0) + f(x_1, t_1)\right)$$

Dado que x_1 aparece en ambos lados de la ecuación, la única forma de aproximar su valor es por medio del enfoque simple de Euler. Así, una forma simple del método es:

$$X_{n+1} = x_n + hf(x_n, t_n)$$

$$t_{n+1} = t_n + h$$

seguido de un estimado corregido:

$$x_{n+1} = x_n + \frac{1}{2}h\left[f(x_n, t_n) + f(X_{n+1}, t_{n+1})\right]$$

El análisis del error basado en series de Taylor muestra [86] que el método simple de Euler tiene un orden de error de $O(h)$, mientras que en el método EPC de $O(h^2)$, lo cual hace a este último un método de integración más exacto.

Sección C.2

Método de Integración de Runge-Kutta

Este método requiere de n funciones de evaluación por cada paso, y es uno de los métodos más exactos para el cálculo numérico de integrales [86][20]. Para una resolución de orden 4, se requieren de las siguientes funciones de evaluación:

$$\begin{aligned}
\delta_1 &= hf(x_0, t_0) \\
\delta_2 &= hf(x_0 + \tfrac{1}{2}\delta_1, t_0 + \tfrac{1}{2}h) \\
\delta_3 &= hf(x_0 + \tfrac{1}{2}\delta_2, t_0 + \tfrac{1}{2}h) \\
\delta_4 &= hf(x_0 + \delta_3, t_0 + h)
\end{aligned}$$

finalmente:

$$x_1 = x_0 + \frac{1}{6}(\delta_1 + 2\delta_2 + 2\delta_3 + \delta_4)$$

El orden de error de este método es de $O(h^4)$, lo cual lo hace más exacto en comparación con los métodos de Euler.

Sección C.3

Método de Integración de Verlet

Este método tiene aplicación directa en el cálculo de deformaciones y se ha empleado principalmente para simulación de telas, sistemas de masas-resortes, y cuerpos rígidos [78] basados en física. El método de Verlet (*Verlet, 1967 [83]*) usa la posición y las fuerzas actuantes sobre un punto en el tiempo t, así como las posiciones anteriores a dicho instante $t - h$ para calcular las nuevas posiciones de acuerdo a las expresiones:

$$x(t+h) = 2x(t) - x(t-h) + h^2 \frac{F(t)}{m} + O(h^4)$$

$$v(t+h) = \frac{x(t+h) - x(t-h)}{2h} + O(h^2)$$

Este método tiene algunas ventajas, considerando la interacción del objeto deformable con otros objetos que colisionan. En primer lugar, sólo se necesita calcular una fuerza por cada paso de integración, la segunda es el orden de error, que es de $O(h^4)$, en comparación con intervalos de h grandes y que iguala al método de Runge-Kutta; y finalmente la tercera es que no depende de la integración de las velocidades.

Agradecimientos

En la vida y en el quehacer académico, se me han presentado una gran cantidad de personas e instituciones, sin cuyo apoyo de todo tipo no habría sido posible la culminación de este trabajo. Para mis profesores, familiares y amigos, no puedo terminar sin ofrecerles mi más humilde retribución a todo ese apoyo que en algún momento me han brindado.

A mis padres, *Julia Vite* y *Emiliano Teodoro*, de quienes día a día tomo la fuerza necesaria para seguir adelante, en contra de tantas situaciones que ofrece la vida. Es a ellos a quien me debo física y profesionalmente; y a quienes, sin duda, ofrezco mi respeto y admiración.

A mis hermanos, quienes me han apoyado con sus sabios e insabios consejos, *Eliza*, *Marisol* y *Luis*. Especialmente a *Mary*, de quien en estos meses he aprendido a admirar el derecho a la vida y a aferrarse a ésta sin importar cuán difícil sea. A *Perla Juliette*, de quien sin esperarlo, trajo en mí la alegría y el enojo que sólo se puede lograr con la inocencia de un niño.

Al *Dr. Fernando Arámbula Cosío*, por su apoyo profesional, académico y humano brindado; ya que gracias a su buen juicio, calidez humana y paciencia durante el desarrollo de todo este trabajo, me permitieron resolver de manera clara y concisa todo objetivo buscado.

Al *Consejo Nacional de Ciencia y Tecnología* (CONACYT), por el apoyo económico que me brindó durante dos años para el desarrollo de mis actividades. Al *Centro de Ciencias Aplicadas y Desarrollo Tecnológico* (CCADET), por facilitarme el uso de sus instalaciones en la realización de pruebas. En especial al Grupo de Imágenes y Visualización, por permitirme encontrar en ellos, excelentes compañeros de trabajo y de la vida. Al *Posgrado en Ciencia e Ingeniería de la Computación*, por poseer a gente tan valiosa, que se entrega cada día a laborar con el mejor entusiasmo y amabilidad. A *Amalia*, *Diana*, *Cecilia* y *Lulú*.

Finalmente, a mi *alma mater*, la *Universidad Nacional Autónoma de México* (UNAM), a quien con orgullo admiro y respeto, como lo que es y seguirá siendo, la máxima casa de estudios de México: libre, científica y humana.

Referencias

[1] Aguilar P., Arellano J., 2010. *Matemáticas Aplicadas a las Ciencias de la Tierra*. Facultad de Ingeniería, UNAM. Primera Edición. México. 556 pp.

[2] Allard J., Cotin S., Faure F., Bensoussan P., Poyer F., Duriez C., Delingette H., Grisoni L., 2007. *SOFA–an open source framework for medical simulation.* Studies In Health Technology And Informatics. Volume 125, Publisher Citeseer, pp. 13-18.

[3] Allen M., Herrera I., Pinder G. F., 1988. *Numerical Modeling in Science and Engineering*. Wiley-Interscience, 99th Edition, 418 pp.

[4] Arámbula Cosío Fernando, 2008. *Automatic initialization of an active shape model of the prostate*. Medical Image Analysis 12, pp. 469-483.

[5] Archundia Abel, 2008. *Cirugía 1: Educación Quirúrgica*. Editorial McGraw-Hill. Tercera Edición. México.

[6] Azcárraga G., 1997. *UROLOGÍA*, Méndez Editores, 7a. Edición, México. pp. 9-28:256-264.

[7] Basdogan C., Chih-Hao, Srinivasan, 2001. *Virtual Environments for Medical Training: Graphical and Haptic Simulation of Laparoscopic Common Bile Duct Exploration*. IEEE ASME Transactions on Mechatronics. pp. 269–285.

[8] Basdogan C., Chih-Hao, Srinivasan M. A., 1999. *Simulation of tissue cutting and bleeding for laparoscopic surgery using auxiliary surfaces*. in Proc. Medicine Meets Virtual Reality 1999. pp. 38-44.

[9] Bielser D., Glardon P., Teschner M., Gross M., 2004. *A state machine for real-time cutting of tetrahedral meshes*. Elseiver. Graphical Models 66, pp. 398–417.

[10] *Blender*. http://www.blender.org/.

[11] Bro-Nielsen M., Helfrick D., Glass B., Zeng X., Connacher H., 1998. *VR simulation of abdominal trauma surgery*. Studies in health technology and informatics 02. pp. 117–23.

[12] Bussiere Ronald, 1997. *Principles of Electrosurgery*. Tektran Incorporated, USA.

[13] Caballero Arquímides. *Tablas matemáticas y formulario geométrico*. Grupo Editorial San Lorenzo. México, 10–15 pp.

[14] Choi Kup-Sze, 2006. *Interactive cutting of deformable objects using force propagation approach and digital design analogy*, Computers & Graphics 30. pp. 233–243.

[15] Coquillart S., 1990. *Extended free-form deformation: a sculpturing tool for 3D geometric modeling*. Computer and Graphics 24 (4), pp. 187–196.

[16] Coquillart S., 1991. *Animated free-form deformation: an interactive animation technique*. Computer and Graphics 25 (4), pp. 23–26.

[17] Cootes T. F., Taylor C. J., Cooper D. H., Graham J., 1995. *Active shape models-their training and application*. Computer Vision and Image Understanding. Vol. 61 (1), January, pp. 38–59.

[18] Cowin S. C., Doty S. B., 2007. *Tissue Mechanics*. Springer. USA. pp. 7–9, 43–85, 95–117.

[19] CUDA Technology. http://www.nvidia.com/object/cuda_home_new.html

[20] Engeln-Müllges G., Uhlig F., 1996. *Numerical Algorithms with C*. Springer, Primera Edición en inglés. 596 pp.

[21] Ferraina P., Oría A., 2002. *Cirugía de Michans*. Quinta edición, Editorial el Ateneo. 1092 pp.

[22] Flores Chávez Félix, 2011. *Implementación de partículas físicas en un ambiente gráfico de cirugía de próstata*. Tesis de Licenciatura. Universidad Nacional Autónoma de México. 63 pp.

[23] Frisoli A., Borelli L., Bergamasco M.. *Modeling biologic soft tissues for haptic feedback with an hybrid multiresolution method.* PERCRO Scuola Superiore Santa Anna, Pisa.

[24] García Guevara Gabriela, 2009. *Síntesis de texturas procedurales para un simulador computarizado de cirugía de próstata*. Tesis de Licenciatura. Universidad Nacional Autónoma de México. 146 pp.

[25] Georgii J., Westermann R., 2006. *A multigrid framework for real-time simulation of deformable bodies.* Computers & Graphics 30, 408–415.

[26] Gibson S. F. F., 1997. *3D ChainMail: a fast algorithm for deforming volumetric objects*. Proceedings of the Symposium on Interactive 3D Graphics, ACM SIGGRAPH, pp. 149–154.

[27] GLUT - *The OpenGL Utility Toolkit*. http://www.opengl.org/resources/libraries/glut/

[28] González R., Woods R., 2002. *Digital Image Processing*, Segunda Edición, Prentice Hall - Pearson Education International, USA. pp. 66.

[29] Herrera Ismael, 2010. *Modelación matemática de sistemas terrestres*. Instituto de Geofísica, UNAM, México, 81 pp.

[30] Hess Roland, 2011. *Blender*. Ediciones Anaya Multimedia. España. pp. 19–26.

[31] Hsu W. H., 1992. *Direct Manipulation of free-form deformations*, Computer and Graphics, 26 (2), pp. 177–184.

[32] Havok. http://www.havok.com/

[33] Irving G., Teran J., Fedkiw R., 2006. *Tetrahedral and hexahedral invertible finite elements*. Elseiver. Graphical Models 68. pp. 66–89.

[34] Iwanowski M., Serra J., 1999. *Morphological Interpolation and Color Images*, 10th International Conference on Image Analysis and Processing (ICIAP'99), pp. 50–55.

[35] Johnston H., Whitehead A., 2009. *Distinguishing games, serious games, and training simulators on the basis of intent*. Proceedings of the 2009 Conference on Future Play on @ GDC Canada, pp. 9–10.

[36] jMonkey Engine. http://www.jmonkeyengine.com/

[37] Kaufmann P., Martin S., Botsch M., Gross M., 2009. *Flexible simulation of deformable models using discontinuous Galerkin FEM*. Graphical Models 71. pp. 153–167.

[38] Kuhnapfel U., Cakmak H. K., Maaβ H., 2000. *Endoscopic surgery training using virtual reality and deformable tissue simulation.* Computers & Graphics 24. pp. 671–682.

[39] Lippert Herbert, 2003. *Anatomía: Estructura y Morfología del Cuerpo Humano*, Marban Libros, 4a. Edición, España. pp. 336, 355–361, 422–429.

[40] Maaβ H., Kühnapfel U.. *Noninvasive Measurement of Elastic Properties of Living Tissue* Institut für Angewandte Informatik, Forschungszentrum Karlsruhe, Postfach 3640, D-76133 Karlsruhe, Germany.

[41] Maciel A., Halic T., Lu Z., Nedel L. P., De S., 2009. *Using the PhysX engine for Physics-based Virtual Surgery with Force Feedback*. Int J Med Robot. September 5(3). pp. 341-353.

[42] Martínez D. S., Valdés R., Aguilar J. R., 2001. *Cirugía: bases del conocimiento quirúrgico y apoyo en trauma*, 3a. Edición, McGraw-Hill Interamericana, 445 pp.

[43] Meier U., López O., Monserrat C., M. C. Juan, Alcañiz M., 2005. *Real-time deformable models for surgery simulation: a survey*. Computer Methods and Programs in Biomedicine 77. pp. 183–197.

[44] Montaño Zetina, 2007. *Imagenología y detectores en medicina*. Revista Cinvestav, Enero-Marzo. México. pp. 16–23.

[45] Montgomery K., Leroy H. B., Bruyns C., Wildermuth S., Hasser C., Ozenne S., Bailey D., 2001. *Surgical simulator for hysteroscopy: a case study of visualization in surgical training*.

[46] *Diccionario Mosby de Medicina, Enfermería y Ciencias de la Salud*. Ediciones Harcourt. España, 2000.

[47] Müller M., McMillan L., Dorsey J., Jagnow R., 2001. *Real-time simulation of deformation and fracture of stiff materials*. Proceedings of the Eurographic workshop on Computer animation and simulation 2001. pp. 113–124.

[48] Müller M., Teschner M., 2003. *Volumetric Meshes for Real-Time Medical Simulations*, in Proceedings of BVM 2003 (Bildverarbeitung für die Medizin), pp. 279–283.

[49] Müller M., Teschner M., Gross M. H., 2004. *Physically-Based Simulation of Objects Represented by Surface Meshes*, in Proc. Computer Graphics International, pp. 26-33.

[50] Müller M., Gross M., 2004. *Interactive Virtual Materials*. in Proceedings of Graphics Interface, London, Ontario, Canada, May 17-19, pp. 239–246.

[51] Müller M., Heidelberger B., Hennix M, Ratcliff J., 2007. *Position based dynamics*. Journal of Visual Communication and Image Representation. Volume 18 Issue 2, April. pp. 109–118.

[52] Narayanasamy V., Wong K. W., Fung C. C., Rai S., 2006. *Distinguishing Games and simulation games from simulators*. Computers in Entertainment. 4, 2 (April), pp. 9.

[53] Natsupakpong S., Cavusoglu M. C., 2010. *Determination of elasticity parameters in lumped element (mass-spring) models of deformable objects*. Graphical Models. Elsevier. 72, October. pp. 61–73.

[54] Nealen A., Mueller M., Keiser R., Boxerman E., Carlson M., 2006. *Physically Based Deformable Models in Computer Graphics*. Computer Graphics Forum, Vol. 25, issue 4, pp. 809-836.

[55] Newton Game Dynamics. http://www.newtondynamics.com/

[56] Nienhuys H. W., Van Der Stappen A.F., 2000. *Combining finite element deformation with cutting for surgery simulations*, in: Proc. Eurographics 2000, pp. 274–277.

[57] Nienhuys H. W., Van Der Stappen A.F., 2001. *A surgery simulation supporting cuts and finite element deformation*. in: Proceedings of the Medical Image Computing and Computer-Assisted Intervention (MICCAI), LNCS 2208, Springer, Berlin, pp. 145–152.

[58] NVIDIA Developers. http://developer.nvidia.com/

[59] O'Brien J., Bargteil A., Hodgins J., 2002. *Graphical modeling of ductile fracture*. ACM Trans. Graph. (SIGGRAPH Proc.) 21. pp. 291–294.

[60] *OpenGL*, The Open Graphics Library. http://www.opengl.org/.

[61] OpenTissue. http://www.opentissue.org/

[62] Padilla M. A., Arámbula F., 2004. *Deformable model of the prostate for TURP surgery simulation*. Computer and Graphics 28, pp. 767–777.

[63] Padilla M. A., Teodoro S., Lira E., Soriano D., Altamirano F., Arámbula F., 2009. *Virtual Reality Simulator of Transurethral Resection of the Prostate*. IEEE Pan American Health Care Exchanges, March 16-20, pp. 116–119.

[64] nVidia PhysX. http://www.geforce.com/Hardware/Technologies/physx/

[65] Qt - Cross-Platform application and UI framework. http://qt.nokia.com/

[66] Reyes Bartolomé, 2010. *Validación de un simulador para entrenamiento en cirugía de próstata*. Tesis de Maestría en Ingeniería Eléctrica e Instrumentación, UNAM. 80 pp.

[67] Sadd Martin H., 2009. *Elasticity: Theory, Applications and Numerics*. Elseiver Academic Press. Second Edition. USA.

[68] Santiago Arce J., 2010. *Simulación de sensación táctil en un simulador de cirugía de próstata*. Tesis de Licenciatura. Facultad de Ingeniería, UNAM. México. 64 pp.

[69] Serby D., Harders M., Szekely G., 2001. *A new approach to cutting into finite element models*. in: Proc. Fourth Int. Conf. on Medical Imaging 2001, pp. 425–433.

[70] Smith S. E., *What is a resectoscope?*. wiseWeek.com. Editado por: Kristen Osborne. Copyright 2003-2011 Conjecture Corporation. `http://www.wisegeek.com/what-is-a-resectoscope.htm`.

[71] SOFA. Simulation Open Framework Architecture. http://www.sofa-framework.org/

[72] Soriano Valdéz D., 2008. *Generación de modelos de mallas de tetraedros para simuladores quirúrgicos*. Tesis Licenciatura (Ingeniero en Computación)-UNAM, Facultad de Ingeniería. México. 53 pp.

[73] Steinemann D., Otaduy M. A., Gross M., 2006. *Fast Arbitrary Splitting of Deforming Objects*. Eurographics/ ACM SIGGRAPH Symposium on Computer Animation 2006.

[74] Steinemann D., Otaduy M. A., Gross M., 2009. *Splitting meshless deforming objects with explicit surface tracking*. Graphical Models 71. pp. 209–220.

[75] Teodoro Vite S., 2008. *Modelado de un ambiente virtual para un sistema de simulación de cirugía de próstata*. Tesis de Licenciatura. Universidad Nacional Autónoma de México. 115 pp.

[76] Teodoro S., Arámbula F., Padilla M. A., Santiago J. R., Reyes B., Flores F., 2009. *Avances de un sistema de Realidad Virtual para el entrenamiento en Resección Transuretral de la Próstata*, MEXCAS 2009, México.

[77] Terzopoulos D., 1987. *Elastically deformable models*, Computer and Graphics, 21 (4). pp. 205–214.

[78] Teschner, Matthias, Heidelberger, 2004. *A Versatile and Robust Model for Geometrically Complex Deformable Solids*. IEEE Computer Society, Proceedings of the Computer Graphics International USA, pp. 312–319.

[79] Teschner M., Heidelberger B., Müller M., Pomeranerts D., Gross M., 2003. *Optimized spatial hashing for collision detection of deformable objects*. Proc. Vision, Modeling, Visualization VMV 2003, pp. 47–54.

[80] TetGen: A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator. http://tetgen.berlios.de/

[81] Madnenat-Thalmann N., Cordier F., 2000. *Construction of a Human Topological model from Medical Data*. Information Technology in Biomedicine. Vol. 4, No. 2, June. pp. 137–143.

[82] Unreal Engine. http://www.unrealengine.com/

[83] Verlet L., 1967. *Computer Experiments on Classical Fluids. Ii. Equilibrium Correlation Functions*, Physical Review, Vol. 165. pp. 201–204.

[84] Visible Human Project®, National Library of Medicine - http://www.nlm.nih.gov/research/visible/visible_human.html.

[85] Wicke M., Steinemann D., Gross M., 2005. *Efficient Animation of Point-Sampled Thin Shells*. Proceedings of Eurographics '05, (Dublin, Ireland, Aug 29th - Sep 2nd. pp. 667–676.

[86] Woolfson M. M., Pert G. J., 1999. *An introduction to Computer Simulation*. Oxford University Press. UK.

[87] Yang Yi-Jun, Yong Jun-Hai, Sun Jia-Guang, 2005. *An algorithm for tetrahedral mesh generation based on conforming constrained Delaunay tetrahedralization*. Computers and Graphics 29. pp. 606-615.

[88] Yang Xinmai, Church Charles, 2006. *A Simple Viscoelastic Model for Soft Tissues in the Frequency Range 6-20 MHz*. IEEE transactions on ultrasonics, ferroelectrics, and frequency control, vol. 53, no. 8, August. pp. 1404–1411.

[89] Yang Dan, 1994. *Mesh Generation and Information Model for device simulation*. Tesis doctoral. Universidad de Stanford.

[90] Zudaire J. J., 2002. *Manual de Urología*, Ariel Ciencias Médicas, España. pp. 297–325.