UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

MODELO DE SIMULACIÓN DE RESECCIÓN DE TUMORES CEREBRALES CON RETROALIMENTACIÓN HÁPTICA Y REALIDAD VIRTUAL PARA ENTRENAMIENTO MÉDICO

T E S I S

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA

DANIEL VARGAS CASTRO

DIRECTOR DE TESIS:

DR. MIGUEL ÁNGEL PADILLA CASTAÑEDA

ICAT, UNAM

MÉXICO, CD. MX., JUNIO DE 2022

Agradecimientos

A mi futura esposa, Eunice, que mientras trabajé en este proyecto me apoyó en todo sentido: desde hacerme una taza de café en las noches de desvelo, hasta resolverme mis dudas metodológicas, de redacción y a veces hasta existenciales. Espero que la próxima vez que alguien lea esto, ya seamos esposos.

A mi mamá, Martha, que toda mi vida ha sido la mejor mamá que podría tener y que desde que era un niño me impulsó para superarme académica y personalmente.

Al resto de mi familia, que me apoyó económicamente para que pudiera continuar con mis estudios de licenciatura y posgrado.

A mi asesor, el Dr. Miguel Padilla, que tuvo la paciencia de explicarme los conceptos teóricos y prácticos para poder llevar a cabo este proyecto.

Al Dr. Sergio Teodoro, que pacientemente me ayudó a solucionar y mejorar varios aspectos técnicos de este proyecto.

Al Dr. Isaac Tello, que se tomó el tiempo para probar este proyecto y darme retroalimentación sobre él.

A la UNAM, que aunque no es perfecta, es una universidad increíble y me ha ayudado a crecer y desenvolverme en todos los sentidos.

Al CONACYT, que mediante su sistema de becas me apoyó para poder concluir mis estudios de posgrado, como ha sido con miles de estudiantes antes de mí y espero que siga siendo con miles de estudiantes después de mí.

A la Secretaría de Educación, Ciencia, Tecnología e Innovación de la Ciudad de México (SECTEI 219/2019) y al proyecto DGAPA PAPIIT T100920, de quienes se recibió apoyos económicos que permitieron la realización de este proyecto.

Y finalmente, a mi gatita, Casiopea, que no me ayudó en nada, pero es agradable tenerla cerca.

Resumen

La neurocirugía es una de las disciplinas médicas técnicamente más exigentes y que mayor responsabilidad conlleva. El paradigma actual de aprendizaje de cirugías es mediante residencias, donde los aprendices operan pacientes reales bajo la supervisión de cirujanos más experimentados. Este sistema tiene la ventaja de que los aprendices siempre trabajan sobre casos reales en un ambiente supervisado, pero carece de un sistema de evaluación objetivo (la calificación depende del criterio del experto), puede poner en riesgo la salud de los pacientes y dificulta la práctica recurrente.

Una alternativa de formación clínica y práctica es el uso de simuladores. En particular, los simuladores computarizados (donde los usuarios interactúan con escenarios virtuales creados en una computadora) han ganado popularidad por su alta fidelidad visual y táctil, además de que mitigan muchas de las desventajas del aprendizaje mediante residencias. Por otra parte, los simuladores híbridos (donde se agregan elementos físicos a un simulador computarizado) mantienen todas las ventajas de los computarizados y además permiten crear un ambiente clínico más realista.

En este trabajo, se presenta el desarrollo de un simulador híbrido que recrea el proceso de resección de un tumor cerebral utilizando un aspirador ultrasónico y una pinza bipolar como herramientas quirúrgicas. El simulador consiste en una estación de trabajo móvil con elementos físicos y un escenario de resección computarizado, donde el comportamiento mecánico de los tejidos blandos se basa en parámetros físicos reales. Los usuarios pueden interactuar con esta escena en tiempo real por medio de dispositivos de realidad virtual y de retroalimentación háptica, brindando una alta fidelidad visual y táctil.

Se reportan pruebas de rendimiento y evaluación general por parte de un neurocirujano experto en resección de tumores cerebrales. Con base en los resultados obtenidos, se concluye que la simulación propuesta podría servir como una herramienta de aprendizaje de esta cirugía y como base para hacer otros simuladores de operaciones similares. Sin embargo, aún es necesario que este simulador pase por un proceso de validación riguroso para probar que su uso mejora las habilidades de los usuarios en el quirófano.

Tabla de contenido

1 Introducción

Un tumor cerebral es un crecimiento de células anormales en alguna parte del cerebro, siendo el tipo más común el meningioma, que surge de las membranas que rodean al cerebro, llamadas meninges [1]. La forma más común de tratar un tumor cerebral es por medio de una neurocirugía donde éste se extrae a través de una craneotomía [2].

De acuerdo con [3] la neurocirugía es una de las disciplinas médicas más técnicamente exigentes y que mayor responsabilidad conlleva. El paradigma de aprendizaje actual para el neurocirujano (y el médico en general) es el de residencias, el cual se inventó a principios del siglo XX y se basa principalmente en la mentoría y la experiencia en escenarios reales [4]. Este sistema tiene la ventaja de que el residente está expuesto a casos reales y cuenta con la supervisión y mentoría de alguien con mayor experiencia, sin embargo, hay autores que consideran que es un sistema obsoleto, pues cualquier error puede poner severamente en riesgo la salud del paciente y se dificulta la repetición [5, 3, 6, 7], pues para entrenar cierta técnica, es necesario encontrar un paciente que requiera de dicha técnica.

En la actualidad hay una preocupación cada vez mayor por la seguridad del paciente y los costos de enseñar en un ambiente clínico real. Por ejemplo, la OMS estima que más de 138 millones de pacientes se ven perjudicados por errores médicos y 2.6 millones mueren cada año por ello [5]. [6] estima que cerca de la mitad de los errores médicos ocurren durante intervenciones quirúrgicas y que el 75 % de éstas eran evitables. [7] presenta cifras similares para algunos países de América, Europa y Oceanía, y habla sobre los costos y los días de hospitalización adicionales que son necesarios gracias a los errores médicos, aunque también argumenta que algunos de estos reportes han sido criticados por su metodología, pero independientemente de las cifras, todo esto muestra una necesidad de reducir estas aflicciones en el área médica.

Una de las formas que se ha propuesto para complementar el entrenamiento de los cirujanos y que ha mostrado efectos positivos en su aprendizaje es la simulación, ya sea por medios orgánicos como cadáveres humanos o animales anestesiados, o inorgánicos

como simuladores sintéticos o computarizados [8]. Estos métodos permiten a los aprendices repetir alguna operación las veces que sea necesario mientras son evaluados de manera subjetiva u objetiva [9], permitiéndoles adquirir las habilidades técnicas y motrices que necesitan, sin poner en riesgo la salud de los pacientes [4].

En las últimas décadas, los simuladores computarizados han ganado popularidad sobre los otros tipos por su alta fidelidad visual y táctil [8]. Estos simuladores suelen funcionar en computadoras de alta gama e incorporar dispositivos de realidad virtual para el despliegue de imágenes y de dispositivos hápticos para dar una sensación táctil. Esto tiende a elevar su costo de adquisición, pero permiten a los usuarios "ver" y "sentir" una escena virtual, como si realmente estuvieran en ella. Además, tienen la ventaja de que permiten una cantidad prácticamente ilimitada de repeticiones de alta fidelidad del procedimiento que simulan, a diferencia de los simuladores sintéticos o con cadáveres, que se desgastan con el tiempo y cuya fidelidad suele impactar directamente su costo [8].

En este trabajo se presenta y justifica el desarrollo de un simulador computarizado para la operación de resección de un meningioma. Este simulador hace uso de dispositivos hápticos, de realidad virtual, de modelos de simulación basados en la biomecánica de tejidos e incorpora algunos elementos físicos. Posteriormente, se argumenta su desempeño, robustez y realismo a partir de métricas de desempeño obtenidas de un neurocirujano con experiencia en esta cirugía, tras usar el simulador.

1.1 Antecedentes

Actualmente existe una gran variedad de simuladores comerciales para entrenamiento quirúrgico, tanto orgánicos como inorgánicos. Incluso existen empresas dedicadas exclusivamente a la venta de simuladores médicos.

En particular, también existe una gran variedad de simuladores computarizados que incorporan dispositivos hápticos y de realidad virtual, pero la gran mayoría de ellos se

enfocan en cirugías laparoscópicas y endoscópicas, pues suelen ser más fáciles de recrear en realidad virtual y requieren de menos sensaciones hápticas que las cirugías abiertas [8].

Dentro de los simuladores computarizados, existen sólo un puñado de simuladores enfocados en procedimientos neuroquirúrgicos, entre los que destacan NeuroVR [10], Immersive Touch [11] y BACSIM [12], del Instituto de Ciencias Aplicadas y Tecnología (ICAT) de la UNAM, aunque este último se enfoca principalmente en neurocirugía vascular microscópica, por el momento.

De estos simuladores enfocados a neurocirugía, NeuroVR (previamente llamado Neurotouch) es el único que se enfoca en la cirugía de resección de tumores cerebrales. Neurotouch se concibió en 2012 como un simulador de resección de tumores cerebrales con un sistema de estereovisión y dos dispositivos hápticos para interactuar con el ambiente virtual [10]. Se ha demostrado su validez de apariencia, contenido y constructo [13, 14].

1.2 Justificación

Los expertos consideran que la mejor forma de aprender neurocirugía es practicándola [3]. Sin embargo, se ha reportado que los simuladores computarizados también tienen un efecto positivo sobre el aprendizaje [15, 16, 4], pues permiten a los aprendices repetir alguna operación las veces que sea necesario mientras son evaluados de manera objetiva [9], mejorando su entendimiento de la anatomía y sus habilidades visoespaciales [16] así como sus habilidades técnicas y motrices [4], con lo cual, potencialmente, se podría llegar reducir todos los costos relacionados con errores médicos [7].

El desarrollo de un simulador computarizado para resección de tumores cerebrales podría ser de gran ayuda para el entrenamiento de neurocirujanos en México, pues de acuerdo con médicos e investigadores del Hospital General de México, del Instituto

Nacional de Neurología y Neurocirugía (INNN) y del Centro de Simulación Médica de la UNAM, no existen simuladores como este en el país y consideran que es poco probable que adquieran uno, pues uno comercial como NeuroVR tiene un costo aproximado de 185,000 dólares.

Un simulador de resección de tumores cerebrales desarrollado en México daría a los estudiantes de neurocirugía la posibilidad de practicar esta operación las veces que quisieran, pudiendo obtener métricas objetivas de su desempeño, sin poner en riesgo la salud de los pacientes. Además, este simulador quedaría abierto para mejoras, o bien, como antecedente para futuros simuladores que se desarrollen. Por ejemplo, podría integrarse como módulo para el sistema BACSIM del ICAT, UNAM.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar y evaluar un simulador de resección de tumores cerebrales utilizando dispositivos de retroalimentación háptica y de realidad virtual que sea útil para el entrenamiento de los neurocirujanos.

1.3.2 Objetivos específicos

- Desarrollar un modelo de simulación válido biomecánicamente.
- Incorporar dispositivos de retroalimentación háptica y de realidad virtual para hacer un simulador altamente inmersivo.
- Reconstruir modelos 3D a partir de imágenes de resonancia magnética de casos reales de personas con tumores cerebrales y adaptarlos al simulador.
- Hacer pruebas con neurocirujanos para obtener retroalimentación de ellos y datos de su interacción con el simulador.
- Evaluar el simulador con base en los datos adquiridos de los neurocirujanos.

1.4 Organización de la tesis

Esta tesis está organizada en seis capítulos, cada uno abordando los diferentes aspectos teóricos y prácticos del sistema propuesto y descripciones del software y hardware utilizado.

En el capítulo 1 se presenta la introducción a este trabajo, definiendo la problemática que se pretende resolver, las formas en que otros la han abordado, los objetivos de este proyecto y su relevancia.

En el capítulo 2 se presenta el marco teórico, describiendo brevemente cómo es una cirugía real de resección de tumores cerebrales y qué es un simulador. Posteriormente, se definen los conceptos más importantes que se requirieron para hacer el simulador, por ejemplo: la simulación de tejidos, el renderizado háptico, los cambios topológicos en las mallas y la realidad virtual. También se describen brevemente los dispositivos hápticos y de realidad virtual utilizados en el simulador, así como las características principales del software de simulación biomecánica SOFA, sobre el que está desarrollado este simulador.

En el capítulo 3 se explica cómo se aplicaron los conceptos vistos en el marco teórico para desarrollar el simulador en cuestión. Primero se habla sobre la creación de la escena de simulación en la computadora y la inclusión del hardware externo, luego de la obtención de las mallas 3D que representan los objetos de simulación, posteriormente de la construcción de la estación de trabajo, después de la generación de métricas de desempeño del usuario y finalmente de la GUI de control del simulador.

En el capítulo 4 se describen y justifican las pruebas realizadas para evaluar el simulador y los resultados obtenidos.

En el capítulo 5 se explican los resultados obtenidos en el capítulo 4, y con base en ellos, se discute sobre el desempeño del simulador y sobre su evaluación como herramienta de entrenamiento médico.

Finalmente, en el capítulo 6 se escriben las conclusiones obtenidas de la realización de este trabajo y los aspectos en los que aún hace falta trabajar.

2 Marco teórico

Un simulador híbrido es un desarrollo computacional complejo, pues requiere aplicar diversas tecnologías y algoritmos que permitan simular la operación o sistema deseado.

En este capítulo se describen brevemente todos los conceptos básicos, algoritmos y tecnologías que se utilizaron en el desarrollo de este simulador para posteriormente, en el capítulo 3, explicar cómo se aplicó todo lo descrito en este capítulo para crear un simulador que cumpla con los objetivos descritos en la sección 1.3.

2.1 Cirugía de resección de tumores cerebrales por craneotomía

Un tumor cerebral es una masa o un crecimiento de células anormales en el cerebro [1]. Existen muchos tipos de tumores cerebrales, los cuales difieren entre sí por el tipo de células donde se genera. El tipo más común de tumor cerebral es el meningioma, que surge de las membranas que rodean al cerebro, llamadas meninges [1].

La forma más común de tratar un tumor cerebral es por medio de una craneotomía [2]. La craneotomía consiste en hacer una incisión en el cuero cabelludo para exponer el cráneo, donde se hace una abertura para exponer la dura madre, la cual se corta para exponer el cerebro [17]. El neurocirujano procede a navegar por el cerebro expuesto a través de un microscopio con el fin localizar el tumor previamente visualizado en imágenes médicas. Posteriormente lo remueve utilizando diferentes técnicas y herramientas, entre las cuales está la pulverización y aspiración del tumor mediante un aspirador ultrasónico [2], como se aprecia en Fig. 1. Durante la cirugía, el neurocirujano debe remover la mayor cantidad posible del tumor, conservando la mayor cantidad de tejido cerebral sano y sin dañar las estructuras cercanas [17]. Una vez removida la mayor cantidad posible del tumor, se sutura la apertura en la dura madre, se coloca y ajusta la sección removida del cráneo y se sutura el cuero cabelludo.

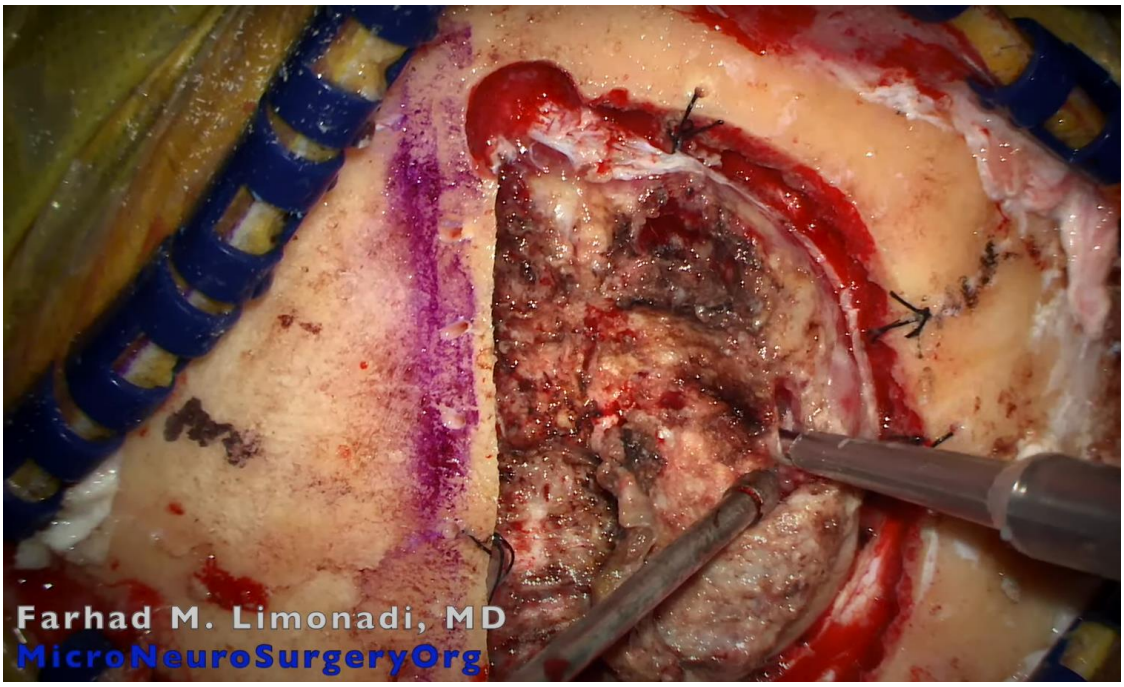

Fig. 1 Imagen de una cirugía de resección de tumores cerebrales vista a través del microscopio. Se aprecia cómo el aspirador ultrasónico (herramienta derecha) remueve fragmentos del tumor mientras se succionan secreciones con el aspirador quirúrgico (herramienta izquierda).
Fuente: captura de pantalla de [18].

2.2 Simuladores híbridos para entrenamiento quirúrgico

Los simuladores pretenden sumergir a los usuarios en un ambiente virtual muy parecido a la realidad, tratando de convencerlos de que los abordajes que tienen en la simulación son los mismos que tendrían en la realidad [8]. Esto se logra proporcionando la mayor cantidad de información multisensorial al usuario en la simulación mediante una combinación de tecnologías, como cascos o lentes para proyectar imágenes estereoscópicas, audífonos para reproducir sonidos, guantes o dispositivos hápticos para dar sensaciones táctiles e incluso dispositivos de despliegue de olores y sabores [7]. Por ejemplo, en Fig. 2 se muestra un simulador quirúrgico que involucra dispositivos hápticos y de realidad virtual, con lo cual se pueden retroalimentar los sentidos del tacto y la vista.

Fig. 2 Imagen del simulador híbrido BACSIM para clipaje de aneurismas.
Se aprecia que el usuario interactúa con el simulador por medio de dispositivos hápticos mientras observa la escena a través de un casco de realidad virtual.
Fuente: [12].

Gracias a los avances tecnológicos de las últimas décadas, los simuladores quirúrgicos computarizados que involucran dispositivos hápticos y de realidad virtual han ido adquiriendo popularidad y han mostrado ser útiles en el entrenamiento médico [8, 19, 3, 4, 15, 16]. Además, han empezado a surgir simuladores híbridos como BACSIM [12], donde se integran elementos físicos (por ejemplo, maniquíes) a los simuladores computarizados, permitiendo crear un ambiente clínico más realista [8] y ergonómico [12].

En la literatura se habla sobre las características que debería tener un simulador quirúrgico. Por ejemplo, [7] argumenta que éstos deberían contener elementos de cuatro categorías: primero, debe tener un respaldo médico, es decir, los médicos son quienes deben definir el problema, identificar sus necesidades de aprendizaje y evaluar la solución final; segundo, los modelos anatómicos deben ser realistas tanto visual como biomecánicamente; tercero, el simulador debe ser lo más inmersivo posible, enfocándose principalmente en el sentido de la vista y el tacto; y cuarto, se debe

permitir interactuar con los tejidos de la simulación, por ejemplo, deformándolos o cortándolos.

En [20] se exponen ideas sobre las características de un simulador quirúrgico computarizado, y en general concuerdan con las ideas de [7], pero además se argumenta que la simulación debe ser eficiente para poder desplegarse en tiempo real, robusta para no ser propensa a errores o artefactos y ser satisfactoria o realista.

Por otra parte, [21] habla específicamente sobre los simuladores de neurocirugía, pero en general sus ideas concuerdan con las de [7] y [20].

Como se puede intuir, la creación de un simulador quirúrgico propone un reto computacional importante, pues es necesario encontrar métodos y tecnologías para cumplir con estas expectativas, y muchos de ellos siguen siendo un tema de investigación abierto.

2.3 Simulación Biomecánica

De acuerdo con [22], la simulación es la imitación de la operación de un proceso o sistema del mundo real a través del tiempo, ya sea de forma manual o generada por computadora. El comportamiento del sistema a simular se estudia a través de un modelo, el cual se crea a partir de suposiciones sobre la operación real de dicho sistema, expresadas como relaciones matemáticas, lógicas o simbólicas.

Biomecánica es una palabra compuesta que implica la aplicación de la mecánica clásica a los sistemas biológicos, como lo es el cuerpo humano [23]. Entre sus aplicaciones está (pero no se limita a) el diseño de prótesis y de órganos artificiales, así como el análisis y simulación de tejidos rígidos como huesos o caparazones, o blandos, como músculos y órganos.

A partir de las dos definiciones anteriores, se intuye que una simulación biomecánica es la imitación del comportamiento de un sistema biológico, basado en las leyes de la mecánica que lo rigen.

2.3.1 Simulación de tejidos blandos en tiempo real

En el cuerpo humano, los tejidos blandos son aquellos que se deforman al interactuar con ellos mismos, con otros tejidos o, en el caso de la cirugía, con herramientas quirúrgicas [24].

Una preocupación común al hacer simuladores computarizados que involucran objetos deformables es que dicha simulación pueda ejecutarse en tiempo real. En este ámbito, "tiempo real" se entiende como una frecuencia de actualización de al menos 30 fotogramas por segundo para la animación [25]. Esto significa que, para simular la deformación de tejidos blandos en tiempo real, es necesario calcular sus deformaciones y redibujar el tejido deformado al menos una vez cada 33 milisegundos, lo cual se puede convertir en una tarea complicada que generalmente implica encontrar un equilibrio entre la velocidad del cómputo y la precisión de este [24].

2.3.2 Métodos numéricos

Hacer una simulación de las deformaciones de un tejido blando completamente precisa y realista en una computadora sería prácticamente imposible, pues sería necesario tener en cuenta todas las propiedades físicas del continuo. Sin embargo, se puede hacer una aproximación a la solución usando algoritmos iterativos sobre discretizaciones espaciales y temporales del objeto a simular [26]. A estos algoritmos se les conoce como métodos numéricos.

En esta sección se describen algunos de los métodos numéricos que se suelen usar para simular las deformaciones de los tejidos blandos.

2.3.2.1 Método de elementos finitos para sistemas dinámicos

El método de los elementos finitos (FEM por sus siglas en inglés) es el más popular para calcular deformaciones de objetos basándose en sus propiedades elásticas [27].

La idea básica del método de los elementos finitos es aproximar el dominio continuo mediante una malla con una cantidad finita elementos pequeños y con forma regular, a partir de cuyos nodos se puede interpolar la posición de cualquier punto en el continuo [26]. Esto se puede representar mediante la ecuación:

$$X = NU, \tag{1}$$

donde X es el vector de coordenadas de cualquier punto del objeto simulado, U es el vector de coordenadas de los nodos del elemento que contiene a X en la malla discretizada y N es la matriz o función de forma, a partir de la cual se interpola X.

Existen varias formas para discretizar espacialmente un objeto, pero una de las más comunes es por medio de tetraedros. A continuación, se resume el método de los elementos finitos para un sistema dinámico discretizado espacialmente mediante una malla de tetraedros. Este método se explica a detalle en [20].

Si el objeto se discretiza mediante una malla de tetraedros, la función de interpolación de forma para encontrar el desplazamiento $u(\boldsymbol{x})$ de cualquier punto $\boldsymbol{x}$ que se encuentra dentro de un tetraedro e es

$$u(\boldsymbol{x}) = \sum_{i=1}^{4} \frac{1}{6V^e} (a_i^e + b_i^e x + c_i^e y + d_i^e z) u_i^e, \tag{2}$$

donde x, y y z son las coordenadas del punto $\boldsymbol{x}$, u_i^e es el vector de coordenadas de un vértice del tetraedro y $6V^e$, a_i^e, b_i^e, c_i^e, d_i^e son

$$6V^e = \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{vmatrix},\ a_i^e = \begin{vmatrix} x_j & x_k & x_l \\ y_j & y_k & y_l \\ z_j & z_k & z_l \end{vmatrix},$$

$$b_i^e = -\begin{vmatrix} 1 & 1 & 1 \\ y_j & y_k & y_l \\ z_j & z_k & z_l \end{vmatrix},\ c_i^e = \begin{vmatrix} 1 & 1 & 1 \\ x_j & x_k & x_l \\ z_j & z_k & z_l \end{vmatrix},\ d_i^e = -\begin{vmatrix} 1 & 1 & 1 \\ x_j & x_k & x_l \\ y_j & y_k & y_l \end{vmatrix} \tag{3}$$

donde x, y y z son las coordenadas del elemento *e*. Los índices inicialmente tienen los valores $i = 1$, $j = 1$, $k = 1$ y $l = 1$ pero se irán permutando cíclicamente con cada iteración de (2).

Lo anterior resuelve el problema de encontrar la función de forma. Sin embargo, aún falta encontrar las deformaciones del objeto discretizado (en este caso, un tejido blando) a través del tiempo. Para ello, es necesario formular su comportamiento como un sistema dinámico de la forma

$$M\ddot{u} + D\dot{u} + Ku = F, \tag{4}$$

donde u es el vector desplazamientos de los nodos de la malla de tetraedros, $\dot{u}$ y $\ddot{u}$ son, respectivamente, su primera y segunda derivada con respecto al tiempo, M, D y K son, respectivamente, las matrices de masa, amortiguamiento y rigidez del objeto simulado y F es el vector de fuerzas externas aplicadas al objeto.

Para discretizar la masa y el amortiguamiento, se puede asumir que en cada nodo de la malla hay cierta cantidad de masa, por lo que se puede calcular *M* y *D* como

$$M = \frac{1}{3}\rho V \qquad D = \alpha M, \tag{5}$$

donde ρ es la densidad de masa del tejido, *V* es el volumen de la malla discretizada y α es un factor de escala.

Para obtener la matriz de rigidez global K del tejido, primero es necesario obtener la matriz de rigidez de cada elemento de la discretización K^e. Esta se obtiene aplicando a cada tetraedro de la discretización la ecuación

$$K^e = B^{eT} C B^e V^e, \tag{6}$$

donde B^e es:

$$B^e = \frac{1}{6V^e}\begin{bmatrix} b_1^e & 0 & 0 & b_2^e & 0 & 0 & b_3^e & 0 & 0 & b_4^e & 0 & 0 \\ 0 & c_1^e & 0 & 0 & c_2^e & 0 & 0 & c_3^e & 0 & 0 & c_4^e & 0 \\ 0 & 0 & d_1^e & 0 & 0 & d_2^e & 0 & 0 & d_3^e & 0 & 0 & d_4^e \\ c_1^e & b_1^e & 0 & c_2^e & b_2^e & 0 & c_3^e & b_3^e & 0 & c_4^e & b_4^e & 0 \\ 0 & d_1^e & c_1^e & 0 & d_2^e & c_2^e & 0 & d_3^e & c_3^e & 0 & d_4^e & c_4^e \\ d_1^e & 0 & b_1^e & d_2^e & 0 & b_2^e & d_3^e & 0 & b_3^e & d_4^e & 0 & b_4^e \end{bmatrix}, \tag{7}$$

tal que $6V^e$, b_i^e, c_i^e y d_i^e son los coeficientes vistos en (3). Por otra parte, si se considera que el material es isotrópico, homogéneo y con un comportamiento elástico lineal (suposiciones comunes en la simulación de tejidos blandos en tiempo real [20, 27, 26]), la matriz del material C es

$$C = \begin{bmatrix} \lambda + 2\mu & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2\mu & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2\mu & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix}, \tag{8}$$

donde λ y μ son las constantes de Lamé

$$\lambda = \frac{E}{2(1+v)}, \qquad \mu = \frac{Ev}{(1+v)(1-2v)}, \tag{9}$$

tal que E y v son el módulo de Young y el coeficiente de Poisson, respectivamente.

A partir de la función de forma vista en (2) y con la solución la ecuación diferencial vista en (4) para u, es posible obtener la deformación de cualquier punto del tejido a simular. Sin embargo, el FEM no considera un algoritmo particular para calcular la solución de (4), por lo que es necesario implementar métodos que permitan solucionar ecuaciones diferenciales ordinarias y sistemas de ecuaciones lineales grandes y dispersos.

2.3.2.2 Método de Euler explícito e implícito

El método de Euler es un método numérico comúnmente utilizado para resolver ecuaciones diferenciales ordinarias con condiciones iniciales dadas. Si bien no es el método más preciso, es ampliamente utilizado por su simplicidad [28].

La idea básica detrás del método es utilizar la ecuación diferencial para calcular la pendiente de la recta tangente a la curva que genera. Como se conoce el punto inicial de la curva, entonces ésta se puede aproximar calculando iterativamente una serie de puntos a lo largo de ella, utilizando la pendiente en cada uno de esos puntos.

Supongamos que se tiene una ecuación diferencial del tipo

$$\frac{dy}{dx} = f(x, y) \tag{10}$$

con condiciones iniciales $P_0 = (x_0\ , y_0)$. Para aproximar iterativamente el valor de x a lo largo de esta ecuación, se puede definir una variable arbitraria h, tal que $x_1 = x_0 + h, x_2 = x_1 + h, x_3 = x_2 + h$, y así sucesivamente, es decir,

$$x_{n+1} = x_n + h. \tag{11}$$

Conociendo los valores iniciales x_0 y y_0, se puede aproximar y_1 calculando $y_0 + hf(x_0, y_0)$ luego, se puede calcular aproximar y_2 calculando $y_1 + hf(x_1, y_1)$ y así sucesivamente, es decir,

$$y_{n+1} = y_n + hf(x_n, y_n). \tag{12}$$

Tras aplicar esta ecuación sucesivamente n veces, se habrá aproximado la solución de (10) para esos n puntos. Este es el método de Euler explícito.

Sin embargo, en los métodos implícitos, para calcular el siguiente valor de y, es necesario haberlo aproximado previamente. Por ejemplo, el método de Euler implícito utiliza la ecuación

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}), \tag{13}$$

donde el término y_{n+1} del lado derecho de la ecuación se puede aproximar utilizando métodos directos o iterativos.

En simulación quirúrgica se suele utilizar el método de Euler implícito para resolver (4), pues tiene la ventaja de ser más estable y soportar pasos de tiempo más grandes que el explícito, aunque tarda un poco más de tiempo en calcular la aproximación [20].

Cabe notar que, en simulación en tiempo real, se trata de resolver (4), por lo que, como se explica en [20], (13) se convierte en

$$\frac{M}{h^2}(u_{n+1} - 2u_n + u_{n-1}) + \frac{D}{2h}(u_{n+1} - u_{n-1}) + Ku_{n+1} = F_n, \tag{14}$$

que se puede resolver con métodos de solución sistemas de ecuaciones lineales.

2.3.2.3 Método del gradiente conjugado

El método del gradiente conjugado (CG, por sus siglas en inglés) es un método iterativo que se utiliza para aproximar la solución a sistemas de ecuaciones lineales que podrían ser demasiado grandes y/o dispersos para resolverse por métodos directos. A continuación, se da una breve explicación de su funcionamiento con base en [29].

Sea

$$Ax = b, \tag{15}$$

donde A es una matriz definida positiva, simétrica y dispersa de tamaño $n \times n$ y $b \in \mathbb{R}^n$, el método del gradiente conjugado trata de aproximar el valor del vector x, tratando de encontrar el valor mínimo de la forma cuádrica

$$f(x) = \frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle. \tag{16}$$

Dado un valor inicial de x, se puede obtener una solución aproximada x^* "bajando" por la forma cuadrática en la dirección hacia donde ésta decrementa más rápidamente, es decir, en la dirección de su gradiente negativo

$$-\nabla f(x) = -\nabla\left(\frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle\right), \tag{17}$$

que como se demuestra en [29], se puede simplificar como

$$-\nabla f(x) = b - Ax. \tag{18}$$

Como $-\nabla f(x)$ es la diferencia entre la solución real b y la aproximada Ax (descritos en la ecuación (15)), entonces también indica qué tan "lejos" se está de dicha solución real. A este valor se le conoce como residuo y para cada posible solución x_i, se define como

$$r_i = b - Ax_i. \tag{19}$$

La versión más básica algoritmo empieza con una aproximación inicial $x^{(0)}$, y a partir de ahí, "baja" en la dirección d del primer residuo, de modo que $d^{(1)} = r^{(0)} = b - Ax^{(0)}$.

Posteriormente, cada aproximación subsecuente a la solución se calcula como

$$x^{(i)} = x^{(i-1)} + s_i d^{(i)}, \tag{20}$$

donde s_i es un escalar que indica "qué tanto" se va a "bajar" en la dirección de $d^{(i)}$ y su valor se actualiza mediante la ecuación

$$s_k = \frac{\langle r^{(i-1)}, r^{(i-1)} \rangle}{\langle d^{(i)}, Ad^{(i)} \rangle}, \tag{21}$$

mientras que $d^{(i)}$ se actualiza mediante la ecuación

$$d^{(i)} = r^{(i-1)} + c_i d^{(i-1)}, \tag{22}$$

donde c_i es un escalar que optimiza la búsqueda de $d^{(i)}$ y su valor se actualiza mediante la ecuación

$$c_k = \frac{\langle r^{(i)}, r^{(i)} \rangle}{\langle r^{(i-1)}, r^{(i-1)} \rangle}. \tag{23}$$

Las ecuaciones (21), (22) y (23) requieren de un valor actualizado del residuo r. Éste se obtiene de la ecuación

$$r^{(i)} = r^{(i-1)} - s_k A d^{(i)}. \tag{24}$$

El algoritmo termina y regresa x^* cuando $r^{(i)}$ es menor o igual a una tolerancia dada o cuando se ha llegado a un número de iteraciones máximo. Generalmente, el usuario puede definir el valor de estas condiciones de paro.

Muchos de los métodos directos de solución de sistemas de ecuaciones lineales tienen una complejidad de tiempo de hasta $O(n^3)$. Por otra parte, de acuerdo con [29] el método CG tiene una complejidad de tiempo de $O(m\sqrt{k})$, donde m es la cantidad de elementos de la matriz A que no son 0 y

$$k = \max_{\|x\|=1} \|Ax\| \max_{\|x\|=1} \|A^{-1}x\|, \tag{25}$$

En esta sección se discutió la versión más básica del método del gradiente conjugado, pero [29] y [30] proponen métodos para mejorar el rendimiento del algoritmo. Por ejemplo, es posible manipular algebraicamente la matriz A de modo que k se reduzca a valores cercanos a 1, con lo cual se logra que la complejidad del algoritmo sea casi $O(m)$.

2.3.3 Simulación basada en restricciones

Además de las deformaciones de tejido, otro aspecto importante en la simulación quirúrgica es la detección y respuesta correcta de colisiones entre modelos.

Algunas colisiones entre modelos pueden ser demasiado complejas para simular en tiempo real utilizando métodos directos. Sin embargo, [31] define una forma de resolver este problema mediante multiplicadores de Lagrange, el cual se describe brevemente en esta sección.

Los autores plantean un sistema dinámico compuesto por modelos rígidos o deformables, descrito por la ecuación:

$$Ma = f(t, x, v) + H^T\lambda, \tag{26}$$

donde a representa la aceleración, t el tiempo, x la posición, v la velocidad, f el vector de fuerza, M la matriz de masa, H^T la dirección de las restricciones del modelo de simulación y el vector λ (los multiplicadores de Lagrange) la fuerza asociada a cada restricción.

Por cada restricción, se asigna una ley que depende de la posición relativa de los objetos que interactúan y otros parámetros como la fuerza máxima y el coeficiente de fricción:

$$\Phi(x_1, x_2, \ldots, x_n) = 0, \tag{27}$$

$$\psi(x_1, x_2, \ldots, x_n) \geq 0, \tag{28}$$

donde Φ representa leyes de interacción bilaterales (como acoplamientos o articulaciones), ψ representa leyes de interacción unilaterales (como colisiones o fricción) y x_i es alguno de los objetos o parámetros en la interacción.

Para resolver estas restricciones, primero se realiza un paso de predicción, donde se soluciona (26) por cada objeto que está en interacción, asumiendo que no hay restricciones ($\lambda = 0$), con lo que se obtiene una predicción del movimiento. Luego, se linealizan las leyes de restricción (27) y (28), y se resuelve el sistema para obtener el valor de λ. Finalmente, con el valor λ ya disponible, se calcula una corrección del movimiento que, sumado con la predicción de movimiento calculada en el primer paso, da como resultado el movimiento final.

2.4 Simulación de remoción de tejido

En graficación por computadora, una malla es una colección de vértices, aristas y caras que definen la forma de un poliedro. La geometría de una malla describe la ubicación de sus vértices en el espacio, mientras que la topología define cómo se conectan los vértices entre sí para generar una figura. Se requiere tanto de la geometría como de la topología para poder visualizar, modelar mecánicamente, detectar colisiones y hacer renderizado háptico sobre una malla [25].

En los simuladores quirúrgicos suele ser necesario hacer cambios topológicos en tiempo real sobre las mallas de los modelos involucrados, por ejemplo, para hacer un corte o sutura en un tejido. Esto presenta un reto, pues es necesario que el

comportamiento visual, mecánico, de colisión y háptico del objeto sea válido sin importar los cambios topológicos que se hagan en la malla que los representa [32].

Una forma eficiente de simular el corte o remoción de tejido es representando dicho tejido mediante una malla volumétrica con tetraedros y eliminando los tetraedros de la zona donde se desea cortar [33]. En las siguientes secciones se describen métodos eficientes para representar una malla volumétrica y remover tetraedros de ella.

2.4.1 Representación versátil de mallas volumétricas

Tener una representación de una malla volumétrica que soporte cambios topológicos es una tarea complicada, pero esencial para un simulador de cirugía. Afortunadamente existe un *framework* versátil y eficiente diseñado por [32] específicamente con este fin, el cual está implementado en SOFA [25] y se explicará en esta sección.

Este *framework* se basa en el hecho de que una malla volumétrica puede estar conformada por tetraedros o hexaedros, los cuales se pueden subdividir en triángulos o cuadriláteros (respectivamente), que a su vez se pueden subdividir en aristas que a su vez se pueden subdividir en vértices. Con esto en mente, los autores proponen una estructura jerárquica de "objetos topológicos", donde los objetos hijo están hechos a partir de sus objetos padre, como se muestra en Fig. 3.

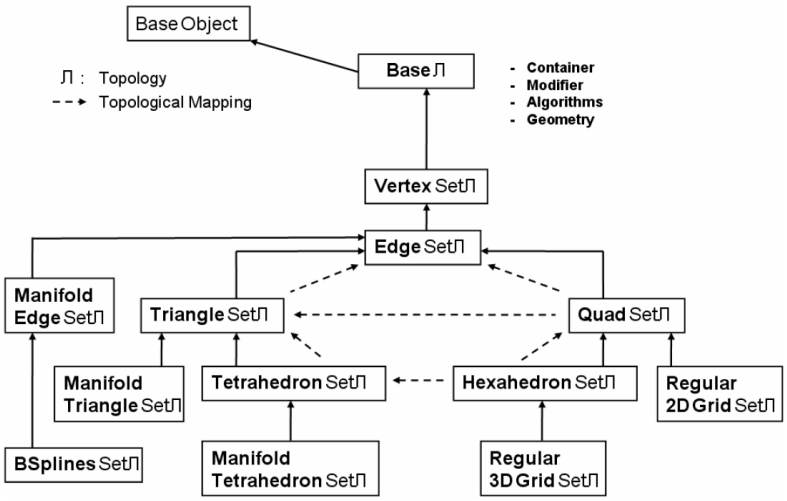

Fig. 3 Árbol de "objetos topológicos". Las flechas continuas representan un mapeo directo entre un objeto topológico y otro, mientras que las punteadas representan un posible mapeo topológico. Fuente: [32].

Cada objeto topológico está compuesto por cuatro miembros: *Container* que describe cómo se mapean los objetos topológicos entre sí; *Modifier* que provee de métodos de bajo nivel para hacer cambios topológicos (como el que se describe en la sección 2.4.2); *Algorithms* que convierte cambios topológicos de alto nivel en métodos de bajo nivel y *Geometry* que contiene información de la geometría de la malla que no necesariamente se guarda en la malla como tal (por ejemplo, longitudes o curvaturas). Por otra parte, se tienen los componentes de SOFA que almacenan la información biomecánica del modelo en cuestión y se actualizan conforme a los cambios descritos por los objetos topológicos.

Cuando se desea hacer cambios topológicos a una malla, se notifica al miembro *Algorithms*, quien consulta la información de *Geometry* para comunicar a *Modifier* los cambios de bajo nivel a realizar, quien finalmente los ejecuta con base en la información de *Container*. Los componentes de SOFA se pueden actualizar antes o después de este proceso, dependiendo del tipo de evento topológico.

Los autores hicieron varias pruebas con su *framework* sobre una computadora con un Pentium-M a 2.13 GHz y 1 GB de RAM. Para el caso de remoción de tetraedros usaron una malla de un corazón con 16,553 tetraedros y otra con 97,087 tetraedros. En ambos casos, utilizaron componentes de SOFA para simular el comportamiento del tejido mediante un modelo de elementos finitos corrotacional. Con esta configuración, lograron remover una sección de 557 tetraedros del modelo de 16,553 en 196 ms y una sección de 2,455 tetraedros del modelo de 97,087 en 883 ms. Esto implica un tiempo de remoción de ~0.35 ms por cada tetraedro, para esta configuración.

2.4.2 Remoción de tetraedros en una malla volumétrica tipo manifold

En esta sección se explica el método para remover tetraedros de una malla volumétrica tipo *manifold* descrito por [34], el cual conserva esta propiedad tras la remoción y está enfocado a la simulación quirúrgica en tiempo real.

De manera general, una malla de tetraedros *manifold* es aquella que es *conformal* (todos los tetraedros que la conforman tienen un volumen > 0 y no se intersecan entre sí) y además no tiene singularidades de vértices o aristas en la superficie, como las que se muestran en Fig. 4. El algoritmo descrito a continuación supone que la malla de entrada cumple con estas propiedades.

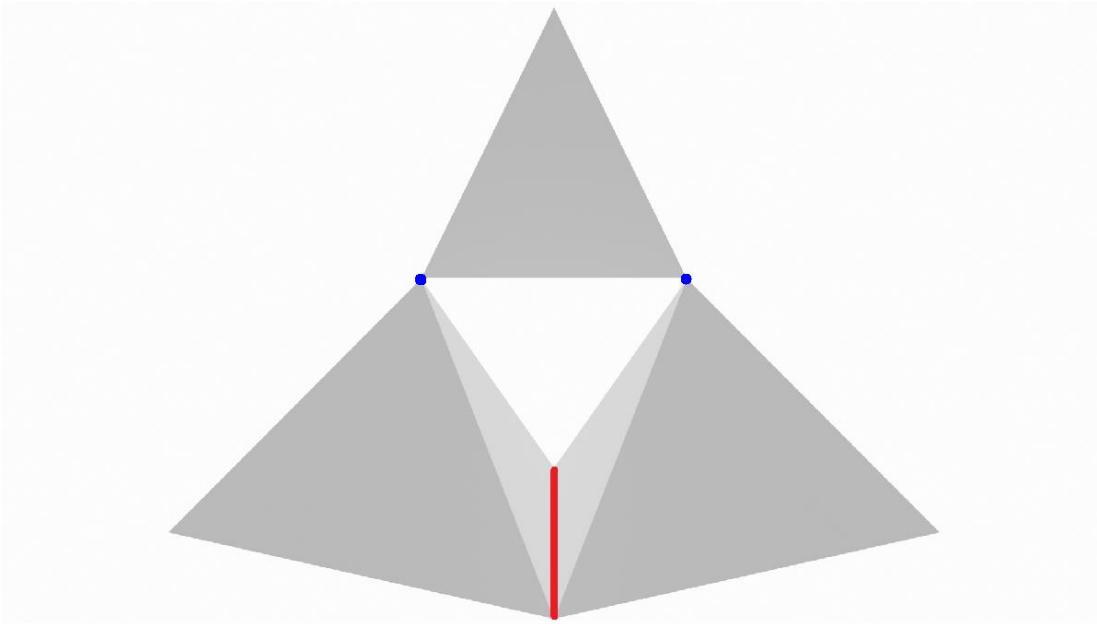

Fig. 4 Ejemplo de una malla de tetraedros tipo *conformal*, pero no *manifold*. En azul se marcan los vértices que crean una singularidad y en rojo la arista que crea una singularidad.
Fuente: elaboración propia.

Sea T un solo tetraedro, éste se puede remover de manera segura, es decir, no crea singularidades en la malla: si T no tiene vértices, aristas o caras en la superficie de la malla, si T tiene una o dos caras en la superficie de la malla y ninguno de sus vértices o aristas opuestos están en la superficie, o si T tiene 3 o 4 caras en la superficie.

Cuando se tiene un tetraedro T que tiene una o dos caras en la superficie de la malla y una o más aristas opuestas a dichas caras también están en la superficie, entonces la remoción de T remoción generaría una singularidad de arista en la malla. Para resolverla, se subdivide la o las aristas que causan la singularidad, con lo cual se "separa" la malla y T se puede remover de manera segura. Fig. 5 muestra algunos ejemplos de este proceso.

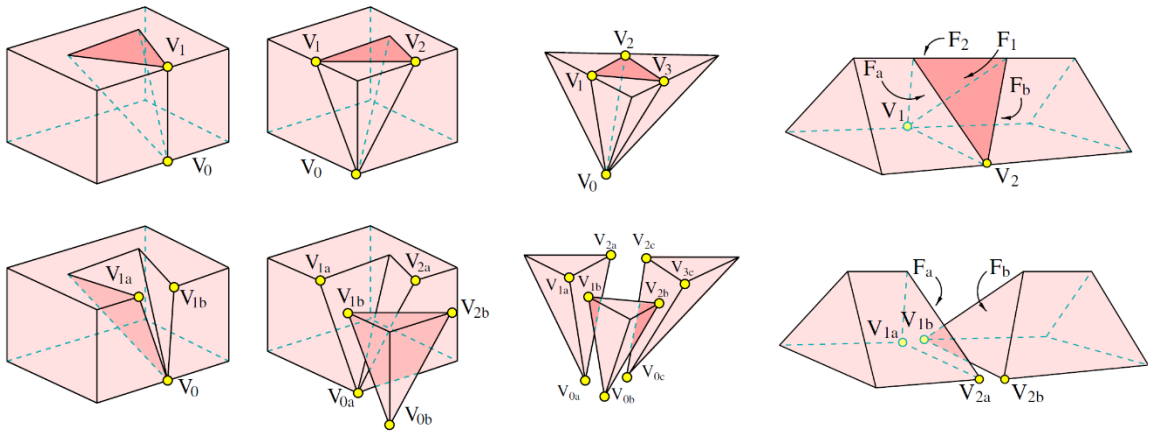

Fig. 5 Ejemplos de remoción de tetraedros que generarían singularidades de arista y su remoción mediante el algoritmo explicado en esta sección. En la parte superior se encuentran ejemplos de tetraedros a remover (resaltados en rosa) y directamente debajo de cada ejemplo su remoción mediante este algoritmo usando separación de aristas.
Adaptado de: [34].

Cuando se tiene un tetraedro T que tiene una cara en la superficie de la malla, pero ninguna de sus aristas restantes está en la superficie, entonces la remoción de T generaría una singularidad de vértice. Para resolverla, se procede a buscar un conjunto removible de tetraedros $\mathcal{T}$ que contenga tetraedros adyacentes a T y que sea del menor tamaño posible, ya sea en cuanto a cardinalidad del conjunto o a volumen total a remover. Este conjunto $\mathcal{T}$ se crea mediante una búsqueda en anchura sobre los vértices adyacentes a T para encontrar otros tetraedros removibles. Si se llega a una profundidad de 10 en la búsqueda o simplemente no se encuentra dicho conjunto $\mathcal{T}$, entonces se busca remover un tetraedro T' que sea adyacente al vértice que causa la singularidad mediante este mismo proceso. Si todo lo anterior falla, no se realiza ninguna remoción, pero afortunadamente, como se verá más adelante, este caso es muy raro y, además, generalmente se puede remover T tras realizar alguna otra operación donde se remueva otro tetraedro T' en su vecindad.

Los autores probaron su algoritmo removiendo aleatoriamente tetraedros de diversas mallas de prueba con entre 1,000 y 10,000 tetraedros en una computadora con un procesador Pentium III. Encontraron que pudieron remover el 99.8 % de todos los tetraedros deseados con un tiempo promedio de 0.2 ms, aunque en algunos casos el

tiempo aumentó a hasta 5 ms cuando había que hacer búsquedas muy extensas de conjuntos removibles.

Cabe notar que, si bien el método de los elementos finitos no requiere que una malla sea *manifold*, las mallas de este tipo suelen resultar en una mayor estabilidad en la simulación y además esta propiedad es esencial para realizar ciertas operaciones sobre la malla, como calcular las normales de sus vértices u obtener los vértices adyacentes a otro vértice [34].

2.5 Conversión de una malla de superficie a una malla de tetraedros

Como se discute en las secciones 2.3.2.1 y 2.4.2, es conveniente utilizar mallas de volumen para la simular las deformaciones y cortes de los tejidos. En este contexto es común que se utilicen mallas volumétricas a partir de tetraedros.

La mayoría de los software de modelado tridimensional trabajan sobre mallas de superficie, que no son compatibles con los métodos de simulación descritos en este capítulo. Afortunadamente, existen diversos softwares y algoritmos que permiten convertir una malla superficial en una de volumen con tetraedros mediante un proceso llamado tetraedralización, uno de los cuales de describe a continuación.

2.5.1 El software TetGen y su método de tetraedralización de mallas

TetGen es un software libre de código abierto capaz de tetraedralizar mallas con alta calidad para su uso en métodos numéricos y cómputo científico [35]. En esta sección, se describe el software y su algoritmo de tetraedralización.

El algoritmo toma como entrada una malla tipo *piecewise linear complex* (PLC), es decir, un conjunto de vértices, aristas, polígonos y poliedros que no se intersecan entre sí. Se asume que esta malla fue previamente creada.

El primer paso del algoritmo es hacer una tetraedralización de Delaunay a partir de los vértices del PLC de entrada. En otras palabras, primero se crea un conjunto de tetraedros a partir de los vértices del PLC, tal que la esfera circunscrita en cada tetraedro no contiene vértices en su interior. Una forma de lograr esto es mediante el algoritmo de Bowyer-Watson [36, 37], el cual, a grandes rasgos, funciona insertando uno a uno nuevos vértices sobre el PLC y removiendo los tetraedros cuyas esferas circunscritas contengan el vértice recién insertado, con lo cual se creará un "agujero" en la malla que será retriangulado utilizando los vértices existentes y el recién agregado.

El segundo paso del algoritmo es realizar (si es posible) una tetraedralización de Delaunay restringida sobre el PLC de entrada, es decir, una tetraedralización con la menor cantidad de elementos posible y que contenga las restricciones (aristas y triángulos) de la malla inicial. Para lograrlo, el algoritmo toma la malla generada en el paso anterior y le agrega los segmentos y triángulos del PLC de entrada. Si no fuera posible esta tetraedralización, el usuario puede permitirle al algoritmo que se pierdan algunas restricciones, lo cual se puede lograr agregando nuevos vértices para subdividirlas (puntos de Steiner).

El último paso del algoritmo es generar una tetraedralización de calidad, es decir, una malla con pocos tetraedros, los cuales deben ser pequeños y lo más parecidos a un tetraedro regular. Para lograrlo se actualiza la malla generada en el paso anterior mediante una adaptación del algoritmo de refinamiento de Delaunay. Este algoritmo, en resumen, busca en la malla de entrada tetraedros de mala calidad y calcula los centros de sus esferas circunscritas. Luego, agrega como vértices los centros recién calculados y los conecta con otros vértices para generar tetraedros de mejor calidad para finalmente actualizar la malla con estos tetraedros.

Los autores compararon su programa con otros dos programas con fines similares, haciendo las mismas tetraedralizaciones sobre seis mallas diferentes en cada uno de los programas. Encontraron que, en la mayoría de los casos, generaba mallas de calidad similar o superior, pero en menor tiempo.

2.6 Renderizado háptico

El renderizado háptico se puede definir como "el proceso de calcular y generar fuerzas como respuesta a la interacción entre objetos virtuales" [38]. Para llevar a cabo este proceso, el usuario mueve una herramienta virtual utilizando un dispositivo háptico, que es un dispositivo robótico reversible de baja inercia, mismo que da retroalimentación de fuerza al usuario.

El renderizado háptico es muy importante en un simulador médico. Imaginemos que un usuario de un simulador interactúa con un tejido virtual a través de un dispositivo háptico. Gracias a la simulación biomecánica y el renderizado de la imagen, el usuario podrá ver una respuesta realista a su interacción, pero el renderizado háptico le permitirá además *sentir* táctilmente esta respuesta [25], como si realmente estuviera interactuando con un tejido a través de una herramienta.

Sin embargo, propone un problema complejo, pues a diferencia de la frecuencia de actualización del renderizado visual, que es de 30 Hz, las fuerzas hápticas se deben actualizar con una frecuencia de 1 kHz para dar una sensación realista [25, 39, 38].

Existen varios métodos para hacer renderizado háptico. Estos métodos se pueden clasificar con base en diversas características del renderizado, por ejemplo, [38] los clasifica de acuerdo con la forma en que cada método maneja la respuesta a las colisiones en la simulación, entre los que destacan los métodos de renderizado basados en penalización y basado en restricciones.

En los métodos basados en penalización, las restricciones de los contactos se modelan como resortes cuya energía elástica aumenta conforme la herramienta virtual penetra un objeto en el ambiente virtual y las fuerzas de penalización se calculan como el gradiente negativo de dicha energía elástica, lo cual empuja la herramienta para mitigar la penetración. Estos métodos tienen la ventaja de que los modelos de fuerza son locales y relativamente simples, por lo que son adecuados para alcanzar frecuencias de actualización de 1 kHz. Además, pueden utilizar esquemas de acoplamiento virtual para agregar una rigidez amortiguada entre el dispositivo real y el virtual, mejorando así su

estabilidad. Sin embargo, en los métodos basados en penalización no se tiene control directo sobre ciertos parámetros físicos, se dificulta modelar las fuerzas de fricción y, si se utiliza acoplamiento virtual, este podría filtrar cambios sutiles en la fuerza renderizada, empobreciendo la sensación háptica.

Los métodos basados en restricciones pretenden restringir el movimiento de la herramienta virtual para que no tenga penetraciones con otros objetos virtuales, aunque la herramienta real sí las tenga. Para ello, se modelan todos los contactos concurrentes como como un solo problema de optimización que se resuelve utilizando multiplicadores de Lagrange (similar a como se describe en la sección 2.3.3) para encontrar fuerzas que produzcan movimientos física y geométricamente válidos. Estos métodos son costosos computacionalmente, pero producen una sensación háptica realista, por lo que se han desarrollado diferentes formas de acelerar el cómputo. Una de estas formas se describe formalmente en [39] y se resume en la siguiente sección.

2.6.1 Aceleración del renderizado háptico basado en restricciones con una matriz de conformidad aproximada

Como se menciona en la sección anterior, una de las formas de hacer renderizado háptico es verlo como un problema de optimización, donde se busca una respuesta adecuada a la colisión bajo las restricciones de la simulación utilizando multiplicadores de Lagrange, brindando una sensación háptica realista. Sin embargo, este método es computacionalmente costoso, principalmente porque requiere que por cada objeto se calcule la matriz de conformidad, la cual resulta del cálculo de la inversa de otra matriz que puede o no cambiar en cada paso de la simulación [31].

En [39] se define un método para acelerar el renderizado háptico basado en restricciones que se basa en precalcular la matriz de conformidad y su inversa, con lo cual se logra alcanzar frecuencias de actualización de fuerzas hápticas de 1kHz. Este método se describe brevemente a continuación.

La simulación utilizando este método supone cuatro pasos: primero un paso donde se mueven libremente los modelos sin considerar fuerzas de contacto, luego un paso donde se determina qué puntos del tejido simulado están en contacto con la herramienta virtual, posteriormente un paso que calcula la respuesta a la colisión basándose en los contactos y las leyes de fricción en cada punto de colisión, y finalmente un paso donde se aplica el desplazamiento con base en las fuerzas calculadas en el paso anterior.

En los primeros dos pasos, los objetos se mueven libremente, por lo que podría haber penetraciones entre objetos. Para evitar estas interpenetraciones, en el tercer paso se calculan las fuerzas de contacto formulándolo como un problema de complementariedad lineal (LCP, por sus siglas en inglés), el cual a su vez requiere del cálculo de la matriz de conformidad de cada objeto:

$$\mathrm{C} = \left(\frac{1}{h^2}\mathrm{M} + \frac{1}{h}\mathrm{B} + \mathrm{K}\right)^{-1}, \tag{29}$$

donde M es la matriz de masa, B la matriz de amortiguamiento, K la matriz de rigidez y h el paso de tiempo.

El método se puede volver complejo cuando hay deformaciones no lineales, causando que C se actualice cada vez que cambie M, B o K. Entonces, para acelerar el método, los autores proponen precalcular una matriz de conformidad en reposo C^0. Con esta matriz y con la matriz de rotación de cada nodo del objeto en cada paso, se puede obtener una matriz de conformidad aproximada $\tilde{C}$, con lo cual se puede resolver el LCP y con ello obtener las fuerzas de contacto.

Para ser congruentes con el LCP, esta matriz $\tilde{C}$ también se utiliza en el último paso de la simulación. Si bien esto causa que toda la simulación sea congruente, resulta en un movimiento aproximado del modelo, pues no se basa en la matriz de conformidad real. Sin embargo, los autores mencionan que esto no es un problema, pues se utilizan las

leyes reales de contacto y de fricción y además la aproximación se corrige parcialmente con el movimiento libre que surge al iniciar el siguiente paso de la simulación.

Para garantizar que no haya interpenetraciones, los autores proponen un esquema de acoplamiento basado en el principio del *god-object*. Éste es un objeto virtual que se acopla a la herramienta virtual usando técnicas de control de impedancia y que además está sujeto a las leyes de la simulación, por lo que no puede penetrar otros objetos. Este *god-object* se trata como cualquier otro objeto en la simulación, es decir, también sigue los cuatro pasos descritos por los autores.

Para acelerar aún más el proceso, los autores proponen separar el hilo de simulación del hilo háptico, permitiendo que éste último corra a una mayor frecuencia. Cuando se resuelve el LCP en el hilo de simulación, se comparte con el hilo háptico. Éste actualiza la posición del dispositivo y resuelve su propio LCP utilizando la solución recibida y la posición actualizada, generando la fuerza a enviar al dispositivo. Este proceso se repite a una frecuencia 1kHz, recibiendo una nueva solución al LCP con cada actualización del hilo de simulación.

2.6.2 El dispositivo háptico Touch™ de 3D Systems

Touch™ es un dispositivo háptico profesional de gama media desarrollado por 3D Systems, el cual puede brindar una retroalimentación de fuerza al usuario, permitiéndole sentir los objetos 3D existentes en una escena virtual a medida que los manipula guiándose por una pantalla [40]. Se puede utilizar en diversas aplicaciones como simuladores o programas de esculpido, rehabilitación, aprendizaje, entre otros. También, se pueden crear programas hápticos para este dispositivo utilizando la API *OpenHaptics*.

La Tabla 1 muestra las especificaciones técnicas de este dispositivo y Fig. 6 es una imagen real del mismo.

Tabla 1. Especificaciones técnicas del dispositivo Touch™ de 3D Systems.
Fuente: Adaptado de [41].

Área de trabajo	160 mm de ancho, 120 mm de alto y 20 mm de profundidad.
Rango de movimiento	Movimiento de la mano pivotando en la muñeca.
Resolución de posición nominal	450 dpi, ~0.055 mm.
Fuerza ejecutable máxima	3.3 N
Rigidez	Eje *x*: 1.26 N/mm Eje *y*: 2.31 N/mm Eje *z*: 1.02 N/mm
Retroalimentación de fuerza	Ejes *x*, *y* y *z* (tres grados de libertad).
Detección de posición/entrada	Ejes *x*, *y* y *z* mediante *encoders* digitales y cabeceo, alaveo y guiñada mediante potenciómetros lineales (seis grados de libertad).
Interfaz	USB 2.0

Fig. 6 Imagen del dispositivo Touch™ de 3D Systems.
Fuente: [40].

2.7 Despliegue de imágenes en tres dimensiones

En la actualidad, la forma más común de desplegar imágenes en tres dimensiones es mediante un casco de realidad virtual [42]. Estos cascos bloquean toda la luz exterior y utilizan *displays* para mandar una imagen diferente a cada ojo, donde cada imagen muestra al mismo objeto o escena, pero con ligeros cambios en la perspectiva, dando la sensación de una imagen tridimensional, de acuerdo con el principio de estereoscopía (que se describe en la sección 2.7.1). Adicionalmente, estos cascos suelen incluir sensores de movimiento, de modo que cuando el usuario mueve su cabeza, la imagen desplegada se mueve correspondientemente, dando la sensación de que el usuario está inmerso en la imagen.

En esta sección, se describe el principio de estereoscopía bajo el que funcionan los cascos de realidad virtual y se describen las características del casco de realidad virtual Oculus Rift CV1.

2.7.1 Estereoscopía

La estereoscopía se puede definir como una técnica que se puede aplicar a un dibujo o imagen bidimensional, tal que, al ser vista con ambos ojos, parece ser tridimensional [43].

Para poder percibir la profundidad en nuestra vida diaria, el cerebro combina las imágenes percibidas por cada ojo, las cuales son ligeramente diferentes debido a su posición y enfoque. La estereoscopía toma ventaja de este principio y logra replicar esta percepción de profundidad generando dos imágenes con el mismo objeto o escena visualizada desde ángulos ligeramente diferentes, que corresponden con los ángulos de visión de los ojos de una persona. Este efecto es más efectivo si el ojo derecho ve únicamente la imagen derecha y el ojo izquierdo ve sólo la imagen izquierda, como sucede con un casco de realidad virtual.

2.7.2 El casco de realidad virtual Oculus Rift CV1

Oculus Rift CV1 es un casco de realidad virtual creado por Oculus VR que se lanzó al mercado en 2016. Actualmente está descontinuado, pero sigue teniendo soporte de la empresa [44].

La Tabla 2 muestra las especificaciones técnicas de este dispositivo, Fig. 7 es una imagen real del mismo. En la tabla se aprecia que la tasa de actualización del despliegue gráfico es de 90 Hz y que cuenta con seis grados de libertad para registrar el movimiento, lo cual permite brindar una experiencia potencialmente inmersiva y en tiempo real.

Tabla 2. Especificaciones técnicas del casco de realidad virtual Oculus Rift CV1.
Fuente: Adaptado de [44].

Tipo de display	PenTile OLED.
Resolución del display	1080 × 1200 pixeles por ojo.
Tasa de actualización del display	90 Hz.
Detección de posición	Seis grados de libertad (tres ejes de rotación en el casco y tres de posición mediante un sensor infrarrojo externo).
Conectividad	HDMI 1.3, USB 3.0, USB 2.0.

Fig. 7 Imagen del casco de realidad virtual Oculus Rift CV1.
Fuente: [44].

2.8 El software Simulation Open Framework Architecture (SOFA)

El software Simulation Open Framework Architecture (SOFA) es una biblioteca de C++ de código abierto enfocada en el desarrollo de simulaciones médicas interactivas por computadora [25]. SOFA permite descomponer escenas de simulación complejas en componentes independientes que representan algún aspecto de dicha simulación, los cuales se comunican entre sí para otorgar un resultado visual y mecánicamente realista.

2.8.1 Representación multimodelo

SOFA descompone la simulación en tres modelos: uno encargado del comportamiento mecánico, otro encargado de la detección y respuesta de las colisiones, y uno encargado del renderizado visual, pero todos se conectan entre sí mediante un "grafo de escena" [25]. A continuación, se describe brevemente cada uno de estos modelos y cómo interactúan entre sí.

2.8.1.1 Modelo mecánico

El modelo mecánico es el encargado del comportamiento mecánico de los objetos.

En SOFA, los objetos rígidos y deformables se modelan mediante un conjunto de nodos, cuyas coordenadas son independientes a las del objeto y se rigen por ecuaciones como:

$$a = PM^{-1}\sum_{i} f_i(x, v) \tag{30}$$

donde x es el vector de posición, v un vector de velocidad, f_i es una función de fuerza, M es la matriz de masa y P es una matriz de proyección con las condiciones de frontera para los desplazamientos.

Cada uno de los términos de esta ecuación se modela a través de componentes de SOFA, como se aprecia en Fig. 8.

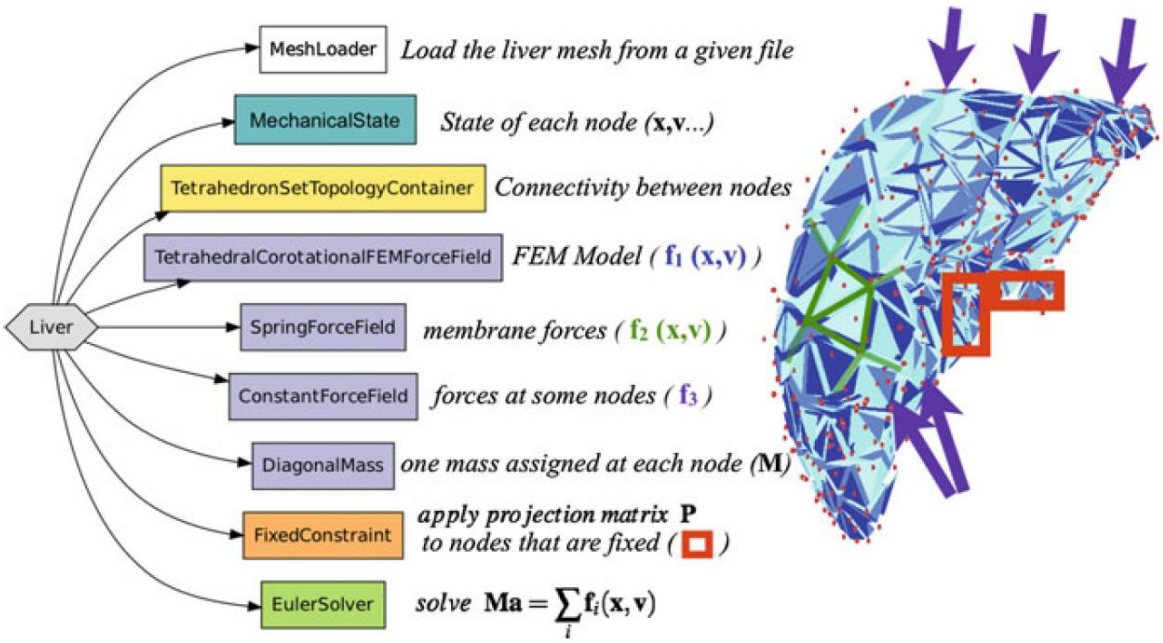

Fig. 8 Modelo mecánico de un hígado en SOFA. Las cajas representan nodos fijos de la malla y las flechas representan fuerzas externas. A la izquierda se aprecia el grafo de escena con "Liver" como padre y sus diversos componentes como hijos. A la derecha de cada componente se aprecia el término que representa en la ecuación $a = PM^{-1}\sum_i f_i(x,v)$.
Fuente: [25].

Cabe notar que cada uno de los componentes en el modelo mecánico es independiente entre sí. De esta forma, si se quisiera, por ejemplo, cambiar de un modelo de masas y resortes a uno de elemento finito, únicamente habría que cambiar un componente, dejando el resto intactos.

2.8.1.2 Modelo de colisión

El modelo de colisión es el encargado de detectar las colisiones entre modelos y generar una respuesta a ellas.

SOFA implementa diferentes modos de detección de colisiones, los cuales se aplican sobre una estructura de datos adicional que almacena la topología y geometría del modelo con la que se hará la detección de colisiones. Esto permite aislar el modelo de colisión del resto de los modelos y hacer la detección de colisiones con una malla más o menos compleja que la malla interna, según sea necesario.

2.8.1.3 Modelo visual

El modelo visual es el encargado del renderizado gráfico.

Este modelo también es, hasta cierto punto, independiente del resto, permitiendo cargar mallas más o menos densas que las utilizadas para el modelo mecánico o el de colisión. Además, SOFA implementa su propia biblioteca basada en OpenGL, la cual permite agregar texturas, iluminación, sombras y *shaders*, entre otras cosas. Todo esto en conjunto permite hacer una simulación realista más allá del comportamiento mecánico.

2.8.1.4 Mapeo entre modelos

En SOFA, cada objeto se suele simular utilizando un modelo mecánico, uno de colisión y uno visual. Para que haya consistencia entre ellos, se utiliza una comunicación de tipo maestro/esclavo por medio de mapeos. En esta comunicación, generalmente el modelo mecánico es el maestro y comunica los desplazamientos y velocidades de los nodos del objeto a sus esclavos, que generalmente son el modelo visual y el de colisión. Posteriormente, los esclavos calculan las fuerzas de interacción entre objetos y las mandan de vuelta al maestro. De esta forma, se pueden calcular y visualizar los desplazamientos, a la vez que se aplican fuerzas sobre el modelo.

2.8.2 Plugins

Un *plugin* es una colección de componentes de SOFA que se pueden utilizar en una escena por medio de bibliotecas de enlace dinámico [45].

Por defecto, SOFA incluye muchos de los componentes que se requieren para hacer una simulación. Sin embargo, es posible generar *plugins* utilizando C++ para dar funciones adicionales al software.

Adicionalmente, el consorcio de SOFA pone a disposición de los usuarios un mercado virtual (disponible en [46]) donde es posible subir y descargar *plugins* hechos por la comunidad.

3 Desarrollo del simulador

El simulador trata de replicar únicamente el proceso de resección de un tumor cerebral (descrito brevemente en la sección 2.1) utilizando un aspirador ultrasónico. Se asume que previamente se hizo la preparación necesaria para dejar expuesta la zona donde se encuentra el tumor.

La escena de simulación computarizada consta de modelos deformables del cerebro y de un meningioma, a los cuales se les pueden quitar elementos para simular la remoción de tejido. El usuario puede interactuar con la simulación en tiempo real utilizando dos dispositivos hápticos, simulando un aspirador ultrasónico y una pinza bipolar. Además, el usuario puede ver la escena en tres dimensiones usando estereoscopía mediante un casco de realidad virtual y ubicarse espacialmente utilizando un modelo de cráneo impreso en 3D. Todo esto se aprecia más claramente en Fig. 9 y Fig. 10.

Durante la simulación, el usuario debe utilizar el aspirador ultrasónico para intentar remover la mayor cantidad de tejido tumoral y la menor cantidad de tejido sano posible. También puede utilizar la pinza bipolar para desplazar tejidos y mejorar su visibilidad.

El usuario puede terminar la simulación en cualquier momento, pero la idea es que lo haga cuando considere que ya terminó de remover la mayor cantidad posible de tejido tumoral. Una vez terminada la simulación, el programa arroja métricas que permiten evaluar su desempeño.

En este capítulo se describe a detalle cómo se implementó lo descrito anteriormente y se justifica por qué se hizo de esa forma. En la primera sección se habla sobre la creación escena de simulación en la computadora y la inclusión del hardware externo, luego de la obtención de las mallas 3D que representan los objetos de simulación, posteriormente de la construcción de la estación de trabajo, después de la generación de métricas de desempeño del usuario, y finalmente de la GUI de control y ejecución del simulador.

Fig. 9 Versión final del simulador. Del lado izquierdo en la computadora se aprecia la escena de simulación y del lado derecho, la estación de trabajo con el hardware externo y los elementos físicos.
Fuente: elaboración propia.

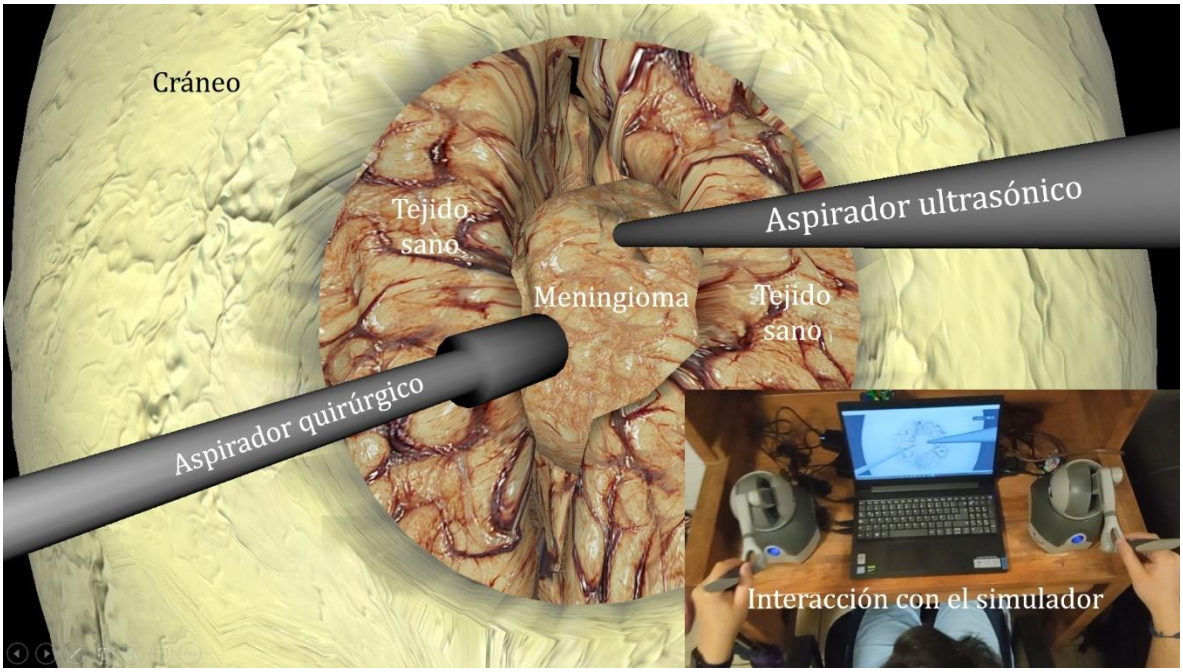

Fig. 10 Captura de pantalla de la simulación por computadora en progreso (sin estación de trabajo ni dispositivos de realidad virtual). Al centro se aprecian los modelos del tumor, del tejido cerebral sano, del cráneo y de las herramientas controladas por el usuario (el aspirador ultrasónico y el aspirador quirúrgico, que después se remplazó con la pinza bipolar). Se observa cómo se está removiendo un segmento del tejido del tumor. En la esquina inferior derecha se aprecia la interacción del usuario con el simulador mediante dispositivos hápticos.
Fuente: elaboración propia.

3.1 Modelo de simulación

La escena de simulación está implementada totalmente en el *framework* SOFA. Ésta consta de los modelos de un tumor, una porción de cerebro que rodea el tumor, un cráneo con una craneotomía y un par de herramientas controladas por el usuario, que son el aspirador ultrasónico y la pinza bipolar.

Cabe notar que los modelos del tumor y del tejido cerebral sano son deformables y permiten la remoción de sus elementos, simulando la extracción de tejido. El aspirador ultrasónico y la pinza bipolar son objetos rígidos que se mueven de acuerdo con los movimientos del usuario. El cráneo es un modelo meramente visual.

Los modelos deformables en este simulador cumplen con los requerimientos propuestos por [21] para simular tejidos blandos en tiempo real: permiten hacer cortes, son físicamente realistas y sus deformaciones se calculan con suficiente rapidez para mantener la simulación en tiempo real.

En esta sección se describen las propiedades físicas de los tejidos blandos, así como la forma en que se implementó la simulación de los objetos rígidos y deformables en SOFA.

3.1.1 Propiedades de los tejidos blandos

Una suposición común en la simulación biomecánica en tiempo real es la de que los tejidos son isotrópicos, homogéneos y tienen un comportamiento elástico lineal [20, 27, 26]. Se ha reportado que hacer estas suposiciones permite hacer una simulación en tiempo real, aunque con una menor precisión, sobre todo para deformaciones muy grandes [20]. En este simulador, se optó por hacer la simulación biomecánica bajo estos supuestos para poder mantener una frecuencia de actualización gráfica superior a 30 Hz y bajo la suposición de que, por la misma naturaleza del procedimiento, los usuarios no harán desplazamientos muy grandes en los tejidos.

Todos los modelos deformables en el simulador están basados en el método de los elementos finitos (FEM) utilizando sus propiedades elásticas reales, pues éste ofrece

una mayor validez biomecánica que otros métodos. Como se explica en la sección 2.3.2.1, el FEM funciona con base en los parámetros elásticos reales de los objetos a simular. Estos parámetros se eligieron con base en los estudios presentados en [47] y se muestran en la Tabla 3.

Tabla 3. Parámetros físicos de los modelos del cerebro y el tumor.
Fuente: elaboración propia.

Modelo	*Módulo de Young*	*Coeficiente de Poisson*	*Densidad de Masa*
Cerebro	3 kPa	0.45	1188.21 kg/m^3
Tumor	50 kPa	0.45	6666.67 kg/m^3

En [47] se aprecia que diversos estudios concluyen que el módulo de Young para el cerebro es de 3 kPa o menos. Para el tumor, se concluye que el módulo de Young siempre es mayor que el del cerebro, aunque los resultados varían entre los 9 y los 180 kPa. Por otra parte, en la mayoría de los estudios se obtiene un coeficiente de Poisson de 0.45, tanto para el cerebro como para el tumor. Teniendo esto en cuenta, se optó por asignarle un módulo de Young de 3 kPa al cerebro y de 50 kPa al tumor. En todos los casos, se utilizó un coeficiente de Poisson de 0.45.

La densidad de masa ρ se calcula mediante la relación:

$$\rho = \frac{m}{V}, \tag{31}$$

donde m es la masa del objeto y V es su volumen.

Como se concluye en [48], la masa promedio del cerebro de una mujer adulta sana es de 1198 gramos y de 1336 gramos para un hombre adulto sano. Teniendo esto en cuenta, se consideró una masa de 1250 gramos para el cerebro. Sin embargo, como se explica en la sección 3.2.1, el modelo del cerebro utilizado abarca únicamente el ~16% de su volumen, por lo que se consideró también sólo el 16% de su masa total, es decir, 200 gramos. De acuerdo con las herramientas de Blender, este modelo tiene un

volumen de 0.000168 m^3, lo que resulta en una densidad de masa de 1,188.21 kg/m^3 para el cerebro.

No se encontró información relacionada con la masa promedio de un meningioma. De acuerdo con las herramientas de Blender, el volumen de la malla utilizada para el tumor es de 0.000015 m^3 y como se mencionó anteriormente, [47] concluye que su módulo de Young siempre es mayor que el del cerebro, es decir, es más rígido. Con este hecho en mente, se hizo la suposición de que el meningioma tiene una masa de 100 gramos, lo que resulta en una densidad de masa de 6666.67 kg/m3, que es mayor a la del cerebro, lo cual explicaría que su rigidez también es mayor. Sin embargo, en un futuro se espera obtener información más precisa, a través de la experiencia clínica de los médicos, o experimentalmente, para obtener su peso real y ajustarlo en el simulador.

3.1.2 Simulación de objetos deformables en SOFA

Como se explica en la sección 3.1, el modelo de simulación cuenta con dos objetos que representan tejidos blandos: el cerebro y el tumor. Ambos objetos son similares en el sentido de que ambos son deformables y a ambos se les puede remover tejido, y por esta razón, ambos se definieron con una estructura muy similar en SOFA. También, se optó por utilizar tetraedros para discretizar el volumen de ambos elementos, como se discute a detalle en la sección 3.2.1.

Como se explica en la sección 2.8.1, en SOFA los objetos se definen mediante nodos que a su vez contienen subnodos donde se define su modelo mecánico, de colisión y visual mediante componentes que representan sus propiedades. En esta sección se explica en qué consistió cada uno de estos modelos para los objetos deformables.

Cabe notar que en todas las subsecciones siguientes y las imágenes que incluyen, con el fin de generalizar, cuando se habla del cerebro, se refiere únicamente a la porción del cerebro que rodea al tumor. La razón de esto se explica en la sección 3.2.1.

3.1.2.1 Modelo mecánico

Para cargar las mallas de tetraedros para cada objeto, se utilizó el componente *MeshGmshLoader*, el cual recibe una malla de tetraedros en formato Gmsh (extensión ".msh") y almacena sus propiedades. Posteriormente, se utilizó el componente *MechanicalObject* para especificar los grados de libertad de este objeto con base en la malla recién cargada. En Fig. 11 se aprecian estos grados de libertad.

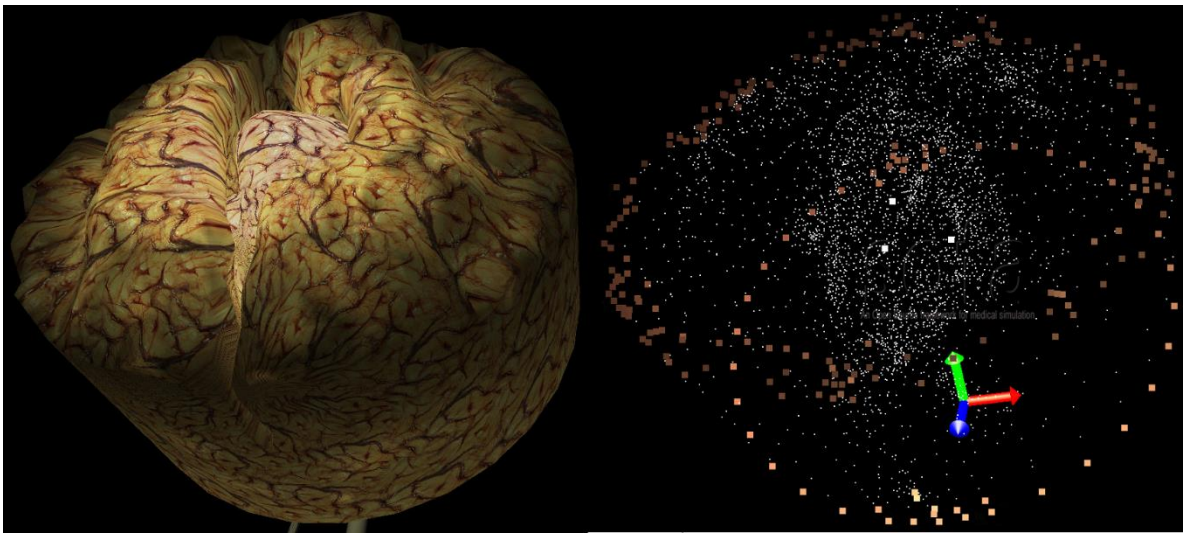

Fig. 11 Modelo visual (izquierda) y restricciones creadas con FixedConstraint (derecha) para el cerebro rodeando el tumor. Cada punto grande, ya sea café o blanco representa un vértice cuyo movimiento se restringió. Los puntos blancos pequeños representan los grados de libertad del modelo mecánico. Fuente: elaboración propia.

Para el cálculo de las deformaciones de los objetos, se utilizaron varios métodos numéricos incluidos en SOFA, los cuales se especifican mediante los componentes:

- *TetrahedronFEMForceField* para el método de los elementos finitos (FEM) con mallas de tetraedros (descrito en la sección 2.3.2.1), con el cual se calculan los desplazamientos de los nodos de las mallas,
- *EulerImplicitSolver* para el método de Euler implícito (descrito en la sección 2.3.2.2), con el cual se resuelven las ecuaciones diferenciales ordinarias que resulten del FEM, y
- *CGLinearSolver* para el método del gradiente conjugado (descrito en la sección 2.3.2.3), con el cual se resuelven los sistemas de ecuaciones generados por las

restricciones para el manejo estable de las colisiones y renderizado háptico de fuerzas que aparezcan durante la simulación.

Cada objeto cuenta con su propia definición de estos componentes. En el caso de *TetrahedronFEMForceField* cada objeto tiene sus propiedades elásticas (definidas en la sección 3.1.1) y sus grados de libertad particulares de acuerdo con su *MechanicalObject*. Por otra parte, para los componentes *EulerImplicitSolver* y *CGLinearSolver* se mantuvieron la mayoría de las propiedades que SOFA les asigna por defecto, excepto la cantidad de iteraciones máxima de *CGLinearSolver*, que se bajó de 25 a 10 porque se notó que con esto mejoraba el desempeño del simulador y mejoraba considerablemente la calidad de la retroalimentación háptica. En Fig. 12 se aprecia el resultado de este modelo mecánico en reposo y en Fig. 13 sufriendo una deformación.

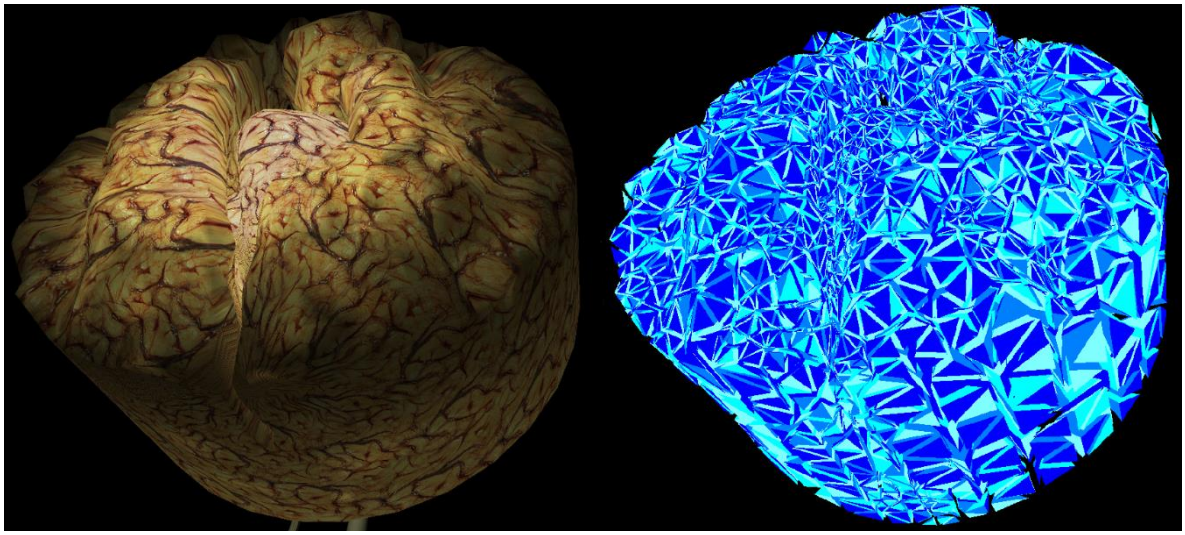

Fig. 12 Modelo visual (izquierda) y mecánico (derecha) del cerebro rodeando al tumor con ambos objetos en reposo.
Fuente: elaboración propia.

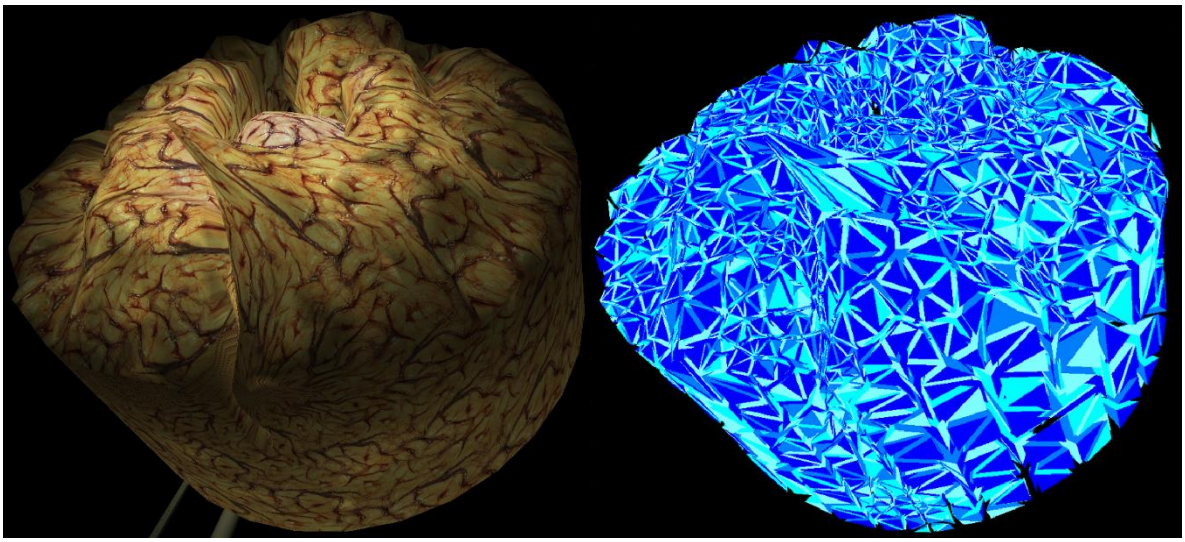

Fig. 13 Modelo visual (izquierda) y mecánico (derecha) del cerebro rodeando al tumor con el cerebro sufriendo una deformación en uno de sus grados de libertad.
Fuente: elaboración propia.

Por otra parte, para especificar la masa de los objetos, se utilizó el componente *MeshMatrixMass*, el cual recibe la densidad de masa y la malla de volumen que representa del objeto, e integra esta densidad de masa por toda la geometría recibida, obteniendo así la masa individual de cada elemento de la malla de volumen.

Dado que los objetos tienen un comportamiento mecánico realista, cuando éstos son expuestos a fuerzas externas, tienden a desplazarse completamente (además de deformarse). Para evitar este comportamiento y simular que tanto el cerebro como el tumor se pueden deformar, pero no desplazar, se forzó que algunos vértices del objeto se mantuvieran siempre fijos, sin importar la cantidad de fuerza que se ejerza sobre ellos. Para ello, se crearon mallas que definen regiones de interés (como se explica en la sección 3.2.2), las cuales se cargaron a la escena mediante el componente *MeshObjLoader*, se guardó la referencia de todos los vértices que cayeran dentro de esa malla con el componente *MeshROI* y finalmente se le pasaron estas referencias al componente *FixedConstraint* para que SOFA restringiera todo movimiento sobre ellos. El resultado se aprecia en Fig. 11.

Como se explica en la sección 2.4.1, para poder hacer remoción de tetraedros de una malla, es necesario tener una representación versátil de dicha malla, y la forma en la

que SOFA mantiene esta representación versátil es mediante objetos topológicos compuestos por cuatro miembros: Container, Modifier, Algorithms y Geometry. Éstos se especificaron en el modelo mecánico mediante los componentes *TetrahedronSetTopologyContainer*, *TetrahedronSetTopologyModifier*, *Tetrahedron-SetTopologyAlgorithms*, *TetrahedronSetGeometryAlgorithms*, respectivamente, los cuales se configuran a partir de *TetrahedronSetTopologyContainer*, que recibe como parámetro las propiedades de la malla previamente cargada. El resultado se aprecia en Fig. 14.

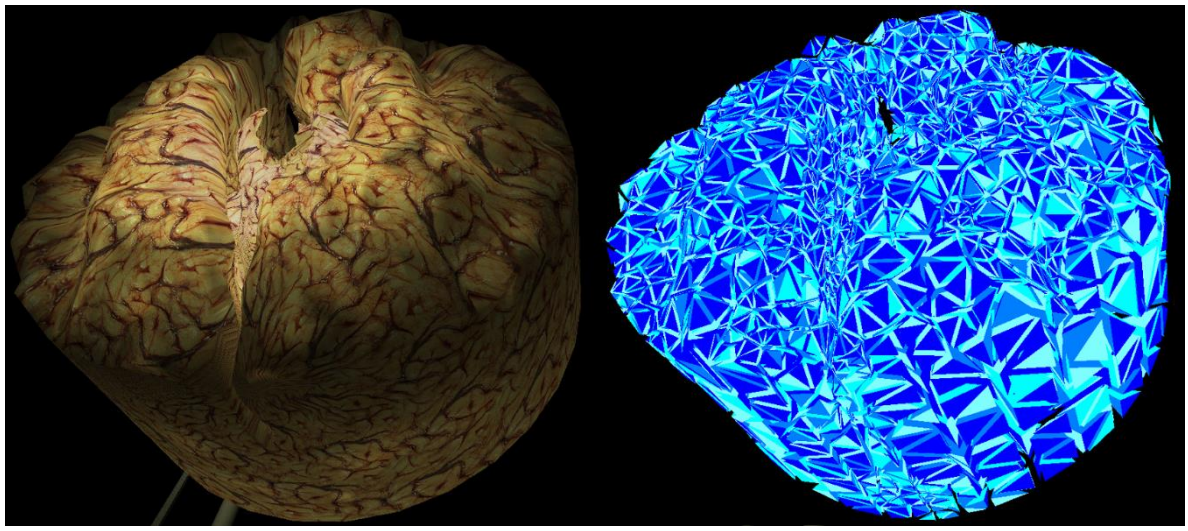

Fig. 14 Modelo visual (izquierda) y mecánico (derecha) del cerebro rodeando al tumor.
Se aprecia que se removieron algunos tetraedros de la zona del tumor.
Fuente: elaboración propia.

Finalmente, para mejorar aún más el desempeño del simulador y la calidad de la retroalimentación háptica, se utilizó el método de precomputación de matrices de conformidad definido en la sección 2.6.1, el cual está definido en el componente *PrecomputedConstraintCorrection*.

3.1.2.2 Modelo de colisión

El modelo de colisión del tumor y el cerebro también son muy similares. De hecho, lo único que cambia entre ellos es la malla que los representa. Por cada objeto, dicha malla se carga mediante el componente *MeshTopology* y es la misma que se utiliza en el modelo mecánico. También es necesario especificar los grados de libertad del modelo

de colisión, lo cual se hace con el componente *MechanicalObject*, pasándole la referencia del componente *MeshTopology* de este modelo. El resultado se aprecia en Fig. 15.

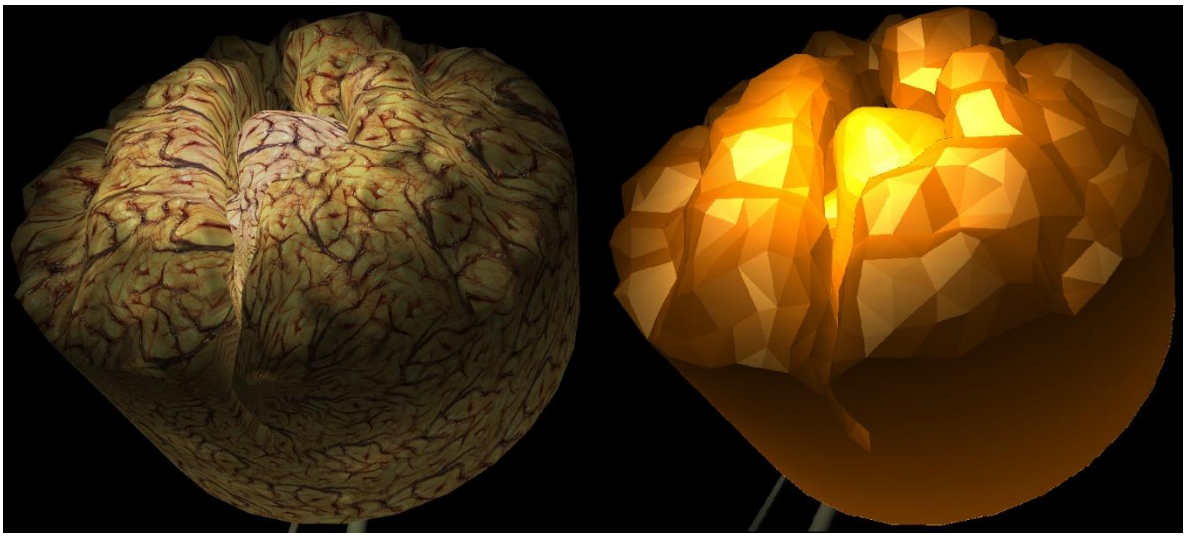

Fig. 15 Modelo visual (izquierda) y de colisión (derecha) del cerebro rodeando al tumor con ambos objetos en reposo.
Fuente: elaboración propia.

Al igual que en el modelo mecánico, para poder remover tetraedros de este modelo también es necesario definir los miembros *Container, Modifier, Algorithms* y *Geometry* del objeto topológico, pero en este caso se especifican sólo los triángulos de la malla mediante los componentes *TriangleSetTopologyContainer*, *TriangleSetTopology-Modifier*, *TriangleSetTopologyAlgorithms* y *TriangleSetGeometryAlgorithms*, y posteriormente se hace un mapeo entre los triángulos de este modelo y los tetraedros del modelo mecánico mediante el componente *Tetra2TriangleTopologicalMapping*.

Adicionalmente, en este modelo se especifica que sólo se detecten colisiones entre triángulos de las diferentes mallas mediante el componente *TriangleCollisionModel*. Cabe notar que en SOFA es posible especificar que también se detecten colisiones de cualquier combinación de polígonos, aristas y vértices entre los diferentes modelos, pero al habilitarlos en esta simulación, imposibilitaba que corriera en tiempo real.

También, es en este modelo donde se habilita el *trigger* de remoción de tetraedros. Cabe notar que estas etiquetas no vienen incluidas por defecto en SOFA, sino que se agregan mediante el plugin *SofaCarving*, como se discute en la sección 3.1.4.1.

Finalmente, como se discute en la sección 2.8.1.4, en SOFA es necesario especificar un mapeo entre modelos de modo que todos estén sincronizados entre sí. En este caso se utilizó el componente *IdentityMapping* para hacer un mapeo identidad entre cada grado de libertad del modelo de colisión y del modelo mecánico, de modo que cuando haya una deformación en el modelo mecánico, ésta también se refleje en el de colisión (como se aprecia en Fig. 16), o que cuando se remueva un tetraedro del modelo de colisión, su ausencia se refleje en el modelo mecánico (como se aprecia en Fig. 17).

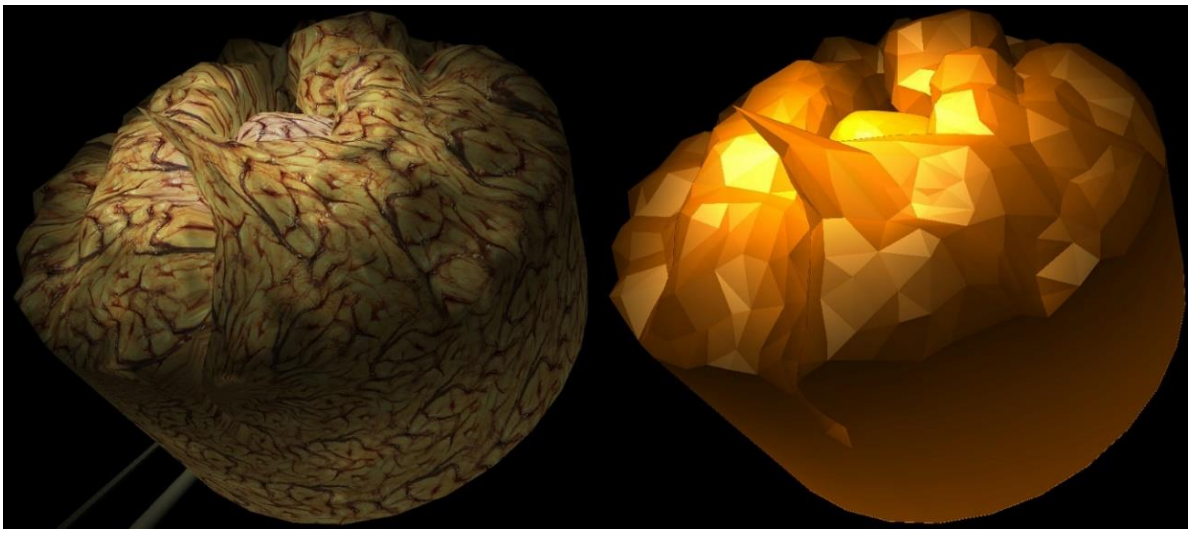

Fig. 16 Modelo visual (izquierda) y de colisión (derecha) del cerebro rodeando al tumor con el cerebro sufriendo una deformación en uno de sus grados de libertad.
Fuente: elaboración propia.

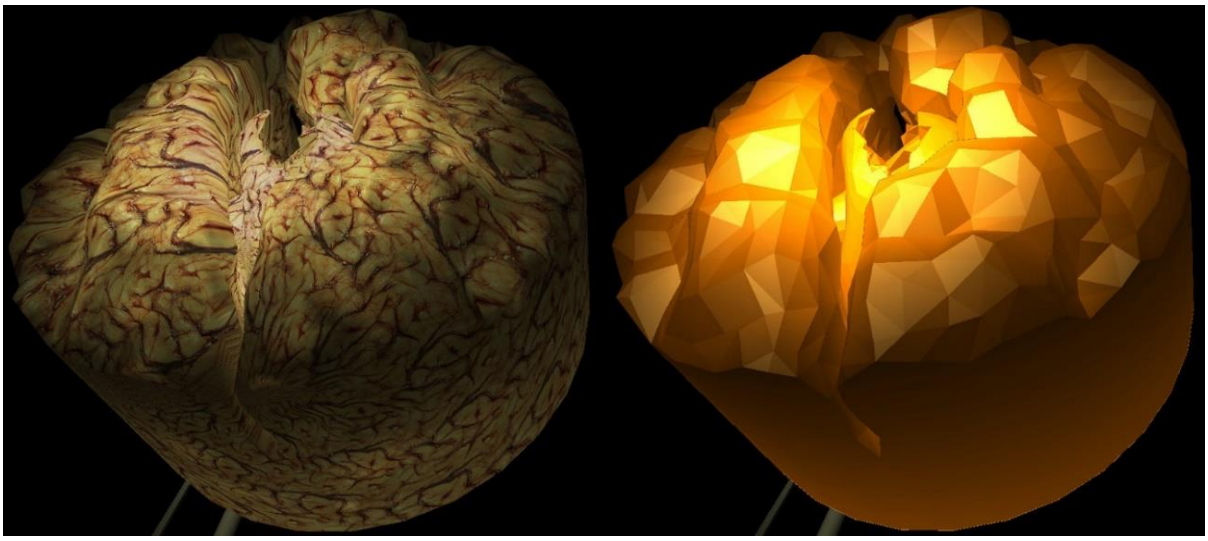

Fig. 17 Modelo visual (izquierda) y de colisión (derecha) del cerebro rodeando al tumor con algunos tetraedros del tumor removidos.
Fuente: elaboración propia.

3.1.2.3 Modelo visual

Para el objeto del tumor y del cerebro, el modelo visual es muy sencillo. Dado que se utiliza exactamente la misma malla que para los modelos mecánico y visual, únicamente es necesario especificar el nombre de la imagen que contiene textura y las componentes difusa, ambiental y especular del material que se utilizará sobre las mallas. Esto se logra mediante las etiquetas *texturename* y *material* del componente *OglModel*.

Para ambos objetos, el material se definió con una componente difusa color gris claro, una componente ambiental gris oscuro, con reflejos especulares blancos y un brillo moderado para lograr que las superficies parezcan húmedas. Por otra parte, las texturas se basaron en una imagen encontrada en el Marketplace de Second Life [49], las cuales se ajustaron en color y tamaño con la ayuda de un neurocirujano para que se vieran lo más realista posible. Este material y una versión reducida de las texturas utilizadas en ambos modelos se aprecian en Fig. 18.

Fig. 18 De izquierda a derecha: textura utilizada para el cerebro, textura utilizada para el tumor y representación del material utilizado en ambos objetos.
Fuente: elaboración propia.

Finalmente, al igual que en el modelo de colisión, se utilizó el componente *IdentityMapping* para hacer un mapeo identidad entre cada grado de libertad del modelo visual y del modelo mecánico, de modo que cuando se hagan cambios en los demás modelos, éstos se vean reflejados en el modelo visual, como se aprecia desde Fig. 12 hasta Fig. 16.

Cabe notar que toda la escena está expuesta a una luz tipo reflector, la cual se habilitó con el componente *LightManager* y se configuró con el componente *SpotLight*, asignándole un color cálido (amarillo claro), una posición similar a la de la cámara y con dirección hacia la craneotomía, como se aprecia en Fig. 19.

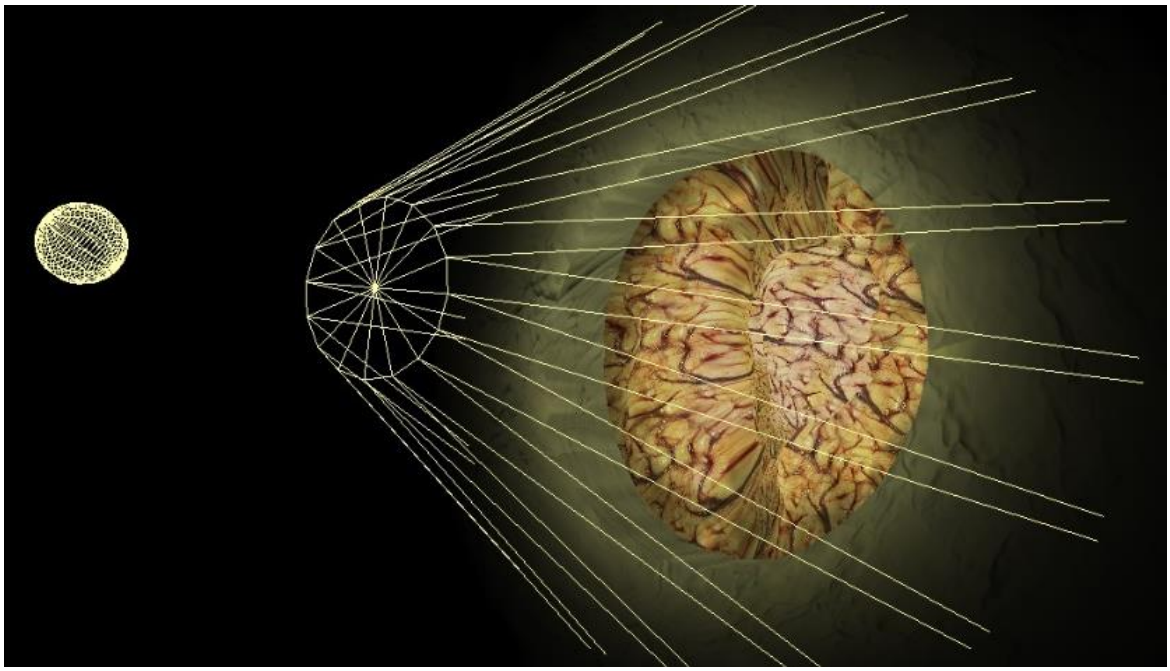

Fig. 19 Representación del "reflector" que ilumina la escena con una luz cálida. La esfera del lado izquierdo representa su ubicación y el cono truncado representa la superficie que ilumina.
Fuente: elaboración propia.

3.1.3 Simulación de objetos rígidos en SOFA

Como se explica en la sección 3.1, el modelo de simulación cuenta con dos objetos rígidos que son las herramientas que el usuario utiliza en cada mano: un aspirador ultrasónico y una pinza bipolar. Dado que estos objetos no son deformables ni requieren de remoción de elementos de sus mallas, sus definiciones en SOFA son relativamente más sencillas que las de objetos deformables.

Si bien, en esta simulación los objetos rígidos siempre son controlados por el usuario mediante dispositivos hápticos, en esta sección únicamente se definen los modelos mecánicos, de colisión y visual en SOFA (de acuerdo con lo descrito en la sección 2.8.1). En la sección 3.1.4.1 se describe la forma en que se sincronizaron estos objetos con los dispositivos externos.

3.1.3.1 Modelo mecánico

Para estos objetos, el modelo mecánico consta únicamente de un nodo en la punta de la herramienta, definido mediante el componente *MechanicalObject*, y de una masa, que en este caso se definió como una matriz diagonal donde todos sus elementos suman 1, mediante el componente *UniformMass*. Ambos componentes tienen sus propiedades por defecto.

En cuanto a métodos numéricos, se utilizó el componente *EulerImplicitSolver* para resolver las ecuaciones diferenciales ordinarias mediante el método de Euler implícito, y el componente *SparseLDLSolver* para resolver sistemas de ecuaciones lineales mediante el método de Factorización LDL. Ambos componentes tienen sus propiedades por defecto.

3.1.3.2 Modelo de colisión

Para que las herramientas causen una retroalimentación háptica consistente, es de suma importancia que las colisiones entre las herramientas y los tejidos sean muy exactas, pero también es importante que su cálculo no afecte negativamente el rendimiento.

Para facilitar la detección de colisiones, se hizo una malla específica para cada herramienta que consta de un solo prisma rectangular que cubre únicamente su punta. Esta malla se cargó mediante el componente *MeshObjLoader*, se almacenó su topología con el componente *MeshTopology* y se almacenaron sus grados de libertad con el componente *MechanicalObject*, dando como resultado lo que se aprecia en Fig. 20.

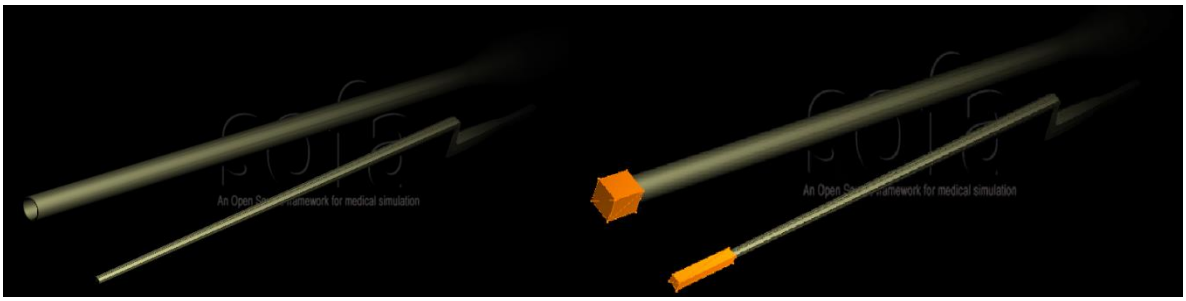

Fig. 20 Modelo visual (izquierda) y modelo de colisión (derecha) de ambas herramientas.
Los prismas anaranjados son las mallas que utiliza el modelo de colisión.
Fuente: elaboración propia.

Por otra parte, para aumentar la calidad en la detección de colisiones, se utilizaron los componentes *TriangleCollisionModel, LineCollisionModel* y *PointCollisionModel* para habilitar la detección de colisiones con cualquier cara, arista o vértice (respectivamente) de las herramientas con los triángulos de los objetos deformables. Adicionalmente, se habilitó la herramienta en la mano dominante como *trigger* para remover tetraedros con el plugin *SofaCarving*, como se explica a detalle en la sección 3.1.2.2.

Finalmente, se utilizó el componente *RigidMapping* para el mapeo entre el este modelo y el mecánico, pues está optimizado para hacer este mapeo en objetos perfectamente rígidos en SOFA.

3.1.3.3 Modelo visual

Dado que el modelo mecánico consta de un solo nodo y el mecánico de un solo prisma, en este caso sí fue necesario cargar y desplegar una malla que represente visualmente los objetos. Esto se logró con el componente *MeshObjLoader*, que almacena una malla en formato OBJ, la cual posteriormente se pasó al componente *OglModel* para desplegarla. En este mismo componente se definió un material con una componente difusa color gris, una componente ambiental negra, con reflejos especulares gris claro y un brillo leve, dando el resultado que se muestra en Fig. 20, aunque es importante resaltar que estos objetos también están expuestos a la luz que se describe al final de la sección 3.1.2.3.

Finalmente, se utilizó el componente *RigidMapping* para el mapeo entre este modelo y el mecánico, pues está optimizado para hacer este mapeo en objetos perfectamente rígidos en SOFA.

3.1.4 Incorporación de dispositivos externos

Como se discute en la sección 2.2, es deseable que los simuladores quirúrgicos involucren la mayor cantidad de sentidos posible para que sean inmersivos.

Este simulador involucra los sentidos del tacto y la vista mediante dispositivos hápticos y de realidad virtual, respectivamente. Sin embargo, SOFA no cuenta con componentes que permitan incorporar estos dispositivos de manera nativa, por lo que fue necesario recurrir a plugins y software externo. Este proceso se describe a detalle en las siguientes secciones.

3.1.4.1 Incorporación de los dispositivos hápticos en SOFA

Para incorporar un dispositivo háptico a la escena de simulación, es necesario encontrar una forma en que los movimientos hechos con estos dispositivos se mapeen a objetos en la simulación y que, al haber colisiones en la simulación, éstas se vean reflejadas como retroalimentación de fuerza en los dispositivos.

De manera nativa, SOFA contiene un plugin llamado *SofaHaptics*, que se basa en la API *OpenHaptics*, que es bastante popular para programar dispositivos hápticos. Sin embargo, también existe un plugin externo llamado *Geomagic*, que a su vez se basa en *SofaHaptics*, y permite mapear los movimientos y las fuerzas entre dispositivos hápticos de la marca 3D Systems y objetos de la escena de simulación. Se utilizó este plugin para mapear el dispositivo háptico en la mano dominante al objeto del aspirador ultrasónico y el dispositivo háptico de la mano no dominante al objeto de la pinza bipolar. En la sección 3.5.2 se describe cómo se configura cuál es la mano dominante.

Para habilitar el plugin *Geomagic*, es necesario importar a la escena los plugins *Geomagic* y *SofaHaptics*. Posteriormente, es necesario configurar el o los dispositivos

hápticos a utilizar mediante el componente *GeomagicDriver* con base en las propiedades asignadas al dispositivo durante su calibración.

Geomagic funciona con base en el algoritmo descrito en la sección 2.6.1, por lo que es necesario habilitar un solucionador de problemas de complementariedad lineal y un paso donde los objetos de la escena se puedan mover libremente, sin considerar colisiones entre ellos. Esto se logra incorporando los componentes *LCPConstraintSolver* y *FreeMotionAnimationLoop*, respectivamente. Cabe notar que, en este caso, no fue necesario modificar sus propiedades por defecto.

Posteriormente, por cada dispositivo háptico, hay que agregar un nodo de SOFA con los componentes *MechanicalObject*, para crear un nodo móvil y *MechanicalStateController* para permitir que el dispositivo externo tenga control total sobre este nodo.

Luego, por cada objeto de SOFA que será controlado por un dispositivo háptico, es necesario agregar el componente *RestShapeSpringsForceField*, que liga la posición de su objeto mecánico con la del objeto mecánico del nodo que se creó previamente mediante un esquema de acoplamiento virtual. Dado que no se puede calcular un valor exacto para la rigidez del resorte que rige el acoplamiento virtual [39], se probaron diversos valores hasta que se obtuvo la sensación háptica más consistente, resultando en un valor final de 10,000 unidades.

Por último, se agregaron los componentes *LCPForceFeedback* y *LinearSolverConstraint-Correction*. El primero sirve para incorporar retroalimentación de fuerza a la herramienta con base en el algoritmo descrito en la sección 2.6.1 y el segundo, para incorporar restricciones al sistema dinámico que rige este modelo y resolverlo mediante multiplicadores de Lagrange, como se explica en la sección 2.3.3.

Una característica importante que también se debe simular es que el aspirador ultrasónico puede remover partes del tejido, pero esta función de aspiración sólo se activa cuando el usuario lo desea.

Para lograr simular la resección de tejido, se usó el plugin *SofaCarving*, el cual permite remover tetraedros de una malla con base en el algoritmo descrito en la sección 2.4.2. Este plugin incorpora dos nuevas etiquetas al *framework*: "*CarvingSurface*" para indicar el objeto al que se le removerán tetraedros y "*CarvingTool*" para indicar el objeto que servirá para remover los tetraedros de la *CarvingSuface*.

Para utilizar el plugin *SofaCarving*, es necesario inicializarlo mediante el componente *CarvingManager* especificando al menos un parámetro llamado *carvingDistance*, que indica a qué distancia tiene que estar la *CarvingTool* de la *CarvingSurface* para removerle un tetraedro. Para simular cortes de tejido, este parámetro debe tener un valor negativo, indicando penetración entre objetos, pero en este caso, como se desea simular una aspiración, se le aplicó un valor positivo, indicando que sólo hay una cercanía entre ellos.

Una vez inicializado el plugin, se agregó la etiqueta *CarvingSurface* en el modelo de colisión de los objetos que representan los tejidos blandos y la etiqueta *CarvingTool* en el modelo de colisión del objeto que representa el aspirador ultrasónico. En todos los casos, la etiqueta se agregó sobre su *TriangleCollisionModel*, de modo que cuando un triángulo con la etiqueta *CarvingTool* se acerca a uno con la etiqueta *CarvingSurface*, el tetraedro asociado a ese triángulo desaparece.

Cabe notar que por defecto el *CarvingManager* siempre está activo, por lo que también es necesario modificarlo para que sólo se active cuando la escena de simulación reciba un evento del dispositivo háptico.

Para habilitar el envío de eventos de los dispositivos hápticos, es necesario activar la etiqueta *handleEventTriggersUpdate* del componente *MechanicalStateController* del objeto que representa el aspirador ultrasónico. Una vez hecho esto, se enviará un evento a la escena de simulación mientras se presione el botón del estilete del dispositivo háptico, activando con ello el *CarvingManager*.

3.1.4.2 Incorporación de un dispositivo de realidad virtual

Por defecto, SOFA sólo permite hacer renderizado hacia la pantalla de la computadora. Sin embargo, existe un plugin externo que permite renderizar la escena utilizando estereoscopía llamado *SofaStereo*. Este plugin crea dos cámaras en la escena viendo hacia el mismo punto, pero con una ligera separación entre ellas, simulando el enfoque de los ojos, como se explica en la sección 2.7.1. Posteriormente, el plugin renderiza lado a lado lo que está viendo cada cámara, como se aprecia en Fig. 21.

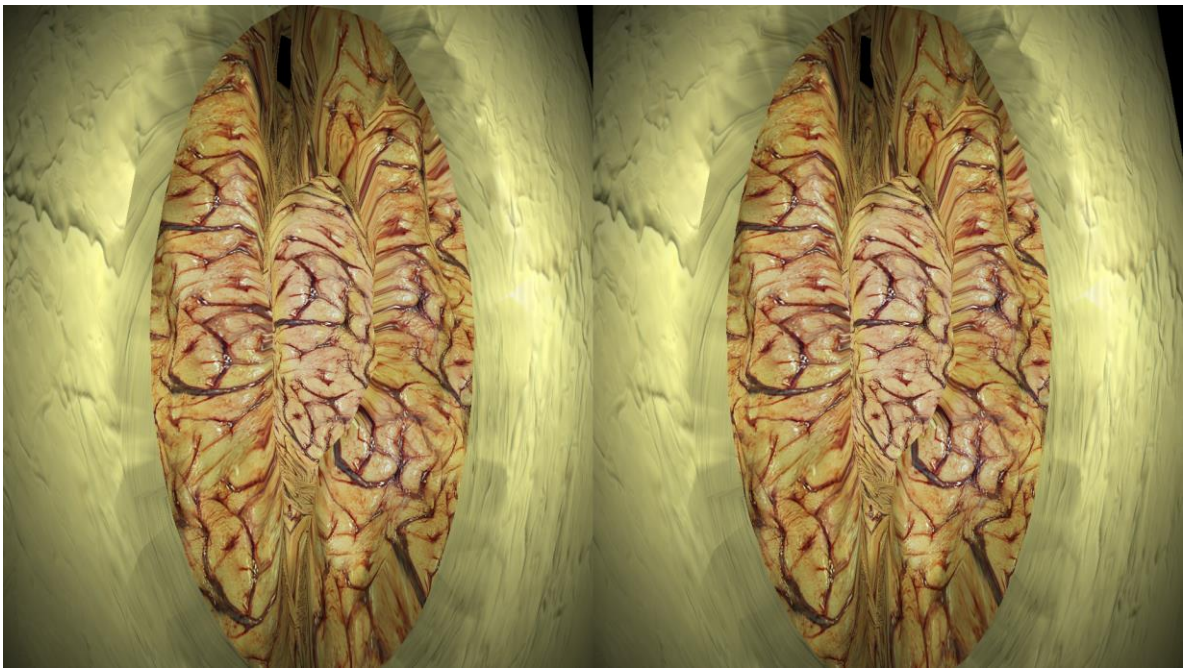

Fig. 21 Renderizado de la escena utilizando estereoscopía. Se aprecia que las imágenes se alargaron en la vertical para ajustarse al alargamiento en la horizontal que sufren al mandarse al casco de RV. Fuente: elaboración propia.

SofaStereo como tal no permite mandar datos directamente a un casco de realidad virtual, sino que renderiza directamente a la pantalla de la computadora, por lo que fue necesario recurrir a un software adicional que permitiera desplegar en el casco lo que se está viendo en pantalla de la computadora.

BigScreen es una aplicación que permite ver contenido multimedia en compañía (virtual) de otros usuarios [50]. Entre sus funcionalidades, está la de proyectar el escritorio de la computadora directamente al casco de realidad virtual.

Para poder visualizar en el casco de realidad virtual la escena de simulación que se renderiza estereoscópicamente con *SofaStereo*, se creó una sala privada de *BigScreen* donde se muestra el escritorio de la computadora sobre una pantalla virtual en todo momento. Para esta sala se activó el modo de visión estereoscópica lado a lado y se eligió un ambiente sin distracciones para que la atención del usuario esté completamente sobre la escena de simulación, dando el resultado que se aprecia en Fig. 22.

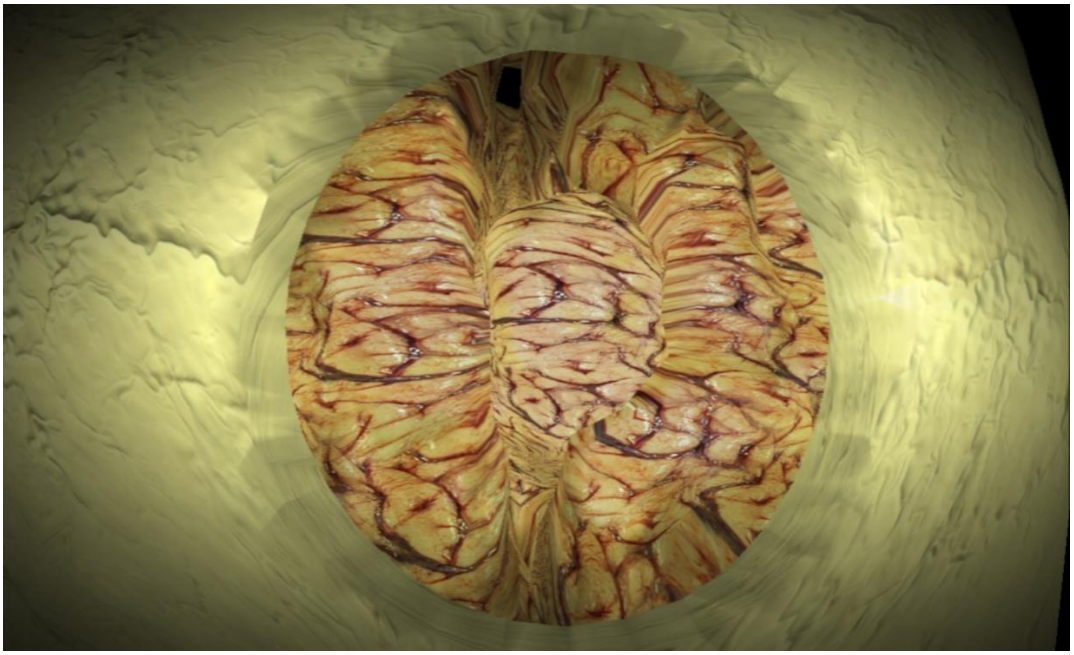

Fig. 22 Captura de la visualización de la escena de simulación en el casco de realidad virtual. Fuente: elaboración propia.

La forma en que *BigScreen* reproduce contenido estereoscópico es mandando la mitad izquierda de la pantalla de la computadora al lente izquierdo del casco y la mitad derecha al lente derecho. Sin embargo, la resolución de la mitad de la pantalla no necesariamente concuerda con la resolución de cada lente, por lo que *Bigscreen* estira cada imagen aproximadamente dos unidades en la horizontal para compensar.

Para contrarrestar un poco este efecto, se escalaron todos los modelos 0.5 unidades sobre el eje X en la escena de simulación (por eso los modelos se ven "aplastados" en Fig. 21). De esta forma, al estirar y desplegar cada render en cada lente del casco, se ve un resultado más natural, como se aprecia en Fig. 22.

3.2 Obtención de mallas 3D

En la sección 3.1 se explica a detalle cómo se desarrolló el modelo de simulación en su totalidad, con lo cual se logró replicar el comportamiento biomecánico de los tejidos blandos y su resección. Sin embargo, como se discute en la sección 2.2, otro aspecto importante a considerar en un simulador es el realismo visual de los elementos que componen la escena de simulación.

Adicionalmente, en la sección 3.1 se habla sobre cómo se cargan los modelos visuales a la escena de simulación. En esta sección se describen las técnicas y las tecnologías utilizadas para obtener estos modelos.

3.2.1 Generación de mallas de volumen a partir imágenes médicas

Como se discute en las secciones 2.3 y 2.4, existen algoritmos eficientes para simular la biomecánica de tejidos blandos y la remoción de tejido, y ambos se basan en que el volumen de la malla que los representa está discretizado mediante tetraedros. Adicionalmente, estos elementos tienen la ventaja de que permiten aproximar geometrías complejas [26, 35] (como la superficie de un cerebro) y existen algoritmos para generar mallas de tetraedros de alta calidad de manera automática, como el descrito en la sección 2.5. Por estas razones, se decidió hacer la simulación de tejidos blandos con base en mallas de tetraedros.

Para incrementar el realismo de la simulación, se optó por obtener los modelos 3D a partir de las imágenes médicas de un caso real de un paciente con un meningioma. Desgraciadamente, no se encontró un software que permita generar directamente mallas de tetraedros de calidad a partir de imágenes médicas, pero sí se encontró software que permite generar una malla de superficie a partir de una imagen médica, la cual se puede refinar y tetraedralizar posteriormente mediante otros softwares.

3D Slicer es un software libre y de código abierto que permite visualizar y analizar imágenes médicas [51], entre otras cosas. Para generar las mallas de superficie del cerebro y el tumor, se cargó a 3D Slicer un archivo DICOM con las imágenes de

resonancia magnética de un paciente que sufría de un meningioma. Posteriormente, se segmentó el cerebro y el meningioma utilizando las herramientas incluidas en el software, como son la umbralización y el crecimiento de regiones. A partir de estas segmentaciones, se creó una malla de superficie que finalmente se exportó en formato Wavefront OBJ. El resultado se aprecia en Fig. 23.

No fue posible obtener una reconstrucción del cráneo a partir de la metodología anterior, pues las imágenes de origen presentaban ruido en esa zona. En la sección 3.2.2 se explica cómo se obtuvo el modelo del cráneo.

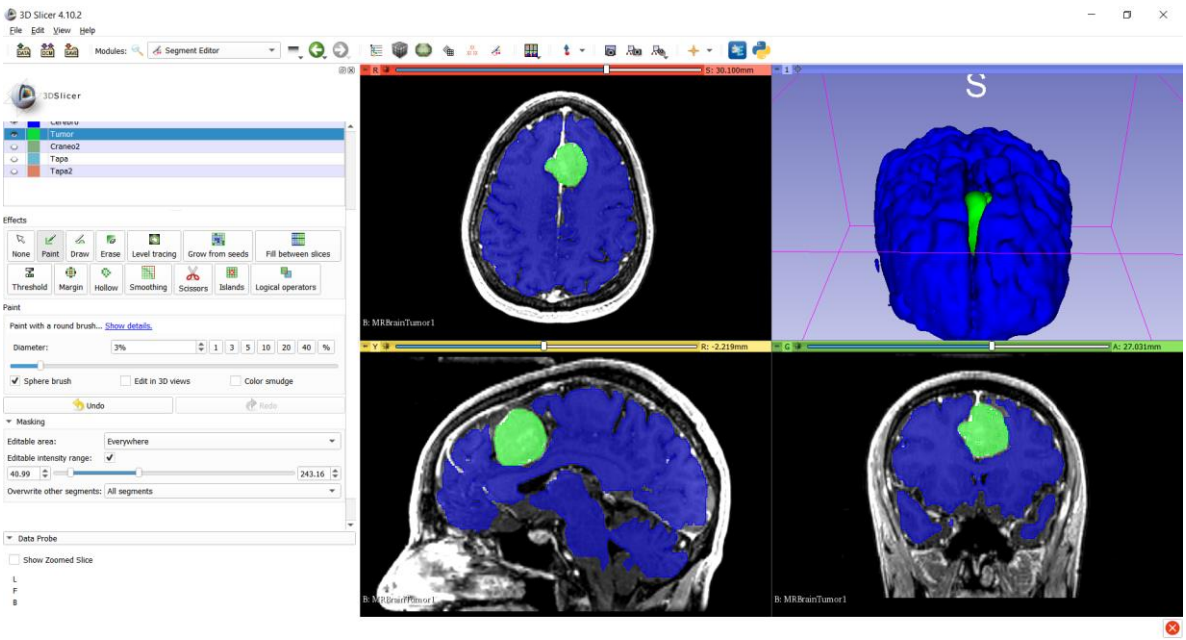

Fig. 23 Reconstrucción de los modelos de cerebro (azul) y tumor (verde) a partir de imágenes de resonancia magnética con 3D Slicer. En la esquina superior derecha se muestra el modelo reconstruido. El resto de las imágenes muestran los planos transversal, sagital y frontal de la imagen original. Fuente: elaboración propia.

3D Slicer tiende a generar mallas muy densas, ruidosas y que no necesariamente son tipo *piecewise linear complex*, lo cual imposibilitaría su tetraedralización con el algoritmo descrito en la sección 2.5.1. Por esta razón, se optó por refinar las mallas resultantes utilizando Blender: un software libre y de código abierto enfocado al desarrollo de modelos tridimensionales [52], que permite modificar una malla desde sus vértices, aristas y caras.

Para refinar la malla generada por 3D Slicer, se cargó el archivo OBJ resultante en Blender. Posteriormente, se usaron sus herramientas de malla para eliminar algunas caras internas, redundantes, separadas de la malla de interés o intersecantes, asegurar que la malla final fuera un *piecewise linear complex* sin ruido.

Como se aprecia en Fig. 1, en una cirugía real, a través del microscopio sólo es visible la parte del cerebro expuesta a través de la craneotomía. Teniendo esto en cuenta y basándose parcialmente en las ideas presentadas en [21], se optó por cortar adicionalmente una sección del cerebro en Blender, dejando únicamente la parte visible a través de la craneotomía, como se aprecia en Fig. 24. A esta sección le llamaremos "área de interés del cerebro" y corresponde a un 15.73% del volumen original del cerebro, con lo cual se obtiene un mejor rendimiento que si se simularan las deformaciones del 100% del cerebro.

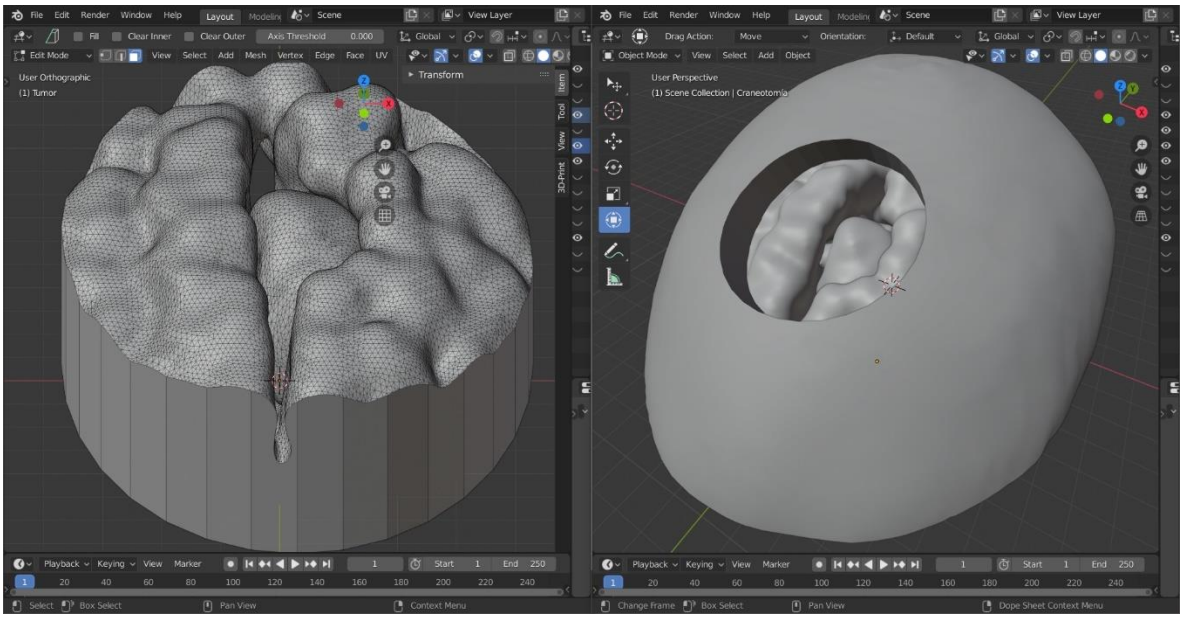

Fig. 24 Corte de las mallas del cerebro y el cráneo. Del lado izquierdo se aprecia el corte que se hizo a la malla original del cerebro, dejando únicamente el área cercana al tumor. También se aprecia la alta densidad poligonal de esta malla. Del lado derecho se aprecia la "craneotomía" hecha al modelo de la bóveda craneal, permitiendo ver el área de interés del cerebro.
Fuente: elaboración propia.

Tras los refinamientos y cortes en las mallas originales, la malla de área de interés del cerebro está conformada por 33,455 polígonos y la del tumor por 10,432 polígonos. Se

intentó tetraedralizar estas mallas, pero en algunos casos se obtuvieron mallas de más de 100,000 tetraedros, que al ser cargadas en la escena de simulación, imposibilitaban que ésta corriera en tiempo real. Con esto en mente, se exportaron nuevamente las mallas a formato Wavefront OBJ para reducir su conteo poligonal en otro software.

MeshLab es un software libre y de código abierto enfocado al procesamiento de mallas 3D [53]. El software cuenta con una amplia variedad de herramientas de edición de mallas, entre las que está "Quadric Edge Collapse Decimation", la cual sirve para reducir la cantidad de caras de una malla a una cantidad deseada.

Dado que aún no se conocía exactamente el impacto que iba a tener la reducción de caras en el desempeño de la simulación, se cargaron a MeshLab los archivos OBJ de los modelos cortados y se utilizó la herramienta Quadric Edge Collapse Decimation para crear nuevas mallas con cantidades de polígonos menores. Para el cerebro, se creó una malla con 10,000 polígonos, otra con 5,000 y otra con 2,000, y para el tumor se creó una malla con 4,000 polígonos, otra con 2,000 y otra con 1,000, como se aprecia en la Tabla 4. Todas estas mallas se exportaron como archivos PLY. En Fig. 25 se aprecia un ejemplo de estas mallas con polígonos reducidos.

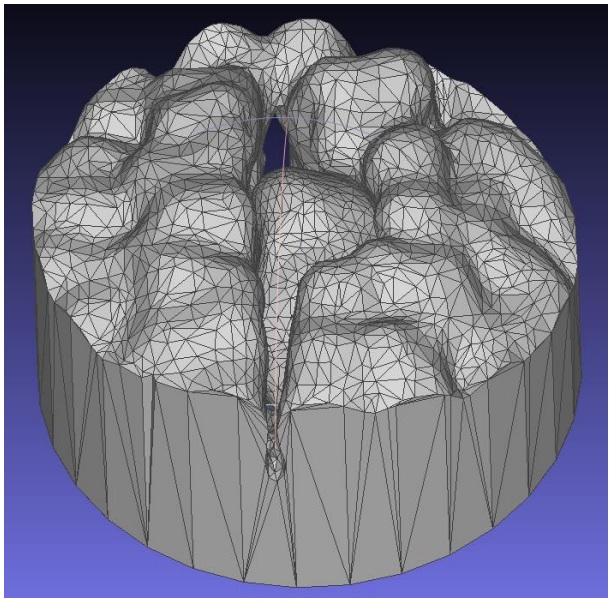

Fig. 25 Resultado de la reducción de caras de los modelos en MeshLab. La malla del área de interés del cerebro tiene 5,000 caras y la malla del tumor tiene 2,000.
Fuente: elaboración propia.

Finalmente, se tetraedralizaron las mallas de superficie con caras reducidas. Tanto Gmsh [54] como TetGen [35] son softwares libres y de código abierto hechos para este fin, pero se optó por utilizar TetGen, pues permite crear mallas de tetraedros de calidad [35] y se observó que da mejores resultados que Gmsh, aunque sólo admite mallas de superficie en formato PLY. Sin embargo, SOFA no es compatible con el formato de mallas de tetraedros que genera TetGen, por lo que se utilizó Gmsh para hacer la conversión a un formato compatible con SOFA.

Para hacer las tetraedralizaciones, se cargaron una por una las mallas de superficie con caras reducidas en formato PLY a TetGen, con lo que se obtuvieron mallas de tetraedros en formato MESH. Estos archivos se abrieron en Gmsh para poder visualizarlos (como se aprecia en Fig. 26) y convertirlos al formato MSH, que es admitido por SOFA. Cabe notar que en TetGen, la cantidad de tetraedros en la malla resultante no se puede ajustar directamente, pero básicamente, generará una malla con más tetraedros entre más caras tenga la malla de superficie de entrada. En la Tabla 4 se aprecian los resultados de estas tetraedralizaciones para todas las mallas.

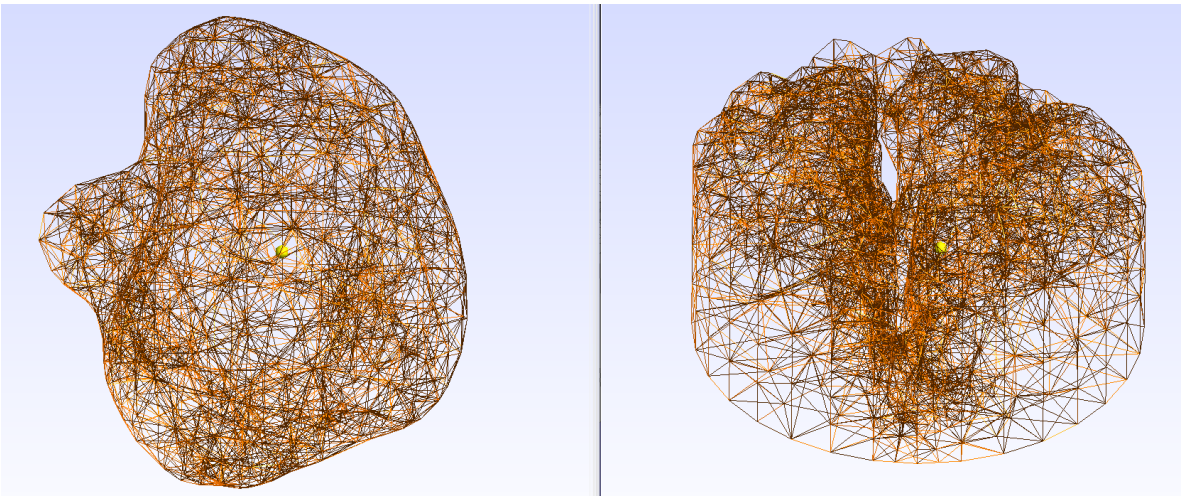

Fig. 26 Visualización en Gmsh del resultado de la tetraedralización de las mallas del tumor (izquierda) y el área de interés del cerebro (derecha). La malla del tumor está conformada por 4,665 tetraedros y la del área de interés del cerebro por 9,119 tetraedros. Ambas tetraedralizaciones se hicieron en TetGen. Fuente: elaboración propia.

Tabla 4. Cantidad de tetraedros generados por cada malla del tumor y el cerebro tras pasar por el proceso de tetraedralización de TetGen.
Fuente: ebaloración propia.

Modelo	*Densidad de elementos la malla*	*Cantidad de polígonos en la malla original*	*Cantidad de tetraedros de la malla final*
Cerebro	Alta	10,000	33,237
	Media	5,000	9,119
	Baja	2,000	5,877
Tumor	Alta	4,000	9,670
	Media	2,000	4,665
	Baja	1,000	2,172

Finalmente, se probó la escena de simulación utilizando ambas mallas con densidad de elementos alta, ambas de densidad media y ambas de densidad baja. Cuando se utilizaron las mallas de densidad alta, en algunos casos la escena no se ejecutaba en tiempo real. Por otra parte, cuando se utilizaron las mallas de densidad media y baja, en la mayoría de los casos se lograba mantener una frecuencia de actualización de superior a 60 Hz (como se describe en la sección 4.1), pero se optó por utilizar los modelos de densidad media para que la simulación biomecánica fuera más exacta y que los elementos removidos fueran más pequeños, dando una mejor sensación visual y táctil.

Cabe notar que también se probaron combinaciones de mallas de diferentes densidades, pero la sensación visual era poco favorable, pues causaba que algunos objetos se vieran mejor definidos que otros o que los segmentos de tejido removido fueran demasiado desiguales, debido a la diferencia en el tamaño de los tetraedros.

3.2.2 Modelado de mallas de superficie

Todos los objetos rígidos y los objetos meramente visuales en la escena de simulación son mallas de superficie. Adicionalmente, como se explica en las 3.1.2.1 y 3.4.2, la escena también contiene mallas que definen regiones de interés que son útiles para la

simulación, pero que los usuarios finales nunca verán. En esta sección se describe la creación de estas mallas.

3.2.2.1 Mallas visuales

Como se aprecia en Fig. 1, durante la cirugía únicamente se observan las puntas de las herramientas utilizadas, que en este caso son la pinza bipolar y el aspirador ultrasónico.

El modelo de la punta del aspirador ultrasónico se hizo desde cero en Blender, con base en las medidas del aspirador comercial SONOPET de la marca Stryker [55]. El resultado de este modelado se aprecia en Fig. 27.

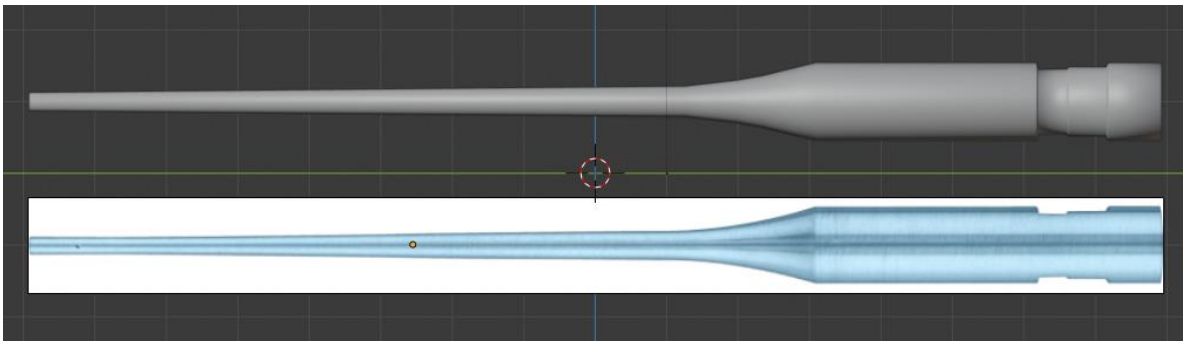

Fig. 27 Modelo de la punta aspirador ultrasónico hecho en Blender. En la parte superior de la imagen se aprecia el modelo final. En la parte inferior se encuentra una imagen de la punta del aspirador comercial SONOPET de la marca Stryker usada como referencia, tomada de [55].
Fuente: elaboración propia.

Tanto el modelo del aspirador quirúrgico como el del cráneo se obtuvieron de los archivos del simulador BACSIM [12], con el permiso de sus autores. Estos modelos se muestran en Fig. 28 y Fig. 29.

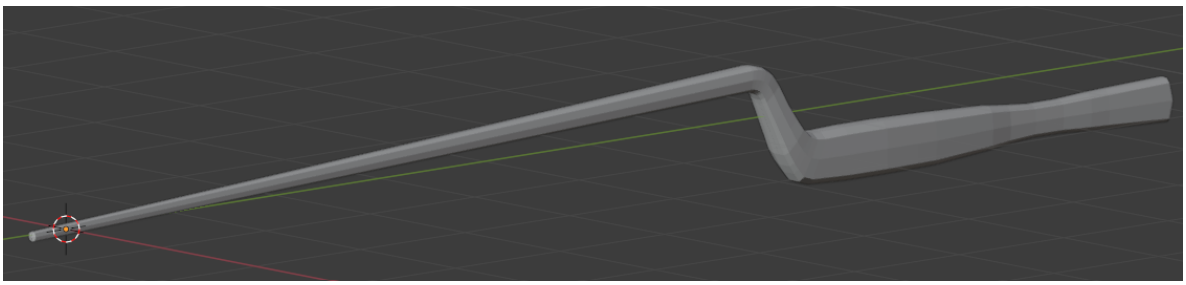

Fig. 28 Modelo de la punta aspirador ultrasónico creado para el simulador BACSIM.
Fuente: elaboración propia.

Fig. 29 Modelo del cráneo creado para el simulador BACSIM [12].
Fuente: elaboración propia.

El modelo aspirador quirúrgico se importó sin ninguna modificación al simulador. Al modelo del cráneo, por otra parte, se le hicieron modificaciones antes de importarlo para adaptarse a esta operación.

Durante la cirugía, el microscopio está fijo sobre la bóveda craneal, por lo que únicamente se aprecia esa parte del cráneo. Teniendo esto en cuenta, y con el fin de reducir la complejidad de la simulación, se hizo un corte transversal al modelo del cráneo, dejando únicamente la sección de la bóveda craneal. Posteriormente, a este modelo se le se le hizo un agujero de manera manual sobre el área del tumor, simulando una craneotomía a través de la cual se puede manipular el área de interés. El resultado de estas operaciones se aprecia en Fig. 30 y el modelo ya integrado en la escena se aprecia en Fig. 24.

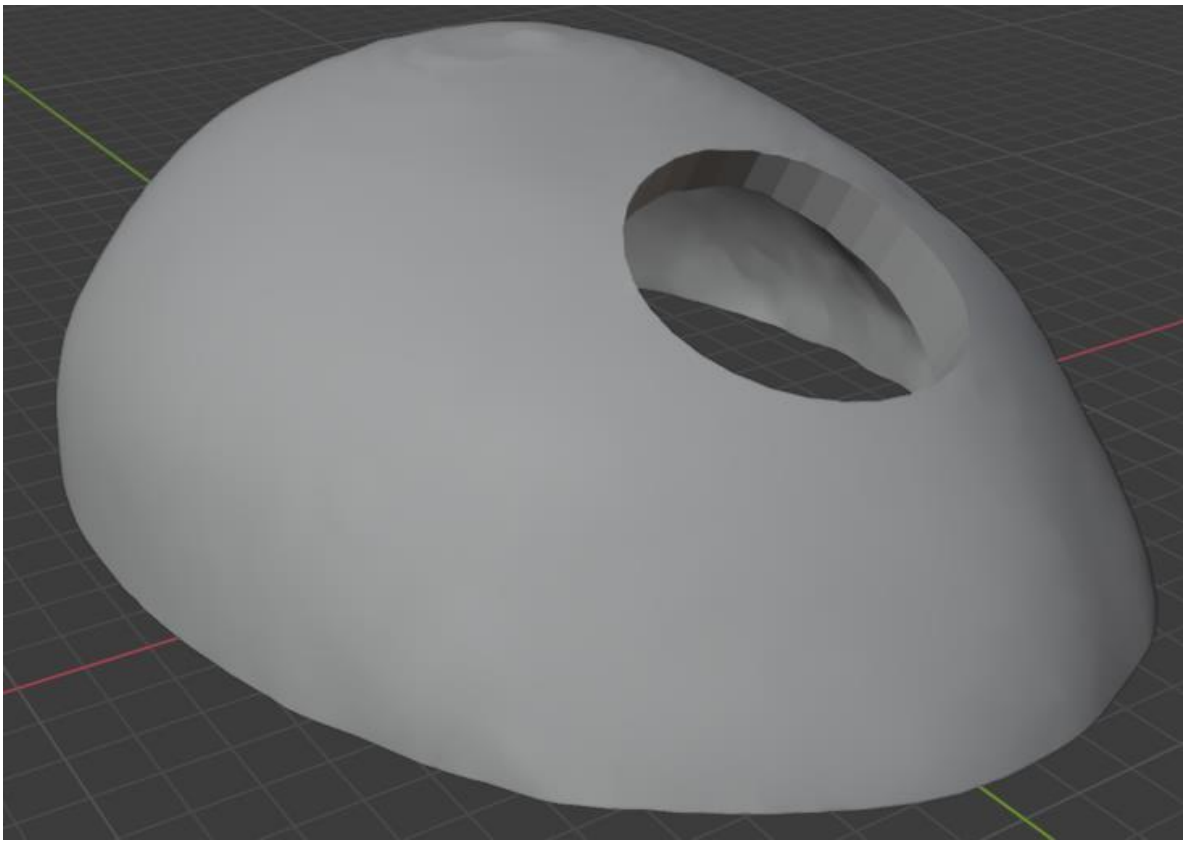

Fig. 30 Modelo del cráneo adaptado para este simulador.
Fuente: elaboración propia.

3.2.2.2 Mallas que definen regiones de interés

En las secciones 3.1.2.1 y 3.4.2 se habla sobre mallas que definen regiones de interés para los objetos simulados. Básicamente, la función de estas mallas es almacenar las referencias de los vértices que caen dentro de ellas para que posteriormente se pueda restringir su movimiento o monitorear sus desplazamientos.

Estas mallas se crearon completamente a mano en Blender, teniendo en cuenta cuáles son los vértices de interés. Para mantener el cerebro en su lugar, se crearon dos anillos en la orilla superior e inferior de la región de interés del cerebro; para mantener el tumor en su lugar, se colocó una esfera en su centro, y para monitorear los desplazamientos del cerebro, se crearon pequeños prismas que abarcan únicamente los vértices que se desea rastrear. Estas mallas se muestran en Fig. 31

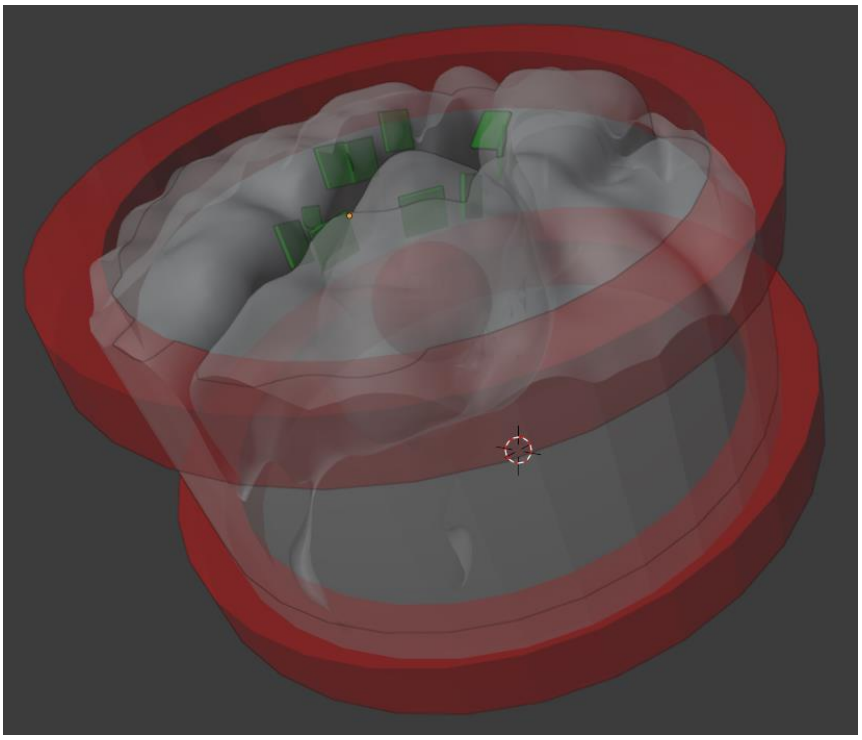

Fig. 31 Mallas que definen regiones de interés. En gris, se aprecian los modelos del cerebro y del tumor, en rojo, las mallas que definen regiones de interés para restringir movimientos de vértices y en verde las mallas para definir los vértices a monitorear.
Fuente: elaboración propia.

3.3 Construcción de la estación de trabajo

Como se explicó en la sección 2.2, un simulador híbrido es aquel que combina elementos de simulación computarizada con elementos físicos reales, brindando una sensación más realista sobre el ambiente donde se realiza la cirugía. Además, durante conversaciones con los médicos para la planeación del simulador, mencionaron que la inclusión de elementos físicos es una característica deseable.

Para lograr que este simulador sea híbrido, se optó por hacer una estación de trabajo muy similar a la creada en [12], pero un poco más portátil. La idea de esta estación de trabajo es, principalmente, que corresponda con los elementos de la escena de simulación y que se parezca al ambiente real donde se llevaría a cabo esta operación, pero también que sea relativamente fácil de transportar e instalar para poder hacer pruebas en espacios reducidos, como una oficina.

La estación de trabajo completa e instalada se aprecia en Fig. 32. Esta estación de trabajo se puede dividir en dos grandes partes: una computadora de gama alta (como la descrita en la sección 4.1) en la que se pueda ejecutar la escena de simulación y a la que se conectan los dispositivos hápticos y de realidad virtual, y una base móvil sobre la que se colocan los dispositivos hápticos y los elementos físicos de la simulación. Cabe notar que no es necesario definir un espacio para el casco de realidad virtual, pues éste debe ir sobre la cabeza del usuario, sin embargo, sí es necesario considerar el espacio para los sensores de movimiento del casco, si es que éste los necesita.

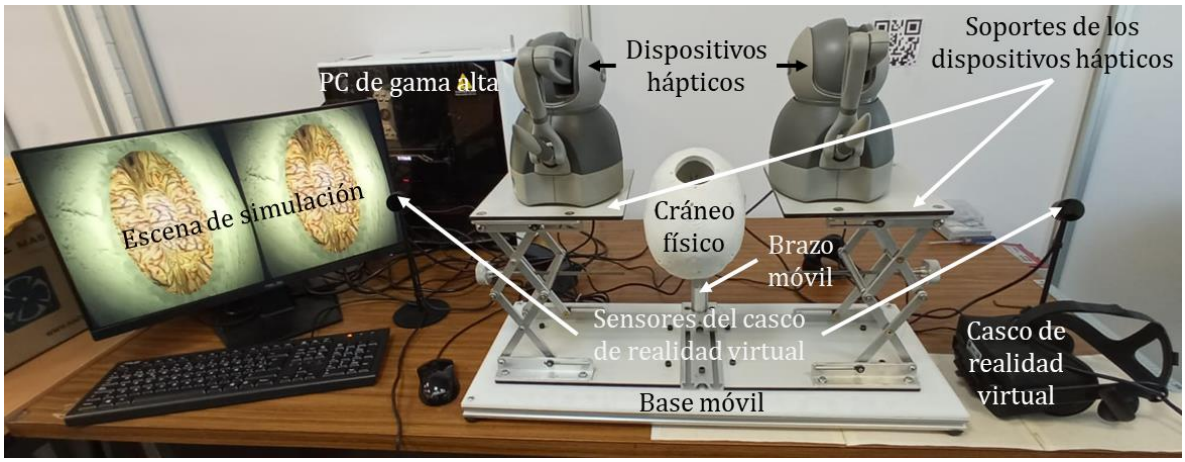

Fig. 32 Partes de la estación de trabajo completa, instalada y corriendo la escena de simulación. Fuente: elaboración propia.

En su versión final, la escena de simulación se almacena en un archivo comprimido que incluye el *framework* SOFA, por lo que debería poder descomprimirse y ejecutarse en cualquier computadora de gama alta que tenga Python 3 instalado. Sin embargo, para poder interactuar con la escena mediante dispositivos hápticos y de realidad virtual, es necesario que dicha computadora cuente con la API de *OpenHaptics*, el software para configurar los dispositivos hápticos y una versión del software *BigScreen* compatible con el casco de realidad a utilizar.

La creación de la escena de simulación se describe a detalle en la sección 3.1 y, como se mencionó anteriormente, debería poder ejecutarse sin problema en cualquier computadora de gama alta, por lo que en lo que resta de esta sección se aborda únicamente la creación y uso de las partes físicas de la estación de trabajo.

El equipo de trabajo del Laboratorio de Bioinstrumentación del ICAT colaboró en la construcción de la base móvil de la estación de trabajo utilizando componentes existentes en el laboratorio. Esta base móvil consta de una placa de acrílico, sobre la que están atornillados un brazo móvil al centro y dos soportes de precisión comerciales a los lados. Sobre el brazo móvil, se coloca un cráneo impreso en 3D con un agujero, que corresponde a la craneotomía de la escena de simulación, y sobre los soportes de precisión hay unas marcas sobre las que se deben colocar los dispositivos hápticos para que su posición sobre el cráneo corresponda con la posición de las herramientas sobre el cráneo en la escena de simulación. Estos componentes y su configuración se aprecian en Fig. 33.

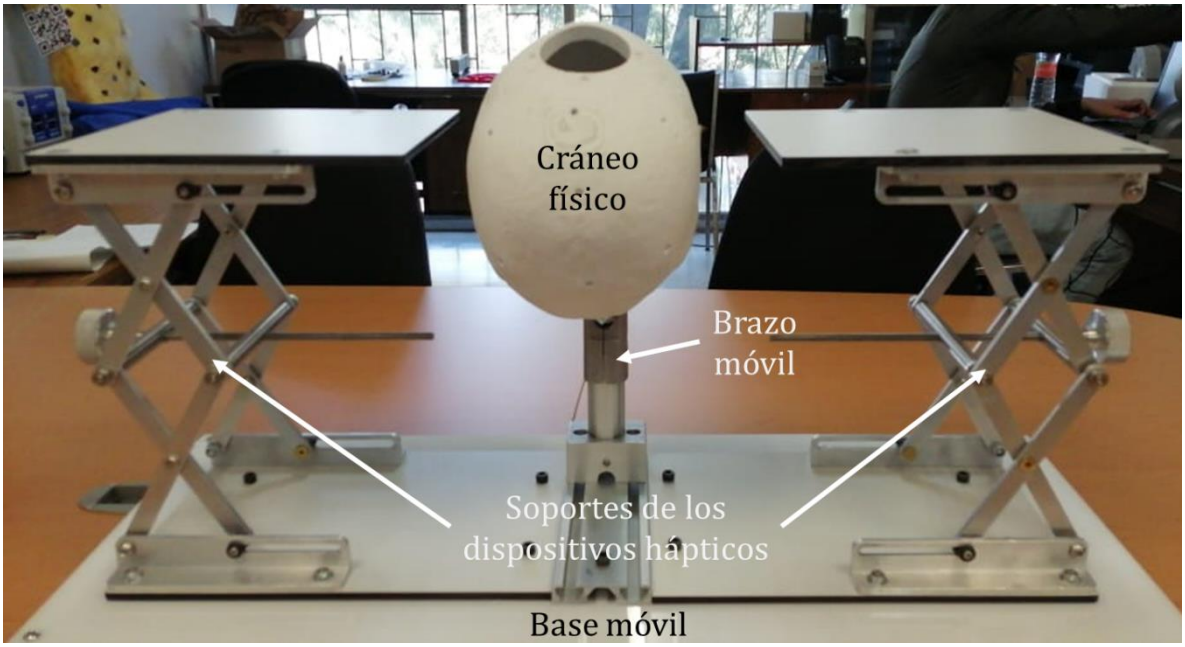

Fig. 33 Base móvil de la estación de trabajo con los elementos físicos del simulador.
Fuente: elaboración propia.

Como se mencionó anteriormente, la base móvil se construyó utilizando materiales existentes en el Laboratorio de Bioinstrumentación del ICAT, excepto el modelo físico del cráneo, que se tuvo que imprimir en 3D.

Como se explica en la sección 3.2.2, en la escena de simulación se utilizó un modelo específico de un cráneo. Para que este modelo fuera consistente en el ambiente virtual

y en el real, se optó por imprimir en 3D ese mismo modelo, sin embargo, fue necesario hacerle algunas modificaciones para que esto fuera posible.

En la sección 3.2.2 se explica que el modelo del cráneo se partió en dos partes, separando la bóveda craneal del resto del cráneo. Para hacer las modificaciones previas a la impresión 3D del modelo, se partió de este modelo ya dividido y con la craneotomía hecha, con la intención de que sea lo más parecido posible al modelo virtual y que pudiera ser impreso en dos partes.

A este modelo del cráneo se le agregó una base cerca del área de la nuca para hacerlo embonar en el brazo móvil de la estación de trabajo y se le agregaron marcas en puntos clave, con las que en un futuro se planea hacer *tracking* para hacer que la escena de simulación y la escena real sean más consistentes espacialmente. El resultado de estas modificaciones se aprecia en Fig. 34.

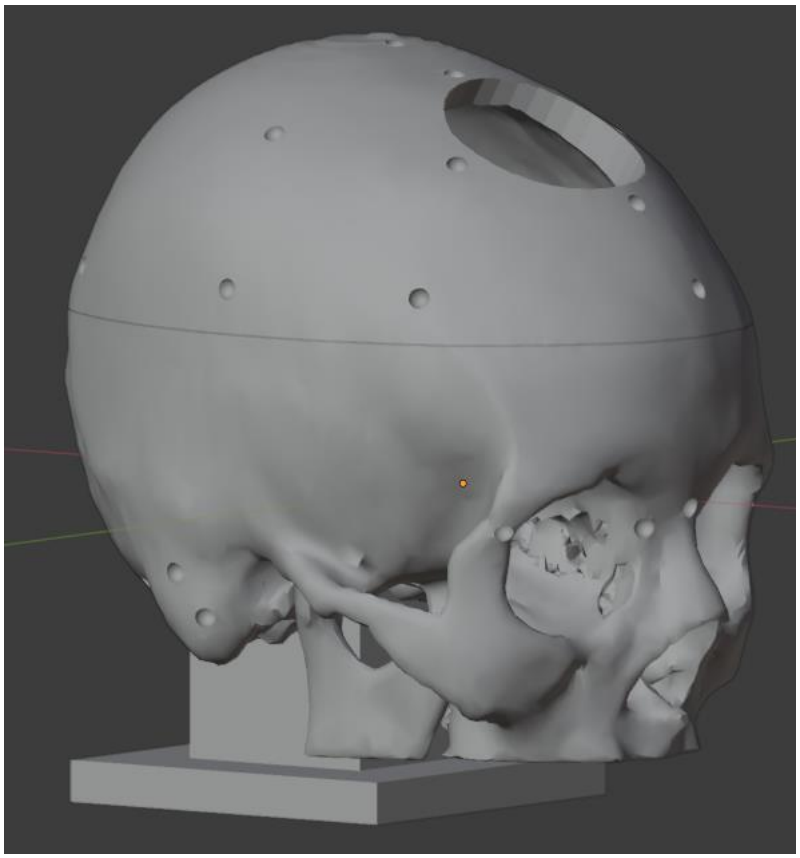

Fig. 34 Modelo final del cráneo para ser impreso en 3D.
Fuente: elaboración propia.

3.4 Generación de métricas de desempeño del usuario

Dado que se pretende que el simulador sea utilizado como herramienta de aprendizaje, es importante que los usuarios que interactúen con él puedan ser evaluados de alguna forma. En esta sección, se justifica y se explica la creación de las métricas de desempeño que se pueden obtener del simulador.

3.4.1 Obtención del volumen de tejido removido

Como se explica en [17], el objetivo de la cirugía es remover la mayor cantidad posible de tejido tumoral, removiendo la menor cantidad posible de tejido sano. Con base en esto, se optó por calcular el porcentaje de tejido tumoral y de tejido sano al terminar la simulación, similar a como lo hizo [10].

Cuando el usuario inicia su sesión de simulación el simulador arroja dos archivos tipo *Visualization Toolkit* (VTK) [56]: uno con la malla de tetraedros del tumor y otro con la malla de tetraedros del tejido cerebral sano en su posición inicial. De igual manera, cuando el usuario decide terminar la sesión, el simulador arroja los mismos archivos VTK, pero ahora con las mallas en su estado final.

Para poder calcular los porcentajes de tejido removidos, se creó un *script* de Python que, con base en los archivos VTK generados por la simulación, calcula el volumen de las mallas en su estado inicial y final, obteniendo con ello el porcentaje de tejido removido. Para ello, el programa suma el volumen de cada uno de los tetraedros que conforman las mallas y posteriormente divide el volumen total de la malla final por el de la inicial. En el Apéndice A se muestra una descripción detallada de este *script*.

3.4.2 Obtención de otras métricas a partir de monitores de SOFA

SOFA incluye un plugin llamado *SofaValidation* que a su vez incluye un componente llamado *Monitor* que permite medir y almacenar en un archivo de tipo Gnuplot [56] una serie de tiempo con la posición, la fuerza y otras métricas en ciertos puntos una malla en intervalos definidos.

Para poder obtener datos más finos sobre el desempeño del usuario, se configuraron cuatro monitores en la escena de simulación: uno de posición en la punta de cada una de las herramientas que controla el usuario, y uno de posición y otro de fuerza en 10 puntos de la malla que representa al tejido sano. Todos los monitores se actualizan con una frecuencia de 100 Hz.

Para ello, se colocaron componentes de tipo *Monitor* dentro de los modelos de colisión de cada objeto. Para las herramientas, se asignó el seguimiento a uno de los vértices en la punta de la herramienta, y para el cerebro, se configuró el monitor para que hiciera seguimiento de todos los vértices que cayeran dentro de una región de interés previamente creada (definida en la sección 3.2.2.2). En Fig. 35 se muestran los puntos sobre los que hace seguimiento el monitor para el cerebro.

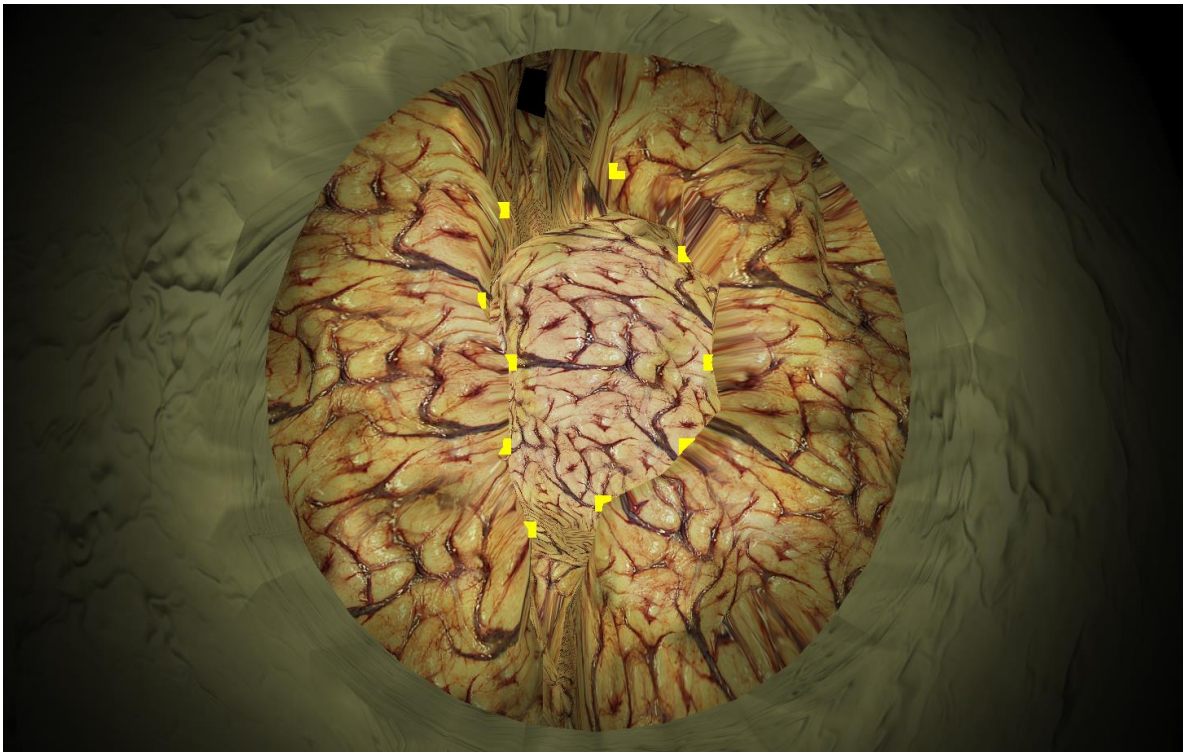

Fig. 35 Captura de pantalla de la escena de simulación. En amarillo se muestran los puntos del cerebro a monitorear, definidos a partir de una región de interés.
Fuente: elaboración propia.

La idea de los monitores en las herramientas es monitorear los movimientos de los usuarios, con lo cual se podría evaluar la destreza del practicante, midiendo

objetivamente la precisión y brusquedad de sus movimientos, su coordinación, entre otros aspectos. Por otra parte, con los monitores en el tejido sano se pretende monitorear las zonas donde se espera que haya más interacción por parte del usuario, permitiendo medir objetivamente qué tanto desplaza los tejidos o qué tanta fuerza aplica sobre ellos.

Con estos monitores se puede hacer un análisis más detallado del desempeño de cada uno de los participantes que interactuaron con el simulador, con lo cual, en un futuro, se espera poder hacer experimentos comparando el desempeño de neurocirujanos expertos e inexpertos (como se hizo en [12]) y con ello validar el simulador para que sea usado como herramienta de entrenamiento.

3.5 GUI de control del simulador

La escena de simulación consiste en un archivo con extensión ".scn" que se puede abrir en el software SOFA, sin embargo, para poder ejecutarla o modificarla, sería necesario tener conocimientos básicos de SOFA. Para hacer que la modificación y ejecución de la simulación sea más accesible, se creó un *script* de Python que despliegue una GUI hecha con Tkinter que permite hacer algunos cambios en la escena de simulación, ejecutarla, y desplegar algunos resultados de desempeño del usuario. A continuación, se describe esta GUI y su funcionamiento.

3.5.1 Descripción de la GUI

Al ejecutar el *script* se despliega la GUI que se muestra en Fig. 36. Esta GUI se divide en cuatro secciones: una sección de instrucciones, una de modificación de variables del usuario, una de calibración de la escena y una de botones de acción. En esta sección, se describe el funcionamiento de cada una de estas secciones.

La sección de instrucciones muestra instrucciones breves para manejar esta GUI, así como la descripción de la acción ligada a cada uno de sus botones.

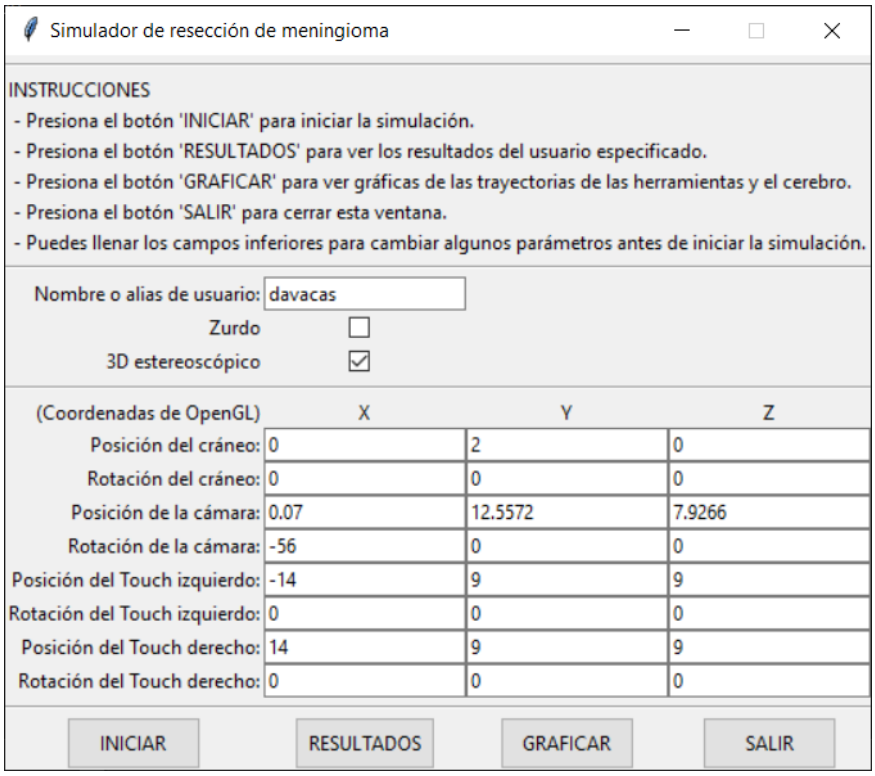

Fig. 36 Captura de pantalla de la GUI de ejecución y modificación del simulador. De arriba abajo, se aprecian las secciones de instrucciones, de modificación de variables del usuario, de calibración de la escena y de botones de acción, separadas por líneas horizontales.
Fuente: Elaboración propia.

La sección de variables de usuario permite modificar el nombre o alias del usuario actual, que sirve para crear una carpeta con ese nombre donde se guardarán sus archivos de métricas de desempeño, definidos en la sección 3.4. También, permite elegir si el usuario es zurdo, lo cual cambia el aspirador ultrasónico a la mano izquierda y la pinza bipolar a la derecha. Por último, se puede elegir si desplegar la escena en 3D estereoscópico o no; si se activa esta opción, se muestra cada imagen de estereoscopía lado a lado (como se muestra en Fig. 21), lo cual permite visualizarla de manera más realista en el casco de realidad virtual. Si no, sólo se muestra el renderizado plano, como se muestra en Fig. 10.

La sección de variables de calibración de la escena permite desplazar y rotar los modelos del cráneo con sus tejidos internos, la cámara y los dispositivos Touch en la escena de simulación. De esta forma, si fuera necesario cambiar la configuración de los

dispositivos físicos, se podría también cambiar la configuración de los modelos en la escena de simulación, sin necesidad de modificar el código fuente.

Por último, la sección de botones permite hacer ciertas acciones sobre la simulación. El botón "INICIAR" permite ejecutar la escena de simulación considerando todos los parámetros que se hayan modificado en las secciones de variables de usuario y de calibración. El botón "RESULTADOS" despliega una pequeña ventana con algunas métricas de desempeño del usuario. El botón "GRAFICAR" muestra gráficas de los desplazamientos de las herramientas y el tejido sano del cerebro. El botón "SALIR" cierra la GUI.

3.5.2 Funcionamiento de la GUI

Internamente, el *script* de Python contiene una versión incompleta del código de la escena de SOFA. Cuando el usuario presiona el botón "INICIAR", se completa el código con base en los datos de las secciones de variables de usuario y de calibración, se guarda el archivo y posteriormente se ejecuta en SOFA mediante un comando de consola.

En cuanto el usuario decide terminar la simulación o presiona el botón "RESULTADOS", el programa abre los archivos de métricas de desempeño guardados en la carpeta con el nombre o alias de usuario especificado y los procesa para calcular y desplegar su tiempo de simulación y sus porcentajes de tumor y de tejido sano removidos, como se describe en la sección 3.4.1. Estos resultados se pueden ver una pequeña ventana adicional, como se muestra en Fig. 37

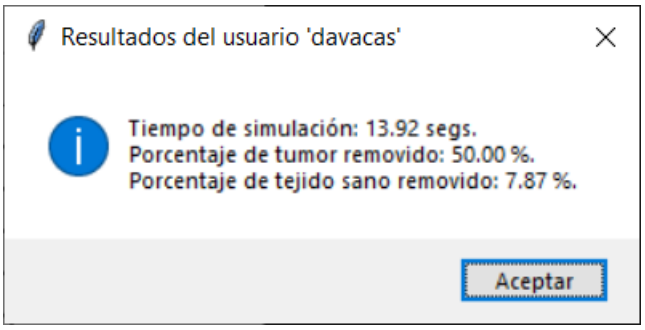

Fig. 37 Captura de pantalla de la ventana de resultados generada por la GUI.
Fuente: elaboración propia.

Finalmente, al presionar el botón "GRAFICAR", el programa lee los archivos de desplazamientos del usuario en cuestión (cuya generación se describe en la sección 3.4.2) y crea gráficas 3D interactivas con el módulo PyPlot de Python, donde se pueden observar estos desplazamientos, como se aprecia en Fig. 38.

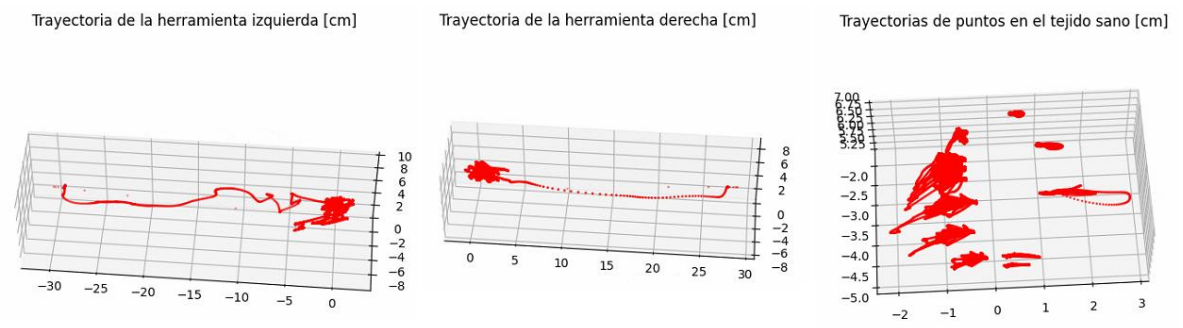

Fig. 38 Gráficas de desempeño generadas por el simulador. De izquierda a derecha las gráficas muestran: desplazamientos de la herramienta izquierda, desplazamientos de la herramienta derecha y desplazamientos de los 10 puntos colocados en el cerebro. Todas las unidades están en centímetros. Los ejes representan distancias de acuerdo con el sistema de coordenadas de OpenGL.
Fuente: elaboración propia.

Cabe notar que estos mismos archivos de desplazamientos están disponibles en la carpeta del usuario en cuestión y se pueden analizar por separado para obtener otro tipo de métricas como velocidades y aceleraciones.

4 Pruebas y resultados

Para probar si el simulador es una herramienta de aprendizaje útil, se obtuvieron diversas medidas de su desempeño, las cuales se agrupan en dos categorías: cuantitativas, para probar que el simulador cumple con los requisitos técnicos que se esperan de un simulador quirúrgico, y cualitativas, para probar que el simulador es "realista" con base en la opinión de un experto.

En esta sección, se describen a detalle los experimentos que se llevaron a cabo para obtener medidas cuantitativas y cualitativas, así como los resultados obtenidos.

4.1 Desempeño cuantitativo del simulador

Se hizo un análisis del desempeño del simulador en una computadora con una CPU Intel® Core™ i7-10700K (8 núcleos físicos a 3.8 GHz), 48 GB de RAM y una tarjeta gráfica NVIDIA® TITAN RTX™.

El análisis consistió en probar el simulador haciendo cuatro tipos de interacción diferentes:

1. sin ninguna interacción del usuario (interacción nula),
2. moviendo las herramientas sin tocar los tejidos virtuales (interacción leve),
3. utilizando las herramientas para desplazar los tejidos (interacción moderada),
4. y desplazando y removiendo partes del tejido con las herramientas (interacción intensa).

Por cada uno de estos cuatro tipos de interacción, se obtuvo el tiempo de actualización mínimo, máximo y medio con desviación estándar del hilo visual y del hilo háptico durante 1000 fotogramas (~16 segundos de simulación). Para ello, se utilizó el componente *AdvancedTimer* de SOFA, el cual calcula y muestra estas estadísticas en intervalos de fotogramas definidos por el usuario. En la Tabla 5 se muestran estos resultados.

Tabla 5. Estadísticas del tiempo de actualización del hilo visual y háptico durante 1000 fotogramas con interacción del usuario nula, leve, moderada e intensa.
Fuente: elaboración propia.

Tipo de interacción	*Hilo*	*Tiempo de actualización mínimo [ms]*	*Tiempo de actualización máximo [ms]*	*Tiempo de actualización medio ± desviación estándar [ms]*	*Frecuencia de actualización media [Hz]*
Nula	Visual	12.25	14.26	12.79 ± 0.31	78.1861
	Háptico	0.17	0.29	0.18 ± 0.02	5,555.5556
Leve	Visual	12.74	17.76	13.69 ± 0.90	73.046
	Háptico	0.17	0.65	0.24 ± 0.07	4,166.6667
Moderada	Visual	12.91	15.21	13.78 ± 0.46	72.5689
	Háptico	0.17	1.61	0.59 ± 0.42	1,694.9152
Intensa	Visual	13.00	17.12	14.35 ± 0.58	69.6864
	Háptico	0.21	1.83	0.96 ± 0.28	1,041.6667

4.2 Desempeño cualitativo del simulador

Con el fin de obtener retroalimentación cuantitativa del simulador, el Dr. Isaac Enrique Tello Mata, neurocirujano del Instituto Nacional de Neurología y Neurocirugía con experiencia en la operación de resección de meningiomas, probó el simulador funcionando en su totalidad, como se aprecia en Fig. 39.

Fig. 39 Imágenes del Dr. Isaac Enrique Tello Mata probando el simulador. Se aprecia que tiene colocado el casco de realidad virtual y está manipulando los dispositivos hápticos. En la pantalla de la computadora se puede ver lo que está viendo el neurocirujano en el casco.
Fuente: elaboración propia.

Para medir la experiencia de los usuarios con el simulador, se adaptó el cuestionario de opinión hecho en [12] donde se mide su facilidad de uso, la dificultad de la tarea y el realismo de la escena, cambiando los reactivos por unos más pertinentes con el simulador en cuestión.

En el cuestionario, por cada uno de sus reactivos, el participante puede seleccionar mediante una escala tipo Likert qué tan de acuerdo o en desacuerdo está con él, siendo 1 que está totalmente en desacuerdo, 5 que está totalmente de acuerdo y 3 es neutral. Adicionalmente, el cuestionario cuenta con un breve consentimiento informado y un apartado para escribir comentarios de forma libre. Este cuestionario está disponible en su totalidad en el Apéndice B.

Tras tres sesiones de simulación, se le pidió al neurocirujano que contestara el cuestionario, afirmando por escrito que respondió únicamente con base en su experiencia, sin influencia de nadie más y que permite que sus respuestas se utilicen para fines de investigación. En la Tabla 6 se aprecian los resultados obtenidos.

Adicionalmente, en el apartado de comentarios el neurocirujano escribió: “El tejido cerebral pudiera ser de consistencia más rígida en lo visual y en lo háptico. Sería útil tener el tumor más adherido al cerebro para poderlo disecar. Sería útil tener la pinza de bipolar abierta, y poderla manipular para “tomar” el tejido, y separar el tejido sano o el tumor. En general, excelente ejercicio.”.

Tabla 6. Respuestas del neurocirujano al cuestionario de evaluación del simulador.
Por cada reactivo, se muestra la categoría a la que pertenece y la puntuación que le otorgó el neurocirujano de acuerdo con su experiencia con el simulador.
En la última columna, se muestra la media de puntuaciones de cada categoría.

Categoría	*Reactivo*	*Puntuación del reactivo*	*Media de puntuaciones de la categoría*
Facilidad de uso	El movimiento de las herramientas es consistente con mis acciones.	5	4.66
	La manipulación de las herramientas es intuitiva.	5	
	Es fácil colocar la herramienta en el lugar correcto.	5	
	La retroalimentación de fuerza es útil al tocar los tejidos.	4	
	La interacción con los tejidos blandos es intuitiva.	5	
	El movimiento de las herramientas virtuales es preciso.	4	
Dificultad de la tarea	La tarea de resección es exacta.	4	4.5
	Fue sencillo remover el tumor con el aspirador ultrasónico.	5	
	Fue fácil evitar el tejido sano para remover el tumor.	4	
	Remover los tejidos blandos es sencillo.	5	
Realismo	La imagen proyectada es realista.	5	4.25
	Cuando interactúo con los tejidos blandos, éstos se comportan como esperaría.	4	
	La sensación táctil del tejido sano era suficientemente realista.	4	
	La sensación táctil del tumor era suficientemente realista	4	

5 Discusión

Evaluar el desempeño de un simulador puede ser complicado porque actualmente no hay métodos para medir de manera precisa el comportamiento de los tejidos [47, 20]. Adicionalmente, con los métodos numéricos y el hardware actual es complicado obtener una simulación de tejidos completamente precisa, por lo que en un simulador quirúrgico debe ser más importante que la simulación sea rápida, robusta y satisfactoria [20]. Sin embargo, sí existen algunas métricas bien definidas que se deben cumplir y algunas pruebas un poco más subjetivas que se pueden hacer.

En el capítulo 4 se muestran las pruebas y análisis cualitativos y cuantitativos que se hicieron sobre el desempeño del simulador. Si bien, estos análisis carecen del suficiente número de pruebas y del diseño experimental para considerarse una validación del simulador como herramienta de aprendizaje, sí nos ayudan a exponer sus fortalezas, debilidades y áreas de oportunidad, las cuales se discuten a continuación con base en los resultados obtenidos.

En general, en la Tabla 5 (sección 4.1) se puede ver que entre más intensa sea la interacción del usuario con el simulador, más decae su desempeño. Esto es de esperarse, pues al haber más interacción del usuario, es necesario calcular más fuerzas de retroalimentación háptica, más deformaciones de los tejidos, y con ello, reestructurar y redibujar la malla que los representa más seguido. Sin embargo, a pesar de estas caídas de desempeño, el simulador cumple con los aspectos descritos en la literatura para ser considerado un simulador quirúrgico en tiempo real.

Específicamente, se logró que el simulador tenga una frecuencia de actualización de fotogramas de 78.1861 Hz para las pruebas menos intensivas (sin movimiento de los dispositivos hápticos) y de 69.6864 Hz para las más intensivas (utilizando las herramientas para desplazar y remover tejidos), con caídas máximas de hasta 56.3063 Hz. Es decir, en todos los casos, la frecuencia de actualización de fotogramas es superior a 30 Hz, que es el estándar comúnmente definido en la literatura [25, 7] y utilizado en simuladores similares [10, 12].

En cuanto al hilo háptico, la frecuencia de actualización en promedio es de 5,555.5556 Hz para las pruebas menos intensivas y de 1,041.6667 Hz para las más intensivas, con caídas máximas de hasta 546.4481 Hz. Esto significa que, en promedio, la frecuencia de actualización del hilo háptico es superior a 1 kHz, que es el estándar comúnmente definido en la literatura [25, 39, 38] y utilizado en simuladores similares [10, 12].

Estos casos donde la frecuencia de actualización del hilo háptico es menor a 1 kHz sólo se dan cuando se están desplazando y/o removiendo los tejidos, es decir, cuando se tienen que obtener fuerzas y calcular desplazamientos para los nodos de las mallas mientras ésta cambia su forma de manera dinámica. Esto sugiere que, de usar mallas con una menor cantidad de tetraedros (y por ende con menos nodos) o un hardware más poderoso, la frecuencia de actualización del hilo háptico se podría mantener por arriba de 1 kHz en todo momento. No obstante, esto implicaría una menor precisión en la tarea de resección de tejido, así como un menor realismo visual y biomecánico, por lo que tendría que encontrarse un punto medio donde todos estos aspectos mantengan calidad.

Por el lado cualitativo, en la Tabla 6 (sección 4.2) se aprecia que el neurocirujano considera que el simulador es un “excelente ejercicio”, otorgándole una puntuación de 5 (la máxima posible) a 7 de los 14 reactivos y una puntuación de 4 a los 7 reactivos restantes.

Específicamente, el neurocirujano evaluó con un promedio de 4.66 a la categoría de facilidad de uso del simulador, otorgándole una puntuación de 5 a los reactivos “El movimiento de las herramientas es consistente con mis acciones”, “La manipulación de las herramientas es intuitiva”, “Es fácil colocar la herramienta en el lugar correcto” y “La interacción con los tejidos blandos es intuitiva.”, lo cual sugiere que el acoplamiento entre las herramientas físicas y virtuales es correcto. Sin embargo, le otorgó una puntuación de 4 a los reactivos de “La retroalimentación de fuerza es útil al tocar los tejidos” y “El movimiento de las herramientas virtuales es preciso”, lo cual podría deberse a las caídas de desempeño del hilo háptico discutidas anteriormente.

En cuanto a la categoría de dificultad de la tarea, el neurocirujano la evaluó con un promedio de 4.5, otorgándole 5 puntos a los reactivos “Fue sencillo remover el tumor con el aspirador ultrasónico” y “Remover los tejidos blandos es sencillo”, sugiriendo que la remoción de tetraedros funciona como se esperaba. Por otra parte, se puntuó con 4 a los reactivos “La tarea de resección es exacta” y “Fue fácil evitar el tejido sano para remover el tumor”, para lo cual el cirujano comenta “Sería útil tener el tumor más adherido al cerebro para poderlo disecar. Sería útil tener la pinza de bipolar abierta, y poderla manipular para “tomar” el tejido, y separar el tejido sano o el tumor.”

Por último, en la categoría de realismo el simulador obtuvo una puntuación media de 4.25, con 5 puntos para los reactivos “La imagen proyectada es realista”, sugiriendo que la construcción de modelos, texturas y materiales fue satisfactoria. Por otra parte, se puntuó con 4 a los reactivos “Cuando interactúo con los tejidos blandos, éstos se comportan como esperaría”, “La sensación táctil del tejido sano era suficientemente realista” y “La sensación táctil del tumor era suficientemente realista”, que hablan sobre el realismo biomecánico y la sensación táctil. Sobre esto, el neurocirujano escribió explícitamente que “El tejido cerebral pudiera ser de consistencia más rígida en lo visual y en lo háptico”. Además, esta menor puntuación en la sensación táctil podría deberse a las caídas en el desempeño en el hilo háptico que se discutieron anteriormente.

Con base en lo anterior, se puede concluir que este simulador cumple en su gran mayoría con las cuatro características que [7] define que debe tener un simulador, pues tiene respaldo médico al haber sido descrito y probado por neurocirujanos activos; es biomecánicamente realista, pues las deformaciones de los tejidos se basan en sus propiedades elásticas reales; es inmersivo al involucrar los sentidos de la vista y el tacto; y permite deformar y remover tejidos. Sin embargo, de acuerdo con la evaluación del neurocirujano, es necesario mejorar algunos aspectos del comportamiento biomecánico del cerebro para dar una sensación visual y háptica más convincente.

Por otra parte, como se discutió anteriormente, el simulador también cumple con los puntos relacionados a la ejecución en tiempo real definidos por [20] y [21], pues el hilo visual siempre se actualiza a más de 30 Hz y el háptico a más de 1 kHz en promedio.

Dado lo anterior, se concluye que el simulador ya está listo para ser utilizado. Sin embargo, aún es necesario acatar las observaciones del neurocirujano (que son relativamente menores) para poder empezar a hacer experimentos de validación de constructo, contenido y criterio del simulador. Durante estos experimentos serían de gran utilidad las herramientas de obtención de métricas de desempeño del usuario (definidas en la sección 3.4) y con ellos finalmente se definiría si utilizar este simulador realmente ayuda a los neurocirujanos a mejorar su desempeño en la cirugía real de resección de meningiomas.

6 Conclusiones y trabajo a futuro

Se desarrolló un simulador de la operación de resección de meningiomas utilizando como base el *framework* SOFA el cual incorpora una serie de algoritmos y métodos numéricos enfocados en hacer simulación biomecánica en tiempo real. Este simulador es híbrido porque involucra elementos físicos con elementos virtuales y es inmersivo, pues involucra los sentidos de la vista mediante dispositivos de realidad virtual y del tacto mediante dispositivos hápticos. Además, el simulador cumple con las características que se esperan de un simulador quirúrgico de acuerdo con la literatura, como la validez visual y biomecánica, y la ejecución en tiempo real. También, cuenta con una GUI de control que facilita a los usuarios su uso, permitiendo modificar algunos parámetros de la escena de simulación previo a su ejecución y ver los resultados del desempeño de los usuarios. Dado lo anterior, se concluye que el simulador ya está listo para ser utilizado y que representa una base sólida sobre la que se pueden seguir agregando mejoras, o sobre la que se podrían construir otros simuladores.

Un neurocirujano experto valoró el simulador y consideró que es un "excelente ejercicio" de aprendizaje, aunque dio algunas observaciones menores relacionadas con el realismo visual y táctil de los tejidos, así como de la operación en general.

Como trabajo a futuro se espera poder acatar estas observaciones y posteriormente iniciar una fase exhaustiva de experimentación con un grupo mayor de neurocirujanos, donde se compruebe su validez de constructo, contenido y criterio, con lo cual se definirá si su uso realmente ayuda a las personas a mejorar su desempeño en la cirugía de resección de meningiomas.

Por otra parte, existen varias mejoras que se podrían agregar en un futuro y que harían al simulador más inmersivo y consistente con la operación real. Por ejemplo, se podría agregar un pedal que active la remoción de tejido del aspirador ultrasónico y que reproduzca un sonido de aspiración realista, para hacer que su funcionamiento sea lo más parecido al de la herramienta real, o se podría agregar un sistema de *tracking* para

que la calibración del sistema sea semiautomatizada y no dependa del correcto posicionamiento de los dispositivos en la estación de trabajo.

Más a largo plazo, esta escena de simulación se podría agregar como un módulo al simulador BACSIM, con lo cual se podrían incorporar otros escenarios de simulación, por ejemplo, donde el paciente sufre sangrado y el usuario debe detenerlo, o escenarios previos al procedimiento de resección, por ejemplo, donde existan diferentes tipos de tumor y se deba posicionar al paciente realizarle la craneotomía de acuerdo con su ubicación, de modo que se pueda simular el procedimiento completo de resección de diferentes tipos de tumores.

Apéndice

A. Cálculo del volumen de una malla de tetraedros

Una de las métricas de desempaño del usuario implementadas en el simulador fue el cálculo del porcentaje de tumor removido. Para ello se realizó el *script* en lenguaje Python 3 que se describe detalladamente a continuación.

Primero, se importa la biblioteca NumPy para facilitar el manejo de vectores y operaciones con ellos.

```
import numpy as np
```

Una forma de conocer el volumen de una malla de tetraedros es sumar el volumen de cada uno de los tetraedros que la componen. Se puede obtener el volumen v_i de un tetraedro a partir de los 4 vértices A, B, C y D que lo conforman si se asume que $\overrightarrow{AB}, \overrightarrow{AC}$ y $\overrightarrow{AD}$ son vectores que definen las aristas (concurrentes en el vértice A) de un paralelepípedo. Posteriormente, el volumen del tetraedro se puede obtener como la sexta parte del volumen de dicho paralelepípedo mediante la ecuación:

$$v_i = \left|\frac{1}{6}\left(\left(\overrightarrow{AB} \times \overrightarrow{AC}\right) \cdot \overrightarrow{AD}\right)\right|. \tag{32}$$

Este cálculo se implementa en la siguiente función, que recibe cuatro vértices `v1`, `v2`, `v3`, `v4` y regresa el volumen del tetraedro formado por ellos.

```
def calcularVolumenTetraedro(v1, v2, v3, v4):
    v12 = v2 - v1
    v13 = v3 - v1
    v14 = v4 - v1
    return (abs(np.dot(np.cross(v12, v13), v14)) / 6)
```

La siguiente función recibe el nombre de un archivo VTK y regresa: una lista de vértices en formato de NumPy y una lista con los índices de los cuatro vértices que conforman cada uno de los tetraedros de la malla. Para ello, se vuelca todo el archivo en una lista donde cada elemento es una línea de éste. A partir de esta lista se crean dos listas: una

que contiene las líneas que con los vértices en el archivo otra que contiene las líneas con los índices. Posteriormente, se convierten estas listas de cadenas en listas de vértices e índices numéricos, y finalmente, se regresan ambas listas.

```
def obtenerVerticesIndicesVTK(nombre_archivo):
    archivo = open(nombre_archivo, "r")
    lineas = archivo.readlines()
    archivo.close()

    vertices_np = np.empty((0, 3))
    indices_np = []

    for indice, linea in enumerate(lineas):
        if linea.startswith("POINTS"):
            palabras = linea.split()
            inicio_verts = indice + 1
            fin_verts = inicio_verts + int(palabras[1])
        elif linea.startswith("CELLS"):
            palabras = linea.split()
            inicio_inds = indice + 1
            fin_inds = inicio_inds + int(palabras[1])

    vertices = lineas[inicio_verts:fin_verts]
    indices = lineas[inicio_inds:fin_inds]

    for vertice in vertices:
        x, y, z = vertice.split()
        vertices_np = np.append(vertices_np, np.array([[float(x), float(y),
        float(z)]]), axis=0)

    for tet in indices:
        s, a, b, c, d = tet.split()
        indices_np.append([int(a), int(b), int(c), int(d)])

    return vertices_np, indices_np
```

Una forma de obtener el volumen de una malla de tetraedros es sumando el volumen de cada uno de los tetraedros individuales que la componen. La siguiente función recibe el nombre de un archivo VTK, obtiene sus vértices e índices mediante la función `obtenerVerticesIndicesVTK` y posteriormente utiliza la función `calcularVolumenTetraedro` sobre cada uno de los tetraedros que contiene el archivo para calcular y sumar el área de cada uno de ellos. Finalmente regresa el volumen del modelo contenido en el archivo.

```
def calcularVolumenVTK(nombre_archivo):
    verts, indices = obtenerVerticesIndicesVTK(nombre_archivo)

    volumen = 0
    for i in indices:
        volumen += calcularVolumenTetraedro(verts[i[0]], verts[i[1]],
        verts[i[2]], verts[i[3]])

    return volumen
```

Para obtener el volumen de tejido removido, simplemente se utiliza la ecuación

$$\%_{removido} = \left(1 - \frac{v_f}{v_i}\right) \times 100, \tag{33}$$

donde v_i es el volumen de la malla sin modificaciones (calculado con este mismo programa) y v_f es el volumen de la malla al terminar la simulación. Dicha ecuación se implementa con la siguiente función, que recibe los parámetros `vi` y `vf` y regresa el porcentaje de tumor removido.

```
def calcularPorcentajeVolumenRemovido(vi, vf):
    return (1 - (vf / vi)) * 100
```

Con las 4 funciones definidas anteriormente, es posible calcular el área del modelo contenido en un archivo VTK. De hecho, así se determinó que el volumen inicial de la malla del tumor es de $\sim 14.93\ u^3$ y el de la malla del tejido cerebral sano es $\sim 168.3\ u^3$.

B. Cuestionario de evaluación cuantitativa del simulador

Por favor, por cada reactivo, ponga una marca en la columna con el número que mejor representa su experiencia con el simulador, donde **1** representa que está totalmente en desacuerdo y **5** que está totalmente de acuerdo con el texto del reactivo. Si tiene algún comentario adicional, por favor, escríbalo en la parte posterior de esta hoja.

Reactivo	**1**	**2**	**3**	**4**	**5**
El movimiento de las herramientas es consistente con mis acciones.					
La manipulación de las herramientas es intuitiva.					
Es fácil colocar la herramienta en el lugar correcto.					
La tarea de resección es exacta.					
Fue sencillo remover el tumor con el aspirador ultrasónico.					
La imagen proyectada es realista.					
Fue fácil evitar el tejido sano para remover el tumor.					
Cuando interactúo con los tejidos blandos, éstos se comportan como esperaría.					
Remover los tejidos blandos es sencillo.					
La retroalimentación de fuerza es útil al tocar los tejidos.					
La interacción con los tejidos blandos es intuitiva.					
El movimiento de las herramientas virtuales es preciso.					
La sensación táctil del tejido sano era suficientemente realista					
La sensación táctil del tumor era suficientemente realista					

Declaro que yo decidí mis respuestas a los reactivos con base en mi propia experiencia con el simulador, sin influencia de nadie más y permito que éstas se utilicen para fines de investigación.

_____

Nombre y firma

7 Referencias

[1] Clínica Mayo, «Tumor Cerebral,» Clínica Mayo, 27 abril 2019. [En línea]. Available: https://www.mayoclinic.org/es-es/diseases-conditions/brain-tumor/symptoms-causes/syc-20350084. [Último acceso: 14 abril 2021].

[2] Cancer Research UK, «Surgery for brain tumours,» 5 noviembre 2019. [En línea]. Available: https://www.cancerresearchuk.org/about-cancer/brain-tumours/treatment/surgery/remove-brain-tumour. [Último acceso: 2021 abril 14].

[3] M. I.-P. Huang Cobb, J. M. Taekman, A. R. Zomorodi, L. F. Gonzalez y D. A. Turner, «Simulation in Neurosurgery—A Brief Review and Commentary,» *World Neurosurgery,* vol. 89, pp. 583-586, 2016.

[4] P. E. Pelargos, D. T. Nagasawa, C. Lagman, S. Tenn, J. V. Demos, S. J. Lee, T. T. Bui, N. E. Barnette, N. S. Bhatt, N. Ung, A. Bari, N. A. Martin y I. Yang, «Utilizing virtual and augmented reality for educational and clinical enhancements in neurosurgery,» *Journal of Clinical Neuroscience,* vol. 35, pp. 1-4, 2017.

[5] EFE, «Los errores médicos causan 2.6 millones de muertes cada año: OMS,» Crónica, Ciudad de México, 2019.

[6] L. L. Leape, «The Preventability of Medical Injury,» de *Human Error in Medicine*, CRC Press, 1994, pp. 13-25.

[7] R. Riener y M. Harders, Virtual Reality in Medicine, Zúrich: Springer, 2012.

[8] S. S. Yan Tan y S. K. Sarker, «Simulation in surgery: a review,» *Scottish Medical Journal,* vol. 0, nº 0, pp. 1-6, 2011.

[9] A. J. Lungu, W. Swinkels, L. Claesen, P. Tu, J. Egger y X. Chen, «A review on the applications of virtual reality, augmented reality and mixed reality in surgical simulation: an extension to different kinds of surgery,» *Expert Review of Medical Devices,* vol. 18, nº 1, pp. 47-62, 2021.

[10] S. Delorme, D. Laroche, R. DiRaddo y R. Del Maestro, «NeuroTouch: A Physics-Based Virtual Simulator,» *Operative Neurosurgery,* vol. 71, nº 1, pp. 32-42, 2012.

[11] ImmersiveTouch, Inc., «Immersive Touch,» ImmersiveTouch, Inc., 2021. [En línea]. Available: https://www.immersivetouch.com/. [Último acceso: 10 junio 2021].

[12] S. Teodoro-Vite, J. S. Pérez-Lomelí, C. F. Domínguez-Velasco, A. Hernández-Valencia, M. Capurso-García y M. Á. Padilla-Castañeda, «A High-Fidelity Hybrid Virtual Reality Simulator of Aneurysm Clipping Repair With Brain Sylvian Fissure Exploration for Vascular Neurosurgery Training,» *Simulation in Healthcare,* vol. Publish Ahead of Print, 2020.

[13] N. Gélinas-Phaneuf, N. Choudhury, A. Al-Habib, A. Cabral, E. Nadeau, V. Mora, V. Pazos, P. Debergue, R. DiRaddo y R. F. Del Maestro, «Assessing performance in brain tumor resection using a novel virtual reality simulator,» *International Journal of Computer Assisted Radiology and Surgery,* vol. 9, pp. 1-9, 2014.

[14] F. E. Alotaibi, G. A. AlZhrani, M. A. S. Mullah, A. J. Sabbagh, H. Azarnoush, A. Winkler-Schwartz y R. F. Del Maestro, «Assessing Bimanual Performance in Brain Tumor Resection With NeuroTouch, a Virtual Reality Simulator,» *Neurosurgery,* vol. 11, nº 2, pp. 89-98, 2015.

[15] J. Gasco, T. Holbrook, A. Patel, A. Smith, D. Paulson, A. Muns, S. Desai, M. Moisi, Y.-F. Kuo, B. Macdonald, J. Ortega-Barnett y J. T. Patterson, «Neurosurgery simulation in residency training: feasibility, cost, and educational benefit,» *Neurosurgery,* vol. 73, nº 1, pp. 39-45, 2013.

[16] A. Bernardo, «Virtual Reality and Simulation in Neurosurgical Training,» *World Neurosurgery,* vol. 106, pp. 1015-1029, 2017.

[17] Columbia Neurosurgical Surgery, «Brain Tumor Surgery,» 2021. [En línea]. Available: https://www.columbianeurosurgery.org/treatments/brain-tumor-surgery/. [Último acceso: 14 abril 2021].

[18] MicroNeuroSurgeryOrg, «Brain tumor surgery: Surgical removal of brain tumor (meningioma),» 5 junio 2020. [En línea]. Available: https://www.youtube.com/watch?v=znfYs9xUP-E. [Último acceso: 1 junio 2021].

[19] A. Suri, D. P. Patra y R. K. Meena, «Simulation in neurosurgery: Past, present, and future,» *Neurology India,* vol. 64, nº 3, pp. 387-395, 2016.

[20] M. Bro-Nielsen, «Finite element modeling in surgery simulation,» *Proceedings of the IEEE,* vol. 86, nº 3, pp. 490-503, 1998.

[21] K. V. Hansen y O. V. Larsen, «Using region-of-interest based finite element modelling for brain-surgery simulation,» *Lecture Notes in Computer Science,* vol. 1496, pp. 305-316, 1998.

[22] J. Banks, J. Carson, B. Nelson y D. Nicol, Discrete-Event System Simulation, Nueva Jersey: Prentice Hall, 2001.

[23] S. C. Batterman y S. D. Batterman, «Forensic Engineering/Accident Reconstruction/Biomechanics of Injury/Philosophy, Basic Theory, and Fundamentals,» de *Encyclopedia of Forensic Sciences (Second Edition)*, Massachusetts, Academic Press, 2013, pp. 395-404.

[24] T.-N. Nguyen, M.-C. Ho Ba Tho y T.-T. Dao, «A Systematic Review of Real-Time Medical Simulations with Soft-Tissue Deformation: Computational Approaches, Interaction Devices, System Architectures, and Clinical Validations,» *Applied Bionics and Biomechanics,* 2020.

[25] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik y S. Cotin, «SOFA: A Multi-Model Framework for Interactive Physical Simulation,» de *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, Berlín, Springer Berlin Heidelberg, 2012, pp. 283-321.

[26] W. Maurel, Y. Wu, N. M. Thalmann y D. Thalmann, Biomechanical Models for Soft Tissue Simulation, Bruselas: Springer, 1998.

[27] G. Yin, Y. Li, J. Zhang y N. Jun, «Soft Tissue Modeling using Tetrahedron Finite Element Method in Surgery Simulation,» de *First International Conference on Information Science and Engineering*, Nanjing, 2009.

[28] J. F. Epperson, An Introduction to Numerical Methods and Analysis, Nueva Jersey: John Wiley & Sons, Inc., 2013.

[29] M. Rambo, «The Conjugate Gradient Method for Solving Linear Systems of Equations,» mayo 2016. [En línea]. Available: http://math.stmarys-ca.edu/wp-content/uploads/2017/07/Mike-Rambo.pdf. [Último acceso: 27 enero 2022].

[30] C. T. Kelley, Iterative Methods for Linear and Nonlinear Equations, Filadelfia: Society for Industrial and Applied Mathematics, 1995.

[31] I. Peterlík, M. Nouicer, C. Duriez, S. Cotin y A. Kheddar, «Constraint-Based Haptic Rendering of Multirate Compliant Mechanisms,» *IEEE Transactions on Haptics,* vol. 4, nº 3, pp. 175-187, 2011.

[32] B. André y H. Delingette, «Versatile Design of Changing Mesh Topologies for Surgery Simulation,» de *Biomedical Simulation - 4th International Symposium, ISBMS 2008, Proceedings*, Londres, Springer, 2008, pp. 147-156.

[33] C. Forest, H. Delingette y N. Ayache, «Removing tetrahedra from a manifold mesh,» de *Proceedings of Computer Animation 2002*, Ginebra, 2002.

[34] C. Forest, H. Delingette y N. Ayache, «Removing tetrahedra from manifold tetrahedralisation: application to real-time surgical simulation,» *Medical Image Analysis,* vol. 9, nº 2, pp. 113-122, 2005.

[35] S. Hang, «TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator,» *ACM Transactions on Mathematical Software,* vol. 41, nº 2, pp. 1-36, 2015.

[36] A. Bowyer, «Computing Dirichlet tessellations,» *The Computer Journal,* vol. 24, nº 2, pp. 162-166, 1981.

[37] D. Watson, «Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes,» *The Computer Journal,* vol. 24, nº 2, pp. 167-172, 1981.

[38] D. Wang, J. Xiao y Y. Zhang, Haptic Rendering for Simulation of Fine Manipulation, Berlín: Springer, 2014.

[39] G. Saupin, C. Duriez y S. Cotin, «Contact Model for Haptic Medical Simulations,» de *Biomedical Simulation - 4th International Symposium, ISBMS 2008, Proceedings*, Londres, Springer, 2008, pp. 157-165.

[40] 3D Systems, «Dispositivo háptico Touch™,» 2021. [En línea]. Available: https://es.3dsystems.com/haptics-devices/touch. [Último acceso: 27 abril 2021].

[41] 3D Systems, «Especificaciones técnicas del dispositivo háptico Touch™,» 2021. [En línea]. Available: https://es.3dsystems.com/haptics-devices/touch/specifications. [Último acceso: 27 abril 2021].

[42] H. E. L. Lowood, «Virtual reality,» Encyclopaedia Britannica, 13 mayo 2021. [En línea]. Available: https://www.britannica.com/technology/virtual-reality/Education-and-training. [Último acceso: 29 septiembre 2021].

[43] A. Tikkanen, «Stereoscopy,» Encyclopaedia Britannica, 27 junio 2013. [En línea]. Available: https://www.britannica.com/technology/stereoscopy. [Último acceso: 29 septiembre 2021].

[44] contributors, Wikipedia, «Oculus Rift CV1,» Wikipedia, The Free Encyclopedia, 21 julio 2021. [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Oculus_Rift_CV1#/media/File:Oculus-Rift-CV1-Headset-Front.jpg. [Último acceso: 1 octubre 2021].

[45] SOFA Consortium, «SOFA Documentation: Create your Plugin,» SOFA, 14 abril 2020. [En línea]. Available: https://www.sofa-framework.org/community/doc/. [Último acceso: 24 marzo 2021].

[46] SOFA Consortium, «SOFA Marketplace,» SOFA, 2021. [En línea]. Available: https://www.sofa-framework.org/applications/marketplace/. [Último acceso: 24 marzo 2021].

[47] F. Morin, M. Chabanas, H. Courtecuisse y Y. Payan, «Biomechanical Modeling of Brain Soft Tissues for Medical Applications,» de *Biomechanics of Living Organs*, Academic Press, 2017, pp. 127-146.

[48] P. Hartmann, A. Ramseier, F. Gudat, M. J. Mihatsch y W. Polasek, «Normal weight of the brain in adults in relation to age, sex, body height and weight,» *Pathologe,* vol. 15, nº 3, pp. 165-170, 1994.

[49] S. Jupiter, «Sanna Brain Surface Seamless Not Animated SET MP,» Second Life™ Marketplace, [En línea]. Available: https://marketplace.secondlife.com/es-ES/p/Sanna-Brain-Surface-Seamless-Not-Animated-SET-MP/15054430?lang=es-ES. [Último acceso: 7 mayo 2021].

[50] bigscreen, inc., *Bigscreen,* San Francisco, 2021.

[51] R. Kikins, S. D. Pieper y K. G. Vosburgh, «3D Slicer: A Platform for Subject-Specific Image Analysis, Visualization, and Clinical Support,» *Intraoperative Imaging Image-Guided Therapy,* vol. 3, nº 19, pp. 277-289, 2014.

[52] Blender Online Community, *Blender - a 3D modelling and rendering package,* Amsterdam: Blender Foundation, 2020.

[53] P. Cignoni, M. Callieri, M. Corsini, M. Dellapiane, F. Ganovelli y G. Ranzuglia, «MeshLab: an Open-Source Mesh Processing Tool,» de *Sixth Eurographics Italian Chapter Conference*, Pisa, The Eurographics Association, 2008, pp. 129-136.

[54] C. Geuzaine y J.-F. Remacle, «Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities,» *International Journal for Numerical Methods in Engineering,* vol. 79, nº 11, pp. 1309-1331, 2009.

[55] Stryker, «The complete guide to SONOPET®,» septiembre 2016. [En línea]. Available: https://neurosurgical.stryker.com/wp-content/uploads/2016/09/SONOPET-complete-OR-guide-brochure.pdf. [Último acceso: 1 junio 2021].

[56] T. Williams, C. Kelley y e. al., «gnuplot homepage,» The GNU Project, junio 2021. [En línea]. Available: http://www.gnuplot.info/. [Último acceso: 22 octubre 2021].

[57] Kitware, Inc., «File Formats for VTK Version 4.2,» abril 2015. [En línea]. Available: https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf. [Último acceso: 17 mayo 2021].