

UniversidadeVigo

Parametrización de modelos en el *framework* SOFA
para la simulación virtual de colonoscopias

Omar Delgado López

Trabajo de Fin de Grado
Escuela de Ingeniería de Telecomunicación
Grado en Ingeniería de Tecnologías de Telecomunicación

Tutores:
Adrián Lugilde López
Fernando Ariel Mikic Fonte

Curso 2024/2025

Índice

1	Introducción	4
1.1	Prevención y detección del cáncer colorrectal	4
1.2	El <i>framework</i> SOFA	5
2	Objetivos	6
3	Metodología y desarrollo	7
3.1	Geometría del modelo	7
3.2	Estructura de una simulación	7
3.3	Principales componentes de la simulación	8
3.4	Modelo físico	8
3.5	Modelo de colisiones	9
3.6	Modelo visual	9
4	Parametrización del modelo	10
4.1	Entorno y detección de colisiones	10
4.2	Selección de la malla	11
4.3	Selección y parametrización de los componentes	11
4.4	Escala y unidades de medida	13
4.5	Valor de los parámetros	13
5	Pruebas realizadas	14
5.1	Prueba 1: Simulación inicial	14
5.2	Prueba 2: Uso de diferentes escalas en la simulación	14
5.3	Prueba 3: Parametrización del intestino	15
5.4	Prueba 4: Ajuste de parámetros	15
5.5	Prueba 5: Sujeción del intestino	16
5.6	Prueba 6: Comparación entre una malla cerrada y una hueca	16
5.7	Prueba 7: Comparación de mallas triangulares y tetraédricas	17
5.8	Prueba 8: Esfera volumétrica	17
5.9	Prueba 9: Incorporación de modelos visuales	17
5.10	Prueba 10: Reducción de elementos de la malla	18
5.11	Prueba 11: Diferentes clases para definir la masa	18
5.12	Prueba 12: Diferentes clases para definir las fuerzas internas del intestino	19
5.13	Prueba 13: Pruebas con <i>solvers</i>	19
5.14	Prueba 14: Pruebas con diferentes primitivas de colisión	19
5.15	Prueba 15: Simulación de la gravedad y ajuste de parámetros	20
5.16	Prueba 16: Manipulación del movimiento de la esfera y adición de una cámara secundaria	20
5.17	Prueba 17: Uso del <i>BoxRoi</i> para modelar las secciones del intestino con más precisión	21

5.18 Prueba 18: Optimización con el <i>plugin</i> Multithreading	21
5.19 Prueba 19: Optimización con el <i>plugin</i> CUDA	21
6 Resultados	22
7 Conclusiones	23
A Estado del arte	27
A.1 Simulación de colonoscopias	27
A.2 Herramientas de simulación	27
A.3 Modelos existentes	28
B Sección ampliada de metodología y desarrollo	28
B.1 Geometría del modelo	28
B.2 Estructura de una simulación	29
B.3 Principales componentes de la simulación	30
B.4 Modelo físico	31
B.5 Modelo de colisiones	33
B.6 Modelo visual	34
C Sección ampliada de parametrización del modelo	35
C.1 Entorno y detección colisiones	35
C.2 Selección de la malla	36
C.3 Selección y parametrización de los componentes	37
C.4 Escala y unidades de medida	41
C.5 Valor de los parámetros	41
D Desarrollo completo de las pruebas realizadas	42
D.1 Simulación inicial	42
D.2 Uso de diferentes escalas en la simulación	46
D.3 Parametrización del intestino	49
D.4 Ajuste de parámetros	52
D.5 Sujeción del intestino	53
D.6 Comparación entre una malla cerrada y una hueca	57
D.7 Comparación con una malla triangular	60
D.8 Esfera volumétrica	62
D.9 Incorporación de modelos visuales	66
D.10 Reducción de elementos de la malla	69
D.11 Pruebas con diferentes clases del componente <i>Mass</i>	70
D.12 Pruebas con diferentes clases del componente <i>ForceField</i>	71
D.13 Pruebas con <i>solvers</i>	71
D.14 Pruebas con diferentes primitivas de colisión	73
D.15 Simulación de la gravedad y ajuste de parámetros	76
D.16 Manipulación del movimiento de la esfera y adición de una cámara secundaria	80

D.17 Uso de <i>BoxRoi</i> para modelar las secciones del intestino con más precisión	82
D.18 Pruebas de optimización: uso del <i>plugin Multithreading</i>	83
D.19 Pruebas de optimización: uso del <i>plugin CUDA</i>	84
E Formatos de archivo de las mallas	85
F Colección de clases disponibles	85
F.1 <i>AnimationLoop</i>	85
F.2 <i>Mass</i>	86
F.3 <i>CollisionPipeline</i>	86
F.4 <i>ForceField</i>	88
G Funcionamiento del modelo de colisiones	89
H Modelos de estructuras de datos	90
I Mallas utilizadas	90

1. Introducción

Desde hace décadas, las herramientas de simulación en el ámbito de la medicina se han investigado y desarrollado para su aplicación en diferentes áreas y procedimientos. Algunos ejemplos de ello son:

- La enseñanza y el aprendizaje de tratamientos médicos en el ámbito educativo, permitiendo su práctica sin riesgos humanos [1].
- Planificación preoperatoria con datos reales del paciente para la mejora en la precisión y seguridad de la cirugía [2].
- Entrenamiento de personal médico frente a la aparición de nuevas técnicas en áreas específicas [3].
- Estudio, investigación y recolección de datos de nuevos tratamientos en un entorno seguro y controlado [4].

Actualmente el uso de diferentes herramientas de simulación está ampliamente extendido y bajo continuo estudio [5] con el objetivo de mejorar la atención médica y, con ello, la salud y la seguridad del paciente. Esto puede verse reflejado en la investigación y aplicación médica de tecnologías como la realidad virtual y aumentada [6] o la simulación por *software* [7], con el objetivo de poder crear entornos realistas, precisos e interactivos enfocados a tratamientos y procedimientos específicos.

A la hora de lograr este objetivo, impera la necesidad de usar modelos de simulación que repliquen el comportamiento, las condiciones, y las sensaciones de un escenario real; y la cantidad de herramientas de simulación disponibles dificulta la selección de aquellas más adecuadas [8]. Por otro lado, a la hora de recrear una simulación se requieren diferentes habilidades como diseño 3D de modelos, cálculos físicos y análisis numérico, renderizado, detección de colisiones o integración con dispositivos hápticos; difíciles de incorporar de manera conjunta [9].

1.1. Prevención y detección del cáncer colorrectal

El cáncer de colon es el tercer tipo de cáncer más frecuente en el mundo, representando aproximadamente el 10 % de todos los casos de cáncer y afectando en la mayoría de los casos a personas mayores de 50 años. Factores relacionados con un modo de vida sedentario y una mala alimentación favorecen su aparición y se prevé un aumento del 63 % de aquí a 2040 [11].

Si se produce una detección temprana la tasa de supervivencia es del 91 %, llegando a disminuir hasta el 72 % si ha alcanzado tejidos u órganos cercanos y descendiendo hasta el 14 % cuando alcanza la metástasis, sucediendo esto en un período de aproximadamente 5 años desde su aparición [12].

La colonoscopia es un procedimiento de suma importancia permitiendo no sólo una detección temprana de este cáncer si no también de otro tipo de lesiones precancerosas o pólipos, que pueden ser extraídos para disminuir el riesgo de cáncer en el futuro [13]. Los riesgos de este procedimiento son bajos, pero una mala práctica puede llegar a causar una perforación que ponga en riesgo la vida del paciente, siendo las probabilidades de ello enormemente bajas en el presente. [14].

La importancia de este procedimiento y el futuro incremento previsto de su demanda aumentan la necesidad de disponer de un modelo preciso y realista que pueda ser usado mediante las herramientas de simulación más apropiadas para reforzar la práctica, el aprendizaje y la asistencia en el mismo.

1.2. El *framework* SOFA

Simulation Open Framework Architecture (SOFA)¹ es un *framework* gratuito de código abierto que permite la creación de simulaciones físicas e interactivas en tiempo real, enfocado en el ámbito de la biomecánica y la robótica [10]. Entre sus principales características, que le permiten distinguirse frente a otras herramientas, destacan:

- Cada simulación se estructura como un grafo acíclico directo. Esto facilita la simulación de múltiples objetos descomponiéndolos en diferentes nodos, cada uno representando de forma independiente una propiedad concreta como la masa, las fuerzas o la topología; permitiendo una enorme flexibilidad a la hora de realizar modificaciones específicas.
- Es una arquitectura basada en representaciones multmodelo, donde cada modelo del objeto representado está enfocado a realizar una función específica. Los tres modelos principales son el modelo físico, de colisiones y visual que, respectivamente, definen las propiedades físicas del objeto, su comportamiento ante interacciones con otros componentes de la simulación y su aspecto visual y renderizado. Cada modelo puede ser diseñado de manera independiente y se sincronizan en tiempo real durante la simulación mediante un mecanismo llamado *Mapping*, que permite propagar las fuerzas y los desplazamientos entre los modelos.
- Es de código abierto, lo que fomenta la colaboración entre investigadores y desarrolladores a la hora de aportar nuevos modelos, algoritmos y *plugins* que mejoren el *framework*. También redistribuye la carga de trabajo de manera que cada especialista pueda enfocarse exclusivamente en su área de interés.
- Tiene una enorme base de *plugins* en constante desarrollo que aportan funcionalidades de gran interés como la integración de dispositivos hápticos o el procesamiento multihilo.

¹sofa-framework.org

Actualmente, surge la necesidad de lograr obtener un escenario de simulación de un intestino grueso correctamente parametrizado, que permita trabajar con un modelo preciso y realista o, al menos, sirva como modelo mínimo viable para el desarrollo del mismo.

2. Objetivos

El principal objetivo a alcanzar en este proyecto será la simulación de un modelo de intestino grueso que sea preciso, computacionalmente eficiente y efectivo; que sirva como base para la realización de una colonoscopia mediante SOFA.

Para ello el proyecto tendrá la finalidad, alcanzando una serie de objetivos de manera gradual, de parametrizar dicho modelo de manera adecuada con el fin de que pueda ser utilizado para el consecuente desarrollo de la simulación de una colonoscopia en líneas futuras.

- El modelo del intestino debe tener las propiedades físicas adecuadas que respondan de la forma más adecuada posible ante las deformaciones causadas por otro objeto en la simulación.
- Estudio y selección de los distintos valores de los parámetros que modelen el intestino, realizando una comparativa y justificando adecuadamente cada paso en la toma de decisiones.
- Simulación de un segundo objeto que pueda interactuar con el intestino para estudiar el comportamiento del mismo ante una colisión y ajustar de manera apropiada los parámetros que lo requieran.
- Desarrollar funcionalidades que permitan al usuario manipular el objeto ajeno al intestino en la simulación mediante el teclado o el ratón para controlar su movimiento.
- Incluir una segunda vista en la simulación que muestre la perspectiva del segundo objeto y permita observar el interior del intestino cuando se introduzca en el mismo.
- Optimización del rendimiento mediante diversas librerías y *plugins* disponibles, alcanzando una tasa de fps adecuada.

Con el logro de los anteriores objetivos, el proyecto desarrollado permitirá añadir a posteriori un modelo de colonoscopio que, parametrizado adecuadamente, un usuario podrá manipular en tiempo real dentro de la simulación, obteniendo un resultado que sirva como base para la realización simulada de una colonoscopia.

3. Metodología y desarrollo

Antes de comenzar con el desarrollo del proyecto, es indispensable comprender la funcionalidad de determinados componentes dentro de la simulación, así como sus propiedades y casos de uso.

3.1. Geometría del modelo

La geometría del modelo está representada por mallas, que son archivos de datos en los que se definen los vértices, aristas y caras que constituyen la estructura de un objeto. Éstos pueden ser conformados por simplices² (triángulos y tetraedros) o hipercubos³ (cuadrados y hexaedros). En el anexo E se definen los diferentes tipos de archivo de una malla y la clase de información que pueden contener en consecuencia.

La cantidad de componentes geométricos que representan un objeto afecta a la calidad del comportamiento físico de un objeto, aumentando el coste computacional proporcionalmente, por lo que la selección de una malla adecuada es un aspecto crítico y será analizado con detenimiento más adelante.

3.2. Estructura de una simulación

La estructura de una simulación está definida por los siguientes componentes:

- *Scene graph*: Representa una simulación y está compuesta por una serie de nodos organizados en una estructura de árbol o grafo acíclico dirigido. La separación en nodos y subnodos permite individualizar cada componente y trabajar en ellos de manera independiente, en un sistema escalable y modular. La creación de un *scene graph* puede realizarse mediante un *script XML* o *python*, éste último mediante la integración del *plugin SofaPython3*.
- *Root node*: Es el principal nodo de la simulación y podrá tener una cantidad indefinida de subnodos. Posee una serie de atributos que permiten definir aspectos fundamentales de la simulación como la duración de un paso temporal o la gravedad del sistema.
- *Child nodes*: Son los subnodos de la simulación. Generalmente, la simulación suele separar el modelo físico, visual y de colisiones en nodos independientes, los cuales se analizarán más adelante en esta sección. A su vez, cada uno de ellos tendrá otra serie de *child nodes* que representarán una serie de propiedades del sistema y tendrán atributos cuyo valor se podrá asignar, como la masa de un objeto o las fuerzas aplicadas sobre el mismo. Cabe destacar que cada componente puede estar definido por diferentes clases, cada una con unas características y funcionalidades propias.

²Un simplex o k-simplex en geometría es el análogo en k dimensiones de un triángulo.

³Un hipercubo o k-cube en geometría es el análogo en k dimensiones de un cuadrado.

3.3. Principales componentes de la simulación

- *AnimationLoop*: es uno de los componentes principales de la simulación y se encarga de realizar las llamadas al resto de componentes para ejecutar cada tarea en el orden adecuado. Se pueden consultar en el anexo F.1 algunas de las distintas clases disponibles para el componente.
- *Visitors*: son componentes diseñados para interactuar y obtener información de los distintos componentes del *scene graph*, recorriendo su jerarquía mediante clases abstractas.
- *CollisionPipeline*: Se divide en varios componentes que se encargan de la detección, computación y resolución de colisiones; y se ejecutan en cada paso temporal. En el anexo G se detallan sus distintas fases.

3.4. Modelo físico

El modelo físico o modelo mecánico permite definir el comportamiento y las propiedades físicas del objeto simulado. Entre sus distintos componentes es preciso mencionar los siguientes:

- *MechanicalObject*: es el componente principal del modelo físico y almacena la información de la estructura y geometría del objeto a partir de la malla utilizada, la cual se importa en la simulación a través del componente *MeshLoader*. Almacena los vectores de estado de los vértices que conforman la geometría, los cuales pueden ser de una o más dimensiones en función del tipo de información que contengan. Los tipos de información compatibles se recogen en el anexo H, y deben indicarse en el atributo *template* del componente. Se utilizará en este proyecto el valor correspondiente a la representación tridimensional de un objeto: Vec3d.
- *TopologyContainer*: indica qué información geométrica se utilizará para el modelo, en función de la clase elegida del componente. Este componente es imprescindible ya que es posible que un objeto contenga información de, por ejemplo, triángulos y tetraedros. El uso exclusivo de los primeros permitirá representar la superficie del objeto pero no su volumen, a diferencia de los segundos.
- *GeometryAlgorithms*: indica, en función de la clase utilizada para el componente anterior, el tipo de algoritmos que se utilizarán para realizar cálculos sobre la geometría de dicho objeto.
- *Solvers*: son los componentes encargados de conformar un sistema de ecuaciones que procese todas las variables implicadas en la simulación y, posteriormente, resolverlo; repitiendo el proceso una vez por cada paso temporal. En concreto, los *ODESolvers* se encargan de la primera tarea y los *linear solvers* de la segunda. Poseen una serie de clases que se distinguen por ser utilizadas en casos muy específicos como la

interacción de fluidos o la representación de temperatura, siendo los más habituales los utilizados en la simulación de cuerpos rígidos y deformables.

- *Mass*: define la masa y/o densidad del objeto. Pueden ser consultadas en el apéndice F.2 las clases disponibles.
- *ForceField*: permite definir cualquier tipo de fuerzas en la simulación, tanto las fuerzas externas aplicadas sobre los objetos de la misma como las fuerzas internas de cada uno. En el anexo F.4 se detallan las más comunes.

3.5. Modelo de colisiones

Mientras que el componente *CollisionPipeline* se encarga de realizar las diferentes fases de detección, computación y resolución de colisiones; comentadas en mayor profundidad en el anexo G, el modelo de colisiones de un objeto está constituido por los siguientes componentes:

- *MechanicalObject*: al igual que en el modelo físico, éste componente almacena la información geométrica del objeto.
- *MeshTopology*: realiza la función análoga al componente *TopologyContainer* del modelo físico.
- *CollisionModels*: también llamadas primitivas de colisiones, definen qué componentes geométricos de la estructura se utilizarán para la detección y computación de colisiones. Por ejemplo: utilizar una primitiva basada en los vértices, *PointCollisionModel*, será más precisa pero implicará una mayor carga computacional que utilizar únicamente los triángulos que conforman la superficie del objeto, con la primitiva *TriangleCollisionModel*.
- *Mapping*: es el componente encargado de asociar los modelos físico y de colisiones, enviando la información del modelo de colisiones al físico para que pueda reaccionar en consecuencia en función de las propiedades físicas asignadas al objeto.

3.6. Modelo visual

Es el encargado del apartado visual de la simulación, permitiendo percibir con más detalle el comportamiento de la misma. Esto se logra con la utilización de una segunda malla que este compuesta por una mayor densidad de elementos. Además

Sólo posee dos componentes principales: *OglModel*, correspondiente al *MechanicalObject* en los otros dos modelos, y *Mapping*, con igual funcionalidad que en el modelo de colisiones. Además, el primer componente permite obtener más información de la malla referente al apartado visual, como color, textura o sombreado.

4. Parametrización del modelo

A continuación, se detallan ampliamente los componentes y las clases utilizadas para cada uno, justificando el motivo de su elección y uso. Dichos componentes se han ido incorporando de manera gradual a lo largo de diferentes pruebas mencionadas en el apartado 5, donde se analiza su eficiencia comportamental y computacional.

4.1. Entorno y detección de colisiones

Se ha optado, para el desarrollo del *script* de la simulación, por el uso de Python frente a XML mediante la utilización del *plugin* SofaPython3. Dicha elección permite no sólo simplificar la sintaxis del código si no la posibilidad de añadir funcionalidades externas al mismo, como la posibilidad de manipular el movimiento de elementos de la simulación mediante el teclado del equipo.

A la hora de definir el *root node*, se deberá tener en cuenta la asignación de una serie de atributos y componentes ya mencionados anteriormente, indispensables en la simulación:

- El atributo *dt* permite definir la duración del paso temporal de la simulación. Esto afecta no sólo a la velocidad de la misma si no también a su precisión, ya que cabe recordar que la detección de colisiones y la resolución de las ecuaciones del sistema se ejecuta una vez por paso temporal.
- El atributo *gravity* permite definir la gravedad de toda la simulación mediante un vector tridimensional.
- El componente *AnimationLoop* usará la clase genérica *DefaultAnimationLoop*, puesto que las demás se enfocan a simulaciones específicas como el uso de *multithreading* para simular múltiples colisiones o la simulación de bucles.
- El componente *CollisionPipeline* no tiene clases distintivas, pero sí las tienen los componentes que definen cada fase: detección, computación y resolución. En este caso, las clases de cada una están asociadas al comportamiento específico de la simulación frente a colisiones y deben asignarse de manera congruente. En el caso que atañe a este proyecto, se utilizará la detección por proximidad, puesto que las otras opciones son específicas para topologías concretas compuestas por determinados elementos u objetos con estructuras geométricas simples, el cuál no es el caso del intestino.
- Se asignarán al *root node* los subnodos que corresponden al modelo físico, visual y de colisiones de los objetos de la simulación. Las funciones *addChild* y *addComponent* permiten asignar de manera sencilla los nodos hijo y componentes de cada nodo.

4.2. Selección de la malla

Respecto a la selección de la malla, teniendo en cuenta que este proyecto no abarca el ámbito del modelado 3D, se ha optado por la búsqueda en la web de diferentes modelos de intestino disponibles, usando finalmente uno aportado por los tutores dado la falta de modelos gratuitos, precisos y sin errores en la topología.

La selección de una malla adecuada es un aspecto crítico en la simulación, ya que la cantidad de componentes geométricos que conforman la misma (vértices, aristas y caras) afecta enormemente al rendimiento de la simulación, así como a la precisión en el comportamiento del intestino. Más adelante se realizarán las pruebas correspondientes, manipulando la cantidad de componentes de la malla mediante la utilización del *software* Blender⁴.

La malla utilizada para el modelo visual puede y debe contener una cantidad de componentes mayor, ya que el apartado visual requiere una precisión adecuada para percibir una forma del objeto sin inconsistencias, además de que el coste computacional de dicho modelo tiene un menor impacto en el rendimiento que los otros. Además, se utilizará el formato de archivo obj ya que está enfocado a la visualización 3D de objetos.

En el caso del modelo físico y de colisiones, la cantidad de componentes debe ser lo suficientemente reducida para no disminuir drásticamente el rendimiento, manteniendo los suficientes para que se perciba un comportamiento con una calidad adecuada. Se utilizará la misma malla en ambos modelos, para que el efecto de colisiones en la geometría afecte adecuadamente a su comportamiento físico, pudiendo provocar una gran ineficiencia el uso de mallas dispares en ambos modelos. El formato de archivo adecuado para dicha malla es msh, por ser el más eficiente en cuanto a la representación geométrica del volumen de un objeto.

4.3. Selección y parametrización de los componentes

Una vez importada la malla debe decidirse qué información geométrica será utilizada para modelar su topología. En el caso de modelar superficies, pueden utilizarse únicamente los triángulos que corresponden a las caras exteriores de la misma, mientras que en el caso de modelar un objeto volumétrico, se utilizarían tetraedros para definir su volumen. La propia geometría de la malla (conformada por triángulos) y la necesidad de definir un intestino volumétrico, resultan en que las clases *TetrahedronSetTopologyContainer* y *TetrahedronSetGeometryAlgorithms* sean las más adecuadas para la simulación.

En cuanto a los *solvers*, se asignarán las clases genéricas de uso común para la simulación de cuerpos rígidos o deformables, como ya se ha mencionado en la sección 3.4. A tales aspectos, cabe mencionar que la clase utilizada para el *linear solver*, *CGLinearSolver*,

⁴Blender es un conjunto de herramientas de creación y modelado 3D gratuito y de código abierto. Web oficial: blender.org

resuelve el sistema de ecuaciones de manera aproximada, realizando una serie de iteraciones en cada paso temporal para alcanzar un resultado más preciso. Dichas iteraciones pueden definirse en el atributo *iterations*, así como los atributos *threshold*, que indica el margen de error máximo de la aproximación, y *tolerance*, que indica la precisión de la misma.

A la hora de asignar la masa, la elección de la clase *DiagonalMass* es la más equilibrada en cuanto a precisión y eficiencia y tiene en cuenta la densidad del objeto para una distribución de masas no uniforme, por lo que será la más adecuada.

Resta, para el modelo físico, entender qué parámetros modelan las propiedades elásticas asociadas a la deformación del intestino. A este efecto, se pueden encontrar estudios y artículos, como el publicado en Journal of Biomechanics sobre las propiedades biomecánicas de diferentes órganos [21], que parametrizan dichas propiedades mediante el módulo de Young y el coeficiente de Poisson. Ambas propiedades permiten definir la capacidad de deformación de un material ante determinadas fuerzas. En el caso de geometrías tetraédricas, pueden modelarse mediante la clase *TetrahedronFEMForceField*.

Para los modelos de colisiones y visual, la clase del componente *Mapping* adecuada para la simulación de cuerpos deformables es *BarycentricMapping*. Además, la primitiva definida para el modelo de colisiones será *TriangleCollisionModel*. Cabe mencionar la importancia del atributo *ContactStiffness*, que permite ajustar la rigidez del contacto permitiendo una colisión percibida de forma más natural.

Se han realizado diversas pruebas para el estudio de los aspectos analizados, como la comparativa entre diferentes topologías, la modificación de parámetros de los *solvers* o la utilización de distintas clases de los componentes *Mass* y *ForceField*, entre otros, que podrán consultarse en el apartado 5.

Por último, han de tenerse en cuenta la utilización de algunas clases a mayores para los siguientes aspectos de la simulación:

- Una fuerza constante que pueda ser manipulada para mover el objeto que colisionará con el intestino. Dicha fuerza puede definirse mediante la clase *ConstantForceField* y deberá poder ser controlada en tiempo real por parte de un usuario.
- El intestino debe estar sometido tanto a una fuerza de gravedad como a una fuerza que lo mantenga fijo a su posición original, igual que lo harían los demás órganos y estructuras del cuerpo. A tales efectos, *RestShapeSpringsForceField* es la única clase que puede mantener un cuerpo fijo en una posición de reposo a la vez que permite su deformación. Sus atributos *stiffness* y *angularStiffness* permiten modificar la fuerza con la que el objeto recupera su posición inicial tras la deformación.
- Para asignar una cámara secundaria desde la perspectiva de un objeto está disponible la clase *OglViewport*.

- Debe tenerse en cuenta que el intestino difiere ligeramente en sus propiedades elásticas en cada una de sus secciones [24]. La clase *BoxRoi* permite dividir un objeto en diferentes partes a las que se podrán asignar distintas propiedades.

4.4. Escala y unidades de medida

SOFA es adimensional, por lo que no utiliza unidades de medida para ningún parámetro, ni si quiera para evaluar la escala del modelo que se está usando. En consecuencia, todos los valores del modelo deben ser congruentes entre sí. Por ejemplo, si la escala del objeto está representada en centímetros en lugar de metros, la densidad de masa debe ser asignada en kg/cm^3 , y la fuerza de la gravedad en cm/s^2 . Esto debe ser así para todos los valores asignados.

Por otra parte, asumir una escala real de longitud no siempre es óptimo en la simulación, ya que usar valores numéricos demasiado elevados o pequeños puede llevar a aproximaciones inexactas en los cálculos de los *solvers* o incluso errores en la simulación.

Dado lo mencionado hasta ahora, cabe mencionar que la naturaleza de SOFA implica generalmente una gran cantidad de ensayos de prueba y error tratando de ajustar determinados parámetros como la masa, el paso temporal o la aplicación de fuerzas en la simulación; partiendo de un valor aproximado al real y ajustando valores y escalas.

4.5. Valor de los parámetros

Al margen de la problemática que se plantea en el anterior apartado, sí resulta necesario partir de unos valores apropiados y que se aproximen a la realidad. Para ello se han obtenido los valores medios de los siguientes parámetros: un módulo de Young y coeficiente de Poisson de $2,63MPa$ y $0,33$ [21] y una densidad de masa de $1088kg/m^3$ [22]. También se ha comprobado que, al margen de la escala final, la malla del modelo respeta la proporción de las dimensiones de las distintas secciones del intestino [23]. Por otra parte, como se había mencionado en el apartado 4.3, el módulo de Young y el coeficiente de Poisson varían levemente en las diferentes secciones [24].

Otros valores que afectan a la simulación pero son intrínsecos del escenario son el *dt* y los parámetros de los *solvers*. Para asignarles un valor en el margen correcto, se han usado como base varios ejemplos de simulaciones disponibles en el GitHub de SOFA⁵ que modelan un hígado. El objetivo principal de las pruebas recae en el ajuste preciso de estos parámetros para simular un comportamiento real. De esta manera, tras realizar las pruebas que se mencionarán a continuación, los valores que han demostrado ser más estables son:

- Un paso temporal de 0,4.

⁵SOFA GitHub web

- Unos valores de 0,5 y 0,3 para los atributos *alarmDistance* y *contactDistance* de la clase *MinProximityIntersection*.
- Una cantidad de iteraciones entre 25 y 30 junto con un valor tanto de *tolerance* como de *threshold* de 10^{-6} para el *CGLinearSolver*.

5. Pruebas realizadas

A continuación se mencionan las pruebas más significativas realizadas a lo largo del proyecto. Se podrá consultar el anexo D para un análisis más minucioso de las mismas, así como de diferentes imágenes de la evolución de la simulación a lo largo de las mismas junto con una serie de gráficas del rendimiento, obtenidas a través de una herramienta de análisis que incluye el propio *framework*.

5.1. Prueba 1: Simulación inicial

- Objetivo: Análisis y observación de las diferentes pruebas que SOFA aporta como base para el desarrollo de nuevas simulaciones, con el fin de obtener un modelo viable a partir del cual desarrollar la simulación que ataña a este proyecto.
- Resultados: Entre las distintos ejemplos destaca la simulación de una colisión entre un hígado y un cuerpo esférico rígido. En ella se aplica una fuerza a la esfera, imitando la gravedad, que colisiona en la parte superior del órgano permitiendo observar la deformación del mismo y cómo éste recupera su forma. Analizando las clases y mallas utilizadas se puede observar, al comparar la escala de las mallas del hígado y el intestino, que la escala utilizada en el hígado dista mucho de la escala real, teniendo un diámetro transversal de apenas 10mm.
- Conclusiones: La simulación del hígado es un buen punto de partida ya que muestra la deformación de un órgano tras una colisión y cómo éste recupera su forma original tras la misma. Por otra parte, cabe pensar que el efecto de la escala en una simulación puede ser determinante en términos de eficiencia, ya que SOFA computa cualquier parámetro de manera adimensional. Esto se debe a la dificultad computacional que implica el uso de valores demasiado elevados o con excesivos decimales, de forma que las unidades de los parámetros deben ser proporcionales a la escala a la que se ha desarrollado la malla.

5.2. Prueba 2: Uso de diferentes escalas en la simulación

- Objetivo: Estudio de la colisión de un cuerpo esférico rígido con el intestino a diferentes escalas, tanto en términos de eficiencia como a nivel visual y escalando de forma análoga cada parámetro. Se realizarán las pruebas con la escala aproximada a las dimensiones del hígado reales, una escala 10 veces mayor y una escala 10 veces menor. Posteriormente, se analizarán en los valores adecuados para la parametrización de un intestino, dejando dicho análisis para la siguiente prueba.

5.3 Prueba 3: Parametrización del intestino

- Resultados: La colisión no produce deformación alguna en ninguna de las 3 escalas. La única diferencia notoria es la velocidad percibida de la esfera, siendo ésta menor a mayor escala. Por otra parte, se puede apreciar que los fps de la simulación han disminuido, aunque sin afectar al rendimiento visual: mientras que la simulación del hígado rondaba los 120 fps, ésta segunda simulación con las mallas del intestino se aproximan a los 80 fps en todas las escalas examinadas.
- Conclusiones: Por una parte, se puede deducir que la velocidad de la esfera se ve afectada dado que a mayor escala la distancia que debe recorrer aumenta en proporción a la misma. Para evitar esto se puede modificar tanto la fuerza aplicada como el paso temporal de la simulación, con las consecuencias de que en el primer caso la fuerza del impacto podría provocar una deformación excesiva y en el segundo una pérdida de calidad considerable. Por otra parte, la disminución de fps es una consecuencia razonable de utilizar una malla más compleja y con mayor cantidad de componentes.

5.3. Prueba 3: Parametrización del intestino

- Objetivo: Análisis a diferentes escalas de los parámetros apropiados del intestino, los cuales ya han sido comentados en el apartado 4.5.
- Resultados: La colisión a escala real se comporta de manera adecuada, provocando una deformación del intestino y la recuperación de su forma original de manera similar a lo ocurrido con el hígado.

Las colisiones a mayor y menor escala se comportan de manera anormal. Mientras que la primera no muestra deformación alguna, la segunda provoca una deformación excesiva del intestino.

De manera añadida, la falta de sujeción del intestino en el espacio provoca que este, tras la colisión, se vaya desplazando lentamente hacia abajo.

- Conclusiones: Hilando con las conclusiones de la anterior prueba, el efecto de mantener la fuerza de la esfera provoca esa deformación exagerada a menor escala y es insuficiente para lograr una deformación a mayor escala. Se puede concluir que, como se menciona en el apartado 4.4, no todas las escalas resultan adecuadas y en el caso que ocupa este proyecto la más adecuada es aquella cuyas dimensiones se encuentran en la media de las de un intestino real.

5.4. Prueba 4: Ajuste de parámetros

- Objetivo: Se ajustarán el paso temporal y la fuerza de contacto de la colisión en un último intento de solventar los problemas en las simulaciones escaladas. La fuerza de contacto se puede ajustar en el atributo *contactStiffness* de las primitivas de colisión.

- Resultados: A mayor escala sigue sin apreciarse una deformación en el intestino, mientras que a menor escala sí se logra un comportamiento similar reduciendo la fuerza aplicada a la esfera y aumentando el valor del atributo *contactStiffness*.
- Conclusiones: No es posible ajustar adecuadamente los parámetros utilizando una malla de longitud elevada, requiriendo aplicar una fuerza excesiva a los objetos simulados resultando aun así en una colisión demasiado rígida. Debido a esto, queda descartada la simulación a mayor escala y las pruebas que prosigan serán realizadas únicamente con la escala normal y la menor.

5.5. Prueba 5: Sujeción del intestino

- Objetivo: En un estado natural el intestino se ve sometido a fuerzas, provocadas por la proximidad y el contacto de diferentes órganos y estructuras del cuerpo, que le permiten deformarse ante determinadas fuerzas puntuales pero le mantienen fijo en su posición. Por tanto, debe encontrarse la manera adecuada de simular dicha fuerza sin que esta afecte a sus propiedades elásticas.
- Resultados: Entre las diversas clases existentes que permiten fijar un cuerpo en una posición fija, *RestShapeSpringsForceField* es la única que permite la deformación de un cuerpo a la vez que lo mantiene sujeto a su posición inicial. Se han comprobado otras clases como *FixedConstraint*, pero sólo la primera mencionada permite la deformación del cuerpo y mantener su sujeción de manera simultánea. No afecta de manera distintiva a los fps de la simulación.
- Conclusiones: La clase mencionada es la única que se adecúa a la necesidad de sujetar el intestino en su posición, además de que permite el ajuste de varios parámetros que definen la resistencia del objeto a ser deformado, y deben escalarse apropiadamente en las simulaciones de ambas escalas.

5.6. Prueba 6: Comparación entre una malla cerrada y una hueca

- Objetivo: Hasta ahora la malla utilizada en las simulaciones era una malla cerrada, sin abertura en el colon. Se procederá a modificar dicha malla mediante el *software* Blender y se realizará una comparativa entre la malla cerrada y la nueva malla para poder apreciar la diferencia de comportamiento entre el volumen de un intestino cerrado y el volumen correspondiente sólo a las paredes del mismo.
- Resultados: En la escala normal la esfera rebota y sale despedida a gran velocidad tras la colisión, mientras que en la escala reducida deforma el intestino como hasta ahora, aunque mucho más brevemente.
- Conclusiones: La simulación de un cuerpo con un volumen mucho mayor, como es natural, implica una resistencia mayor a una deformación aplicada con la misma fuerza.

5.7. Prueba 7: Comparación de mallas triangulares y tetraédricas

- Objetivo: Al disponer ahora de una malla hueca, cabe estudiar si sería posible simular de forma viable el intestino con una superficie triangular en lugar de un volumen tetraédrico. De ser positivo el resultado, permitiría una simulación adecuada con un coste mucho más reducido, ya que no se procesaría la totalidad del volumen del objeto. Esto se puede lograr de manera sencilla modificando los componentes relativos a la geometría y utilizando el componente de fuerza elástica análogo a superficies triangulares.
- Resultados: Las simulaciones en ambas escalas provocan deformaciones inestables e incluso la ruptura de la malla.
- Conclusiones: El uso de una malla triangular no es eficiente para objetos volumétricos.

5.8. Prueba 8: Esfera volumétrica

- Objetivo: Dotar el cuerpo esférico de volumen y cambiar su comportamiento rígido por uno deformable, en aras de obtener una colisión más acorde a la realidad.
- Resultados: La escala original presenta el mismo comportamiento que la escala mayor en la prueba 4, mostrando una simulación demasiado rígida o que no colisiona adecuadamente debiendo aumentar o reducir en exceso determinados valores. La escala menor revela un comportamiento adecuado, aunque el coste de computación aumenta provocando un descenso de fps a casi 30 tras procesar la colisión. Visualmente, la deformación en el órgano es más clara y la esfera se hunde moderadamente antes de que las fuerzas internas del intestino la desvíen en el eje horizontal.
- Conclusiones: Finalmente la escala más adecuada es la menor, adquiriendo ahora sentido la escala reducida que se utilizaba en el ejemplo inicial. El uso de un cuerpo volumétrico para la esfera muestra un aumento de calidad visible, remarcando la importancia que adquiere parametrizar la masa, elasticidad y rigidez de contacto de manera adecuada en la simulación. El coste computacional aumenta, como es razonable, y el desvío de la esfera en el eje horizontal es debido a que la fuerza de la gravedad todavía no ha sido asignada adecuadamente.

5.9. Prueba 9: Incorporación de modelos visuales

- Objetivo: Proporcionar un modelo visual al intestino y la esfera.
- Resultados: La diferencia de una malla distinta mucho más precisa para el modelo visual permite apreciar de manera mucho más clara las deformaciones de la simulación. Los fps se mantienen estables, sin verse afectados.

5.10 Prueba 10: Reducción de elementos de la malla

- Conclusiones: El uso de modelos visuales en la simulación logra una mejora en la calidad visual con un aumento mínimo en el coste, ya que sólo reflejan la respuesta del procesamiento de los modelos físico y de colisiones.

5.10. Prueba 10: Reducción de elementos de la malla

- Objetivo: Modificar la cantidad de componentes de la malla, tanto de la utilizada para los modelos físico y de colisiones como para el modelo visual, estudiando su impacto en la eficiencia de la simulación y eligiendo aquella más equilibrada en términos de eficiencia y calidad visual.
- Resultados: Se han modificado las mallas para realizar diferentes pruebas, desarrollando 5 mallas con diferente cantidad de componentes y cuyos resultados se pueden apreciar en la tabla 1, pudiendo consultarse la cantidad de componentes de cada malla en el anexo I.

Rendimiento de las mallas	
Malla	Mínima cantidad de fotogramas por segundo alcanzada
Intestino	27 fps
Intestino 2	31 fps
Intestino 3	42 fps
Intestino 4	51 fps
Intestino 5	61 fps

Tabla 1: Fotogramas por segundo en el momento de mayor carga computacional según la cantidad de componentes de las mallas, ordenadas de mayor a menor cantidad.

- Conclusiones: Se elegirá la malla correspondiente al 'Intestino 5', teniendo en cuenta que hasta ahora había estado siendo utilizada la denominada 'Intestino 2'. Cabe mencionar que se han intercambiado también las mallas del modelo visual con las de los modelos físico y de colisiones, pero la variación en las primeras apenas supone un impacto considerable.

5.11. Prueba 11: Diferentes clases para definir la masa

- Objetivo: Analizar otras clases que permitan definir la masa y densidad de un objeto. Dichas clases son descritas brevemente en el anexo F.2.
- Resultados: De las únicas otras dos clases disponibles, *MeshMatrixMass* y *Uniform-Mass*, la primera aumenta el coste de manera excesiva situando el mínimo de fps en 24 sin mostrar un cambio perceptible en el comportamiento, mientras que la segunda no afecta a la eficiencia pero tampoco al comportamiento de la simulación.
- Conclusiones: Mientras que una de las opciones sugiere un coste elevado, la única alternativa viable mantiene un buen rendimiento pero implica una distribución de

5.12 Prueba 12: Diferentes clases para definir las fuerzas internas del intestino

la masa uniforme y no permite asignar la densidad del objeto, sólo su masa. La opción actual sigue siendo la más adecuada: *DiagonalMass*.

5.12. Prueba 12: Diferentes clases para definir las fuerzas internas del intestino

- Objetivo: Analizar otras clases que permitan definir las fuerzas internas que reflejen el comportamiento de los cuerpos ante una colisión. En este caso, no todas las clases son compatibles con un objeto volumétrico conformado por tetraedros, por lo que la selección de alternativas se reduce a uno: *TetrahedralCorotationalFEMForceField*. Otras de las clases más comunes del componente son mencionadas en el anexo F.4
- Resultados: No se aprecian efectos notables ni en los fps de la simulación ni a nivel visual.
- Conclusiones: La clase *TetrahedralCorotationalFEMForceField* puede llegar a ser esencial en otro tipo de deformaciones más complejas que sometan a un objeto a una tensión mucho más grave y prolongada, pero en el caso que atañe a este proyecto no parece implicar un cambio de ningún tipo.

5.13. Prueba 13: Pruebas con *solvers*

- Objetivo: Comprobar el efecto de modificar los parámetros de los *solvers* y, más específicamente, los *linear solvers*, ya que modifican la cantidad de tiempo de procesamiento que se le dedica a un determinado objeto en la simulación.
- Resultados: Aumentar los parámetros del *linear solver* del intestino y disminuir los de la esfera provocan que la colisión se perciba más claramente en toda la sección en la que la esfera colisiona, pero la deformación en el punto exacto de la colisión es menor. Además, los fps disminuyen moderadamente, alcanzando un mínimo de 45.
- Conclusiones: La utilización de valores diferentes en los *solvers* de distintos objetos parece una elección acertada cuando alguno de ellos requiere un interés mayor a costa de los demás, a pesar de que puede afectar significativamente al coste. Además, utilizar *solvers* comunes no afecta al coste computacional.

5.14. Prueba 14: Pruebas con diferentes primitivas de colisión

- Objetivo: Comprobar como afecta la elección entre un modelo o primitiva de colisión conformado por esferas, como ocurría en la simulación inicial con la esfera, y un modelo conformado por triángulos, que afecta a las caras de los tetraedros que conforman el volumen de ambos objetos actualmente. Cabe recordar que en la sección 3.5 se mencionaba la funcionalidad de dichos modelos.

5.15 Prueba 15: Simulación de la gravedad y ajuste de parámetros

- Resultados: En la simulación inicial la esfera sin volumen estaba dotada de un modelo de colisiones esférico conformado por una sola esfera cuyo radio se indicaba en un atributo de la clase. Cuando el modelo se utiliza para un volumen, este genera esferas superpuestas hasta cubrir dicho volumen, lo que implica que las esferas más superficiales sobresalgan un espacio considerable y provoquen que la colisión se produzca sin que los modelos visuales coincidan adecuadamente, dejando un espacio muy visible entre los objetos.
- Conclusiones: El componente *SphereCollisionModel* es inviable para representar un cuerpo volumétrico esférico en la simulación.

5.15. Prueba 15: Simulación de la gravedad y ajuste de parámetros

- Objetivo: Se añadirá la gravedad a todo el entorno de simulación, ya que el intestino también debe estar sometido a la misma para reflejar la realidad de forma adecuada.
- Resultados: El efecto de la gravedad en el intestino requiere una consiguiente modificación de los valores del componente *RestShapeSpringsForceField*, entre otros, ya que el intestino ahora se ve levemente desplazado por la gravedad aplicada.
- Conclusiones: La colisión se percibe de manera más natural ahora, ya que la esfera sigue deformando el intestino en la zona del impacto pero afecta de forma más visible al resto de la sección del intestino afectada.

5.16. Prueba 16: Manipulación del movimiento de la esfera y adición de una cámara secundaria

- Objetivo: Permitir a un usuario mover la esfera mediante el uso del teclado, así como mantenerla fija en un punto. A mayores, se debe añadir una cámara secundaria que muestre la perspectiva de la misma en cada momento para poder apreciar el interior del intestino cuando el usuario la introduzca en el mismo.
- Resultados: Ha resultado posible aplicar una fuerza constante al intestino en cada uno de los ejes según la tecla pulsada, así como detener su movimiento cuando convenga. Por otro lado, la cámara se mantiene fija asociada a uno de los vértices de la superficie de la esfera y se ha añadido la opción de rotar el ángulo de dicha cámara en cada eje a través de diferentes teclas.
- Conclusiones: Tanto la manipulación del movimiento de la esfera como la vista desde una segunda perspectiva y la posibilidad de rotar la cámara de la misma han sido implementadas con éxito.

5.17 Prueba 17: Uso del *BoxRoi* para modelar las secciones del intestino con más precisión

5.17. Prueba 17: Uso del *BoxRoi* para modelar las secciones del intestino con más precisión

- Objetivo: En aras de tener en cuenta que las diferentes secciones del intestino poseen propiedades ligeramente dispares [24], es posible mediante el componente *BoxRoi* aislar partes del volumen de mismo objeto y otorgarles distintas propiedades a cada una.
- Resultados: El coste de asociar las distintas secciones a un *BoxRoi* con distintos valores de módulo de Young y coeficiente de Poisson implica un aumento crítico del coste computacional, provocando que la simulación se mantenga por debajo de los 30 fps.
- Conclusiones: Deben utilizarse, alcanzado el punto actual, *plugins* que permitan el aprovechamiento máximo de los recursos del equipo para una optimización adecuada de la simulación.

5.18. Prueba 18: Optimización con el *plugin Multithreading*

- Objetivo: Utilizar el *plugin multithreading*, que permite aprovechar los núcleos de los que dispone la CPU realizando tareas de manera simultánea, para mejorar la eficiencia de la simulación.
- Resultados: La mejora obtenida, de al menos 10 fps, sigue manteniendo un nivel bajo de fps superando ligeramente los 30. En ausencia de *BoxRoi*, supone también un aumento de aproximadamente 10 fps.
- Conclusiones: El uso del *plugin* supone una mejora, aunque insuficiente para mantener unos fps elevados.

5.19. Prueba 19: Optimización con el *plugin CUDA*

- Objetivo: Utilizar el *plugin SOFA*, que permite ejecutar la simulación mediante la GPU, para mejorar la eficiencia de la simulación.
- Resultados: CUDA no parece funcionar de manera adecuada en la simulación, provocando una caída de fps aun mayor debido al mal funcionamiento o implementación del *plugin*. Para solventar dicho comportamiento, se han realizado las siguientes comprobaciones:
 - Se han ejecutado ejemplos de simulaciones con CUDA y han funcionado correctamente, mejorando con creces las simulaciones probadas en comparación con la ausencia del *plugin*. Se descarta la opción de una instalación o configuración errónea del mismo.

- La implementación con CUDA se realiza sustituyendo determinadas clases por otras, sin modificar sus valores, por lo que se ha procedido a simular cada cambio de manera individual y focalizar la zona de error. Ha resultado inconcluso dado que ciertos componentes deben modificarse en conjunto a costa de un fallo de compatibilidad del *plugin*.
- Se han sustituido sólo las mismas clases que en otros ejemplos que aporta SOFA, pero el resultado ha sido inefectivo o insuficiente ya que los componentes con más coste actualmente son los *TetrahedronFEMForceField* asignados a los distintos *BoxRoi*.
- La ejecución del *plugin* eliminando los *BoxRoi* tampoco parece ser el motivo, descartando la posible incompatibilidad entre componentes.
- Conclusiones: No ha sido posible ejecutar la simulación con CUDA de manera adecuada, de la misma forma que tampoco se ha logrado encontrar el error que lo causa. A efectos de dejar constancia de ello, el equipo utilizado para el desarrollo del proyecto es un portátil que dispone de una GPU NVIDIA GeForce RTX 3050. Finalmente se ha optado por tratar de implementar el componente *BoxRoi* con algunas limitaciones, reduciendo a un sólo *BoxRoi* aquellas secciones con mayor similitud entre ellas. Esto permite una simulación con una mayor cantidad de fps permitiendo a su vez una mínima diferenciación entre las zonas del intestino más dispares.

6. Resultados

La simulación obtenida ha alcanzado los siguientes resultados finales:

- Un comportamiento del intestino adecuado y eficiente, que se deforma ante colisiones de manera adecuada.
- La obtención de modelo parametrizado de forma meticulosa, habiendo realizado un estudio de las diferentes opciones disponibles y realizando una comparativa entre diferentes clases y valores de los parámetros implicados.
- La posibilidad de que un usuario manipule, mediante teclado, la posición de un objeto dentro de la simulación de manera sencilla.
- La visualización desde la perspectiva del objeto manipulado, permitiendo la observación tanto externa como interna del intestino de manera similar a la cámara de un colonoscopio. Además, es posible manipular con el teclado el ángulo de visión de la vista de cámara.
- La utilización exitosa de algunas de las herramientas disponibles para la optimización de la simulación, permitiendo aprovechar las capacidades de computación disponibles en el equipo de la manera más eficiente posible.

7. Conclusiones

El objetivo central expuesto al inicio de este proyecto ha sido alcanzado de manera efectiva, obteniendo un modelo de intestino apto que puede ser utilizado como base para el desarrollo de la simulación de una colonoscopia.

Usando como base la documentación de SOFA⁶, se ha llevado a cabo un amplio estudio en la comparación y selección de clases de cada componente y los valores de sus variables, siendo este el núcleo para la construcción de un modelo apropiado en el *framework*.

La inversión de tiempo requerida para recrear un modelo es uno de los principales factores a tener en cuenta en esta clase de proyectos. El método para conseguir los objetivos planteados sólo ha podido y podrá lograrse en otros proyectos del ámbito mediante ensayos de prueba y error.

En aras de alcanzar la mayor eficiencia posible, la optimización del modelo ha supuesto un aumento moderado de la misma, obteniendo una calidad estable de fps. Esto demuestra la dificultad en el ámbito médico a la hora de conseguir un modelo físicamente preciso y eficiente, dando a entender que el punto más importante no es el modelado en sí, si no equilibrar un modelo simple pero preciso con el uso de herramientas que consigan explotar todas las capacidades de computación disponibles y abordar determinados problemas simplificándolos de manera modular.

La creación de un componente cuyo movimiento pueda ser manipulado por parte del usuario ha sido satisfactorio y las colisiones son estables y precisas, dejando a la discreción del usuario la posibilidad de que otros componentes del teclado y el ratón puedan ser usados para un movimiento más cómodo.

Cabe mencionar que la falta de documentación y ejemplos acerca de la cantidad de componentes, clases y variables; junto con su constante evolución, son una seria dificultad para reconocer cuándo se consigue la mayor exactitud posible para un modelo, así como también la calidad de la malla usada.

Tras obtener un modelo viable, las líneas futuras planteadas para este proyecto son las siguientes:

- Adición de una malla de colonoscopio que, con un valor adecuado de sus parámetros, pueda ser manipulado directamente por el usuario en la simulación.
- Integración de los *plugins* correspondientes que permiten interactuar directamente con la simulación usando realidad virtual y aumentada y dispositivos hápticos.

⁶Documentación oficial de SOFA

- Uso de datos reales como casos de estudio para poder modelar la aparición de pólipos, lesiones y tejidos cancerosos. También puede ser importante evaluar la inflamación y el ensanchamiento ante determinadas situaciones para ajustar el modelo en consecuencia.
- Implementación de nuevas funcionalidades que permitan al usuario cortar el modelo para simular la extracción de pólipos y tejidos del intestino.

Cabe destacar que actualmente se encuentra en las últimas fases de desarrollo un nuevo *framework* denominado Filasofia [25]. Su objetivo es simplificar el desarrollo de las simulaciones y mejorar el rendimiento visual, permitiendo un ajuste más preciso de las mismas usando SOFA y sus componentes como base.

Referencias

- [1] Herrera-Aliaga, E., Estrada, and L. D. (2022). Trends and innovations of simulation for twenty first century medical education. *Frontiers in public health*, 10, 619769.
- [2] Nuzhnaya, K. V., Mishvelov, A. E., Osadchiy, S. S., Tsoma, M. V., AM RS, K. K., Rodin, I. A., and Povetkin, S. N. (2019). Computer simulation and navigation in surgical operations. *Pharmacophore*, 10(4).
- [3] Maghoul, P., Boulet, B., Tardif, A., and Haidar, A. (2017). Computer simulation model to train medical personnel on glucose clamp procedures. *Canadian Journal of Diabetes*, 41(5), 485-490.
- [4] Lamé, G., and Dixon-Woods, M. (2020). Using clinical simulation to study how to improve quality and safety in healthcare. *BMJ simulation & technology enhanced learning*, 6(2), 87.
- [5] Olivier, B. G., Swat, M. J., and Moné, M. J. (2016). Modeling and simulation tools: from systems biology to systems medicine. *Systems Medicine*, 441-463.
- [6] Bin, S., Masood, S., and Jung, Y. (2020). Virtual and augmented reality in medicine. In *Biomedical information technology* (pp. 673-686). Academic Press.
- [7] Wang, L., Dou, Q., Fletcher, P. T., Speidel, S., and Li, S. (2021, September). Medical Image Computing and Computer Assisted Intervention—MICCAI 2022. In *Proceedings of the 24th International Conference*, Strasbourg, France (Vol. 12901, pp. 109-119).
- [8] Sá-Couto, C., Patrão, L., Maio-Matos, F., and Pêgo, J. M. (2016). Biomedical simulation: evolution, concepts, challenges and future trends. *Acta medica portuguesa*, 29(12), 860-868.
- [9] Erdemir, A., Mulugeta, L., Ku, J. P., Drach, A., Horner, M., Morrison, T. M., ... and Myers Jr, J. G. (2020). Credible practice of modeling and simulation in healthcare:

- ten rules from a multidisciplinary perspective. *Journal of translational medicine*, 18(1), 369.
- [10] Faure, F., Duriez, C., Delingette, H., Allard, J., Gilles, B., Marchesseau, S., ... and Cotin, S. (2012). Sofa: A multi-model framework for interactive physical simulation. *Soft tissue biomechanical modeling for computer assisted surgery*, 283-321.
- [11] “Cáncer colorrectal”. World Health Organization (WHO). (2024, 10 de febrero). [Online]. Available: <https://www.who.int/es/news-room/fact-sheets/detail/colorectal-cancer>
- [12] “Cáncer colorrectal - Estadísticas”. Cancer.Net. (2024, 10 de febrero). [Online]. Available: <https://www.cancer.net/es/tipos-de-cáncer/cáncer-colorrectal/estadísticas>
- [13] “Follow-Up Colonoscopy after Positive FIT Test”. National Cancer Institute. (2024, 10 de febrero). [Online]. Available: <https://www.cancer.gov/news-events/cancer-currents-blog/2022/positive-fit-stool-test-colonoscopy>
- [14] Fisher, D. A., Maple, J. T., Ben-Menachem, T., Cash, B. D., Decker, G. A., Early, D. S., ... and Dominitz, J. A. (2011). Complications of colonoscopy. *Gastrointestinal endoscopy*, 74(4), 745-752.
- [15] MAZZOTTA, A. D., & SCHOSTEK, S. Physical Simulator for Colonoscopy: A Modular Design Approach and Validation.
- [16] Morato, R., Tomé, L., Dinis-Ribeiro, M., & Rolanda, C. (2022). Endoscopic skills training: the impact of virtual exercises on simulated colonoscopy. *GE-Portuguese Journal of Gastroenterology*, 29(6), 374-384.
- [17] Vilmann, A. S., Lachenmeier, C., Svendsen, M. B. S., Søndergaard, B., Park, Y. S., Svendsen, L. B., & Konge, L. (2020). Using computerized assessment in simulated colonoscopy: a validation study. *Endoscopy International Open*, 8(06), E783-E791.
- [18] Guerriero, L., Quero, G., Diana, M., Soler, L., Agnus, V., Marescaux, J., & Corcione, F. (2018). Virtual reality exploration and planning for precision colorectal surgery. *Diseases of the Colon & Rectum*, 61(6), 719-723.
- [19] Finocchiaro, M., Cortegoso Valdivia, P., Hernansanz, A., Marino, N., Amram, D., Casals, A., ... & Koulaouzidis, A. (2021). Training simulators for gastrointestinal endoscopy: current and future perspectives. *Cancers*, 13(6), 1427.
- [20] M. Finocchiaro et al., A Framework for the Evaluation of Human Machine Interfaces of Robot-Assisted Colonoscopy, in *IEEE Transactions on Biomedical Engineering*, vol. 71, no. 2, pp. 410-422, Feb. 2024, doi 10.1109/TBME.2024.3342125
- [21] Johnson, B., Campbell, S., and Campbell-Kyureghyan, N. (2020). Biomechanical properties of abdominal organs under tension with special reference to increasing strain rate. *Journal of Biomechanics*, 109, 109914.

- [22] “Density » IT'IS Foundation”. Latest News. (2024, 10 de febrero). [Online]. Available: <https://itis.swiss/virtual-population/tissue-properties/database/density/>
- [23] “Pesos y medidas de los órganos del cuerpo humano”. www.elsevier.com. (2024, 10 de febrero). [Online]. Available: <https://www.elsevier.com/es-es/connect/medicina/pesos-y-medidas-de-los-organos-del-cuerpo-humano>
- [24] CHENG, Wubin. Development of a kinetic model for loop-free colonoscopy technology. 2013. Tesis Doctoral. University of Saskatchewan.
- [25] Poliakov, V., Tsetserukou, D., & Vander Poorten, E. (2023, December). Filasofia: A Framework for Streamlined Development of Real-Time Surgical Simulations. In 2023 21st International Conference on Advanced Robotics (ICAR) (pp. 585-591). IEEE.

A. Estado del arte

A.1. Simulación de colonoscopias

El desarrollo de métodos para la simulación de colonoscopias ha permitido la práctica de habilidades técnicas en un entorno seguro y sin riesgo, logrando una mejora de las aptitudes necesarias. Existen diferentes estudios que han confirmado la mejora de dichas capacidades mediante la simulación, tanto física [15] como virtual [16].

Por otra parte, los simuladores virtuales frente a los físicos son también una gran herramienta de asistencia durante los procedimientos reales [17] y permiten la integración de tecnologías que facilitan una mejor planificación preoperatoria [18], emergiendo como el método más adecuado para la simulación.

A.2. Herramientas de simulación

Poniendo el foco en la diversidad de herramientas disponibles, la calidad exigida del modelo y el requerimiento de conocimientos en diferentes áreas de especialidad, como se menciona en [8] y [9], suponen una dificultad a la hora de seleccionar la más adecuada. También cabe tener en consideración la alta exigencia de una relación entre precisión y coste computacional, a la vez que debe preverse la integración de herramientas externas como dispositivos hápticos que permitan interactuar con el modelo.

Recientes estudios que se han llevado a cabo [19] marcan las pautas decisivas en una herramienta para conseguir alcanzar una simulación a la altura de las tecnologías emergentes, llegando a la conclusión de que el futuro de este ámbito estará determinado por simuladores virtuales colaborativos y accesibles, modulares, precisos tanto en las físicas como el realismo visual, escalables, interactivos, e integrables con tecnologías emergentes como realidad virtual y aumentada o *machine learning*.

Evaluar una comparativa de las herramientas disponibles actualmente puede resultar inconcluyente y poco riguroso, dado que muchas de ellas están enfocadas al uso dentro de un área específica y los criterios para una selección adecuada dependen de las necesidades de cada profesional. En contraparte, sí se ha podido observar, dadas las características mencionadas en la sección 1.2 que SOFA cumple los requisitos adecuados que suplen las necesidades generales demandadas en el ámbito de la simulación médica permitiendo:

- La creación de simulaciones eficientes, con un amplio abanico de posibilidades (*multithreading*, uso de GPU para computar determinadas características, integración de motores gráficos externos, etc.) que permitan aprovechar la máxima capacidad de los recursos disponibles en el equipo alcanzando una utilización óptima de los mismos.

- El uso de un *framework* modular y de código abierto que no sólo permita la colaboración entre profesionales de distintos ámbitos de manera sencilla para un mismo proyecto si no también la posibilidad de integrar fácilmente nuevas funcionalidades por parte de la comunidad de desarrolladores logrando resolver las exigencias demandadas por los profesionales.
- Abarca además la integración de tecnologías emergentes como dispositivos hápticos o *machine learning*, lo que le permite mantenerse adaptado a las necesidades exigidas por el sector profesional en cada momento.

A.3. Modelos existentes

Tras realizar una búsqueda exhaustiva en diversas webs de ámbito académico como Google Scholar, PubMed o la colección de *papers* que alberga SOFA en su web⁷, cabe destacar que se ha realizado en los últimos años un estudio comparativo de diferentes dispositivos hápticos simulando un intestino mediante el *framework* SOFA [20]. Dicho estudio tenía como objetivo establecer una comparativa en el personal médico sobre el esfuerzo requerido y la precisión obtenida realizando una colonoscopia controlada por diferentes dispositivos hápticos, en pos de encontrar el más adecuado. La simulación permitía el movimiento de una cámara en el interior del intestino mediante un dispositivo háptico y utilizaba el *plugin* que integra Unity 3D permitiendo simular los aspectos visuales (texturas, iluminación, etc) de la manera más realista posible.

B. Sección ampliada de metodología y desarrollo

Antes de comenzar con el desarrollo es necesario abordar el funcionamiento de las diferentes clases y parámetros de simulación de manera individual y entender su uso y propiedades de cara a la realización de las primeras simulaciones.

El primer paso que debe realizarse es definir de manera adecuada la geometría del modelo que se pretende simular. En ciencia computacional, esto se logra mediante la discretización de los elementos de su topología y el uso del Método de Elementos Finitos⁸ para su manipulación.

B.1. Geometría del modelo

Las mallas son complejos celulares⁹ compuestos por simplices¹⁰ (triángulos y tetraedros) o hipercubos¹¹ (cuadrados y hexaedros) que representan una superficie o una geometría

⁷<https://www.sofa-framework.org/applications/publications/>

⁸El Método de Elementos Finitos o MEF es un método que permite la resolución de ecuaciones diferenciales complejas dividiendo un problema en partes más simples denominadas elementos finitos.

⁹Un complejo celular o CW-complex es un tipo de espacio topológico utilizado en topología algebraica y geometría diferencial.

¹⁰Un simplex o k-simplex en geometría es el análogo en k dimensiones de un triángulo.

¹¹Un hipercubo o k-cube en geometría es el análogo en k dimensiones de un cuadrado.

sólida. Dichas estructuras incluyen vértices, aristas y caras y son las más comúnmente usadas en simulaciones quirúrgicas en tiempo real.

SOFA permite importar mallas para su uso en la simulación mediante el componente *MeshLoader*. Existen varios formatos de archivo de malla compatibles, algunos limitados por el tipo de información que pueden contener; lo que reduce sus aplicaciones pero permite archivos de menor tamaño. Pueden ser consultados en el apéndice E.

La importancia de la selección de una malla apropiada es crítica. Cuando un objeto se representa con mayor cantidad de componentes geométricos en la malla (vértices, aristas, caras, etc), su comportamiento físico será más realista pero el coste computacional también crecerá en consonancia. Será necesario un estudio comparativo preciso de la cantidad de componentes para lograr la mayor exactitud posible manteniendo una cantidad de fps aceptable para la simulación.

B.2. Estructura de una simulación

En SOFA, las simulaciones reciben el nombre de *scene graphs* y están compuestas por nodos organizados en una estructura de árbol o grafo acíclico dirigido. Cada objeto simulado en el *scene graph* está descrito en nodos separados, y cada modelo de representación de un objeto también puede ser realizado en diferentes subnodos.

El nodo principal del *scene graph* es denominado *root node*, y de este partirán una serie de subnodos, llamados *child nodes*, que representarán cada objeto y modelo de la simulación.

A modo de ejemplo, en la figura 1 se puede observar un único *child node* "Liver" que define al primer objeto. Este corresponde al modelo físico y contiene diversos componentes. Cada componente u objeto de un nodo define una característica concreta (e.g., masa), y podrá tener uno o más atributos que lo caractericen (e.g., densidad). El diseño altamente modular y escalable permite la independencia de cada uno de los componentes, pudiendo modificarlos individualmente sin afectar al resto.

Para continuar su desarrollo con otro ejemplo, en la figura 2 se puede observar un nuevo *child node* denominado "Visual", que corresponde a su respectivo modelo y contiene varios componentes para definirlo. La representación multimodelo permite la separación, en este caso, de los parámetros que representan el modelo físico y su comportamiento (*child node* "Liver"), y el modelo visual que describe la superficie del objeto (*child node* "Visual").

Para profundizar en una simulación más compleja con varios objetos, el último ejemplo expuesto en la figura 3 muestra dos objetos diferentes: un hígado y un corazón. Se puede apreciar que los *child nodes* de los modelos físicos tienen los mismos componentes, pero las clases utilizadas serán diferentes. Esto permitiría, por ejemplo, simular un modelo físico del hígado y los pulsos eléctricos del corazón.

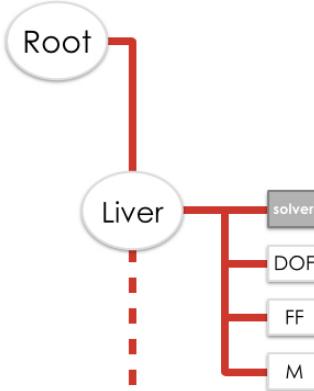


Figura 1: *Scene graph* con un único *child node*.

Como se ha mencionado, los diferentes componentes definidos en los *child nodes* poseen una serie de atributos cuyo valor puede ser asignado por el usuario. A su vez, existen diferentes clases que se pueden utilizar para cada componente. Por ejemplo, el componente que define la masa, representado por el *child node* "M", podrá ser de la clase *UniformMass*, si la masa del objeto representado es uniforme, o *DiagonalMass*, si la masa del objeto se distribuye en diferente proporción según la densidad de vértices en su geometría. Ambos tienen un atributo decimal denominado *totalMass* y, a mayores, *DiagonalMass* también posee otro denominado *massDensity*.

La construcción de un *scene graph* podrá realizarse mediante *scripts* en XML o Python, en los que cada nodo y componente podrá ser definido mediante una única línea. Para el uso de Python se requiere la integración del *plugin* SofaPython3.

B.3. Principales componentes de la simulación

- *AnimationLoop* es uno de los componentes imprescindibles en la creación de un *scene graph*. Se encarga de secuenciar las tareas necesarias en cada paso temporal de la simulación y llamar, mediante *Visitors*, a los componentes encargados de ellas, e.g., resolver las ecuaciones que definen las físicas (como el movimiento de un objeto al que se le aplica una fuerza) o la detección de colisiones. Al ser un componente esencial, SOFA lo creará por defecto utilizando la clase *DefaultAnimationLoop*. Otras de las clases disponibles podrán ser consultadas en el apéndice F.1.
- Los *Visitors* son herramientas diseñadas para interactuar y obtener información de los distintos componentes del *scene graph*, recorriendo su jerarquía mediante clases abstractas.
- El componente *CollisionPipeline* es el encargado de la detección, computación y resolución de colisiones a través de diferentes fases detalladas en el apéndice G. Es un componente esencial en la simulación y se ejecuta en cada paso temporal.

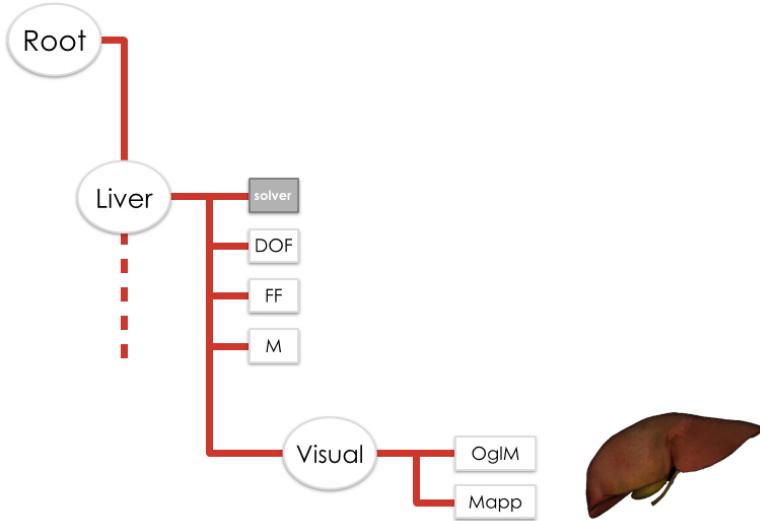


Figura 2: *Scene graph* en el que se pueden observar el modelo físico y visual de un objeto representados en los *child nodes* "Liver" y "Visual", cada uno con determinados componentes que los definen.

B.4. Modelo físico

El modelo físico o mecánico permite definir el comportamiento y las propiedades físicas del objeto que se simule. Tanto este como los demás modelos poseen un componente principal que almacenará la información de la estructura y geometría del objeto, denominado *MechanicalObject* en los modelos físico y de colisiones y *OglModel* en el modelo visual.

Dicho componente principal contiene los diferentes vectores de estado de los vértices que conforman la geometría, pudiendo asignar a cada modelo una geometría independiente que tenga mayor o menor cantidad de información geométrica y permita adaptar el coste computacional de cada modelo a los requisitos de la simulación.

Los vectores de estado serán representados por vectores de una o más dimensiones dependiendo del tipo de información que esté contenida en el modelo. Esto deberá ser indicado en el atributo *template* del *MechanicalObject*. En el caso que concierne a este proyecto, se asignará el valor *Vec3d* para indicar la representación tridimensional de un objeto. Se recogen en el anexo H los demás modelos que pueden ser utilizados y sus diferentes aplicaciones.

A mayores de los vectores de estado, es necesario indicar qué información geométrica se quiere usar para el modelo. Es posible que la malla contenga información geométrica de, por ejemplo, triángulos y tetraedros. El uso de polígonos sólo permitiría la simulación de la superficie del objeto, mientras que el uso de poliedros, la simulación del volumen. A este efecto, se seleccionará la clase del componente *TopologyContainer* correspondiente a la topología deseada, mientras que se deberá seleccionar congruentemente la del

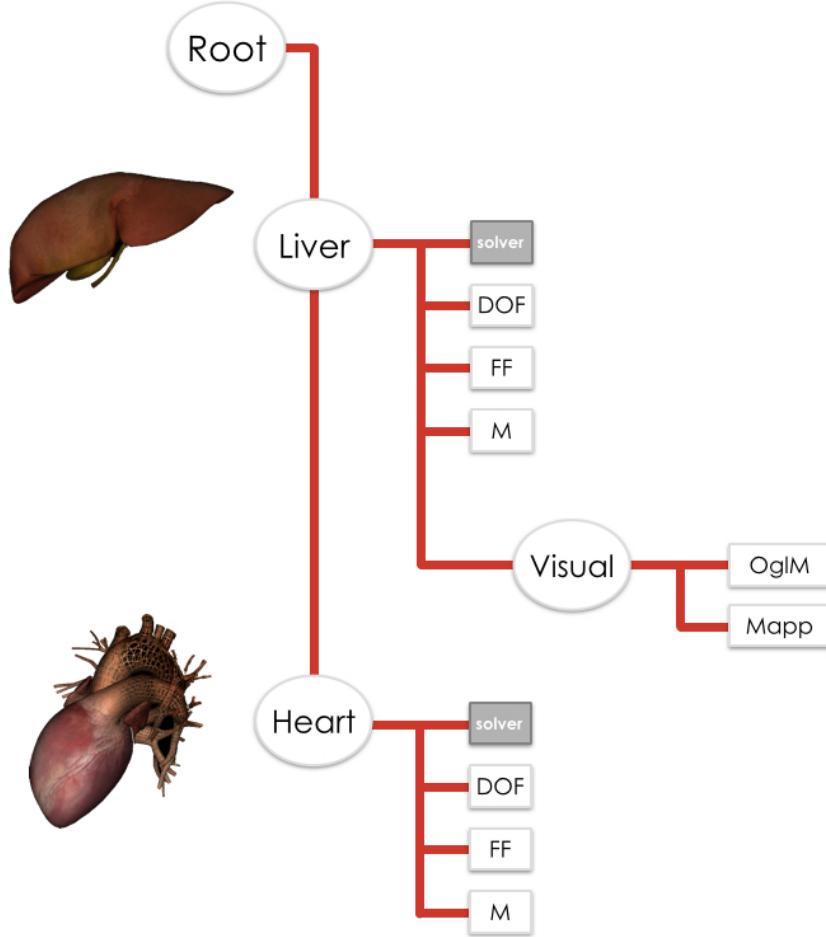


Figura 3: *Scene graph* en el que se simulan dos objetos distintos, cada uno con sus respectivos *child nodes*.

componente *GeometryAlgorithms*, que permitirá computar los cálculos que dependan y afecten a la geometría como el comportamiento visual. Por ejemplo, la clase *TriangleSetTopologyContainer* albergaría información de la superficie del objeto conformada por triángulos y debería utilizarse en consonancia la clase *TriangleSetGeometryAlgorithms* para la computación de los mismos.

Por otro lado, SOFA requiere de una serie de algoritmos que permitan computar la aplicación de fuerzas sobre un objeto y su respuesta ante ellas, dependiendo además de factores como su masa y sus propiedades elásticas. Para ello, una simulación en tiempo real usualmente requiere discretizar la evolución de todo el sistema, que será computado mediante un sistema de ecuaciones diferenciales ordinarias, en pequeños pasos temporales.

Los esquemas de integración son métodos numéricos que describen como obtener una solución aproximada para un sistema de ecuaciones diferenciales ordinarias, calculando la posición y velocidad de los objetos de la simulación en cada instante temporal mediante su

resolución. En SOFA, estos reciben el nombre de *ODESolvers* y permiten computar dichas ecuaciones en un sistema lineal matricial. Una vez obtenido dicho sistema, los componentes encargados de su resolución reciben el nombre de *linear solvers*. Existen diversas clases pero la mayoría son utilizadas para simulaciones muy específicas como la interacción de fluidos o la representación de temperatura. La mayoría de simulaciones de ejemplo de colisiones e interacciones de cuerpos rígidos y/o deformables utilizan los mismos *solvers* con pequeñas variaciones de sus parámetros y es recomendable la utilización de dichas clases para una simulación estable y eficiente.

Los aspectos mencionados hasta el momento han abarcado los mecanismos básicos de una simulación y aquellos componentes requeridos que permiten representar un objeto a partir de su información topológica. Para poder interactuar con el objeto, ya sea para modelar su comportamiento ante las fuerzas que se le apliquen o para estudiar su colisión con otros objetos, deberán tenerse en cuenta tres aspectos fundamentales:

- Para que una serie de fuerzas interactúen sobre un objeto, este debe tener definida una masa.
- Se pueden aplicar una serie de fuerzas externas para recrear la gravedad, empujar y mover el objeto por el entorno o mantenerlo fijo en un punto, ya sea fijando todo el objeto o sólo determinadas partes del mismo, entre otras acciones.
- En una simulación, un objeto puede ser deformable o rígido. En cualquier caso, dicho objeto deberá estar definido por una o más fuerzas internas que modelen su resistencia a ser deformado o roto, así como la capacidad del mismo de recuperar su posición inicial en el primer caso. Las clases disponibles pueden variar ampliamente según el tipo de maleabilidad, elasticidad, tenacidad y demás propiedades del objeto.

Los componentes *Mass* y *ForceField* permiten, respectivamente, asignar la masa y las fuerzas externas e internas que se pueden aplicar al objeto. Sus clases pueden ser consultadas en detalle en los apéndices F.2 y F.4.

B.5. Modelo de colisiones

A la hora de procesar las colisiones, el componente *collision pipeline* es el encargado de definir dicho proceso. Para una comprensión detallada del funcionamiento y las diferentes fases que lo componen es recomendable consultar el apéndice G, como ya se ha mencionado anteriormente.

Respecto al componente principal, *MechanicalObject*, no será necesario indicar esta vez su atributo *template*. Siguiendo los estándares de diseño de *scene graphs* propuesto por los desarrolladores del *framework*, se creará un *child node* por cada objeto que se deseé simular, representando éste su modelo físico, mientras que los modelos de colisiones y visual serán *child nodes* del modelo físico. En este caso, el modelo de colisiones usa

automáticamente el valor dado en el modelo físico para el atributo *template*, pues sería por defecto el único que podría usar.

Deberá declararse por otro lado el componente *MeshTopology*, que almacenará la información de la topología de la misma forma que lo hace el *TopologyContainer* en el caso del modelo físico. En este caso, no será necesario indicar mediante ninguna clase específica el tipo de información geométrica que contiene, pues esta labor pertenece a las clases del componente *CollisionModel*.

En el caso de querer simular colisiones, se debe indicar mediante el uso de una serie de primitivas denominadas *CollisionModels* qué información geométrica se tendrá en cuenta para la colisión, pues no requiere el mismo coste computacional el uso de, e.g., triángulos que vértices y, a su vez, tampoco se obtendrá la misma precisión. Sí es posible indicar más de una si se desea que el *framework* las utilice, siguiendo el orden declarado, para ajustar la precisión en los puntos de colisión coincidentes con las formas geométricas más complejas. El orden de declaración más apropiado a seguir son: poliedros, polígonos, aristas y vértices.

Por último, para poder conectar los diferentes modelos de los componentes simulados, será necesario crear algún tipo de enlace que permita la comunicación entre ellos. El modelo físico debe recibir información de colisiones del respectivo modelo y reaccionar en consecuencia en función de sus propiedades físicas, así como el visual y el de colisiones deben responder moviéndose en consonancia con el físico. SOFA dispone de un componente denominado *Mapping* que permite el intercambio de información entre diferentes modelos conectando los *MechanicalObject* de cada uno de ellos entre sí.

B.6. Modelo visual

El modelo visual es el más sencillo de los tres, encargándose únicamente del apartado visual de la simulación. Su objetivo es representar un objeto observable similar al real, y para ello es recomendable el uso de una malla con más componentes que la utilizada en los demás modelos, permitiendo una visión precisa del resultado de las interacciones de la simulación.

Posee dos componentes principales: el componente *Mapping*, cuyo objetivo y funcionamiento se ha detallado en el anterior apartado, y el *OglModel*.

El *OglModel* es, en realidad, el equivalente al *MechanicalObject* en el modelo visual y, mediante OpenGL¹², permite obtener de la malla determinados componentes visuales además de los vectores de estado, e.g., color, textura y sombreado. También posee atributos que permiten asignar dichas características directamente en el componente.

¹²Open Graphics Library.

Todos los componentes tendrán un atributo *name* que podrá ser asignado en su declaración, permitiendo identificar únicamente al componente dentro de la simulación y referenciarlo para acceder a sus atributos cuando sea preciso. Para el cumplimiento de buenas prácticas, por legibilidad del código y por necesidad en caso de que así sea, todos los componentes tendrán asignado un valor en dicho atributo.

C. Sección ampliada de parametrización del modelo

A continuación, se detallarán las clases seleccionadas para cada componente y sus atributos junto con su explicación detallada, justificando el motivo de su elección y uso. Dichos componentes han sido probados incorporándolos gradualmente a través de diferentes pruebas, detalladas en el apartado 5, para realizar las comparativas frente a otras posibles elecciones descartadas y observar su impacto en la eficiencia comportamental y computacional.

C.1. Entorno y detección colisiones

Para la creación del *script* que defina la simulación, se ha optado por implementar el uso de Python mediante el *plugin* SofaPython3. Su sintaxis más sencilla facilita la comprensión, escritura y modificación del código y, además, permite el uso de variables, funciones, estructuras de control y librerías externas que amplían las funcionalidades del mismo.

En primer lugar, el *root node* tiene varios atributos o variables que resultan de utilidad para establecer ciertos parámetros de simulación. Caben destacar:

- *dt*, variable de tipo *double* que permite indicar cuánto durará cada paso temporal de la simulación. Su valor afecta a la velocidad de la simulación y la precisión del comportamiento y deberá modificarse a lo largo de las diferentes pruebas para buscar el equilibrio adecuado.
- *gravity*, vector tridimensional que define una fuerza constante durante la simulación, usualmente para representar la gravedad en el eje y de coordenadas.

A su vez, las funciones *addComponent* y *addChild* de cada nodo nos permiten agregar sus respectivos componentes y *child nodes*.

Antes de comenzar a crear los *child nodes* que representen los modelos físico, visual y de colisiones del intestino, deberán asignarse al *root node* tanto los componentes base de la simulación, ya mencionados en el apartado 3.3, como los *plugins* y librerías de SOFA que se vayan a utilizar:

- Las librerías propias de las clases y componentes de SOFA están planteadas como *plugins* y, por tanto, deberán importarse de la misma manera. Ha de tenerse en cuenta que hay dos tipos de *plugins*: los primeros representan las librerías propias de cada componente que vienen incluidas en el *framework*, mientras que los segundos añaden

funcionalidades específicas (mejora del rendimiento, integración con dispositivos hapticos, etc.) al mismo y deben añadirse al *framework* previamente de manera específica¹³. Se añadirán como componentes utilizando la clase *RequiredPlugin* e indicando en el atributo *name* el nombre de la librería o *plugin* a importar en cuestión.

- Para el componente *AnimationLoop* se usará la clase genérica *DefaultAnimationLoop*, puesto que las demás se enfocan a simulaciones específicas como el uso de *multithreading* para simular múltiples colisiones o la simulación de bucles.
- Se creará el componente *CollisionPipeline* (el único que posee exclusivamente una única clase con su mismo nombre), y se utilizarán las clases genéricas para cada uno de sus componentes, puesto que el resto de clases han sido creadas para comportamientos muy específicos como el *multithreading*:
 1. *BruteForceBroadPhase* para la fase *BroadPhase*, encargada de detectar una colisión de forma temprana calculando las zonas de los objetos que puedan colisionar en algún momento próximo y las que no.
 2. *BVHNarrowPhase* para la fase *NarrowPhase*, cuya función es comprobar las zonas de posible colisión y detectar cuándo sucede la interacción.
 3. *DefaultContactManager* contendrá un atributo denominado *response* con el valor *PenaltyContactForceField*, indicando que durante el procesado de la colisión los objetos involucrados tendrán una pequeña fuerza de rechazo entre ellos para evitar que los modelos se atravesen entre sí y simular un contacto más realista.
 4. En la clase *MinProximityIntersection* se le asignará un valor numérico a los atributos *alarmDistance* y *contactDistance*, donde el primero indicará a partir de qué distancia deberá empezar a procesarse la colisión y el segundo la distancia a la que se considera que habrá contacto (dos objetos nunca colisionarán de manera perfecta, a pesar de que así deba apreciarse visualmente).

El comportamiento de esta *CollisionPipeline* funciona de tal manera que, a medida que los dos objetos se aproximen, irá detectando si están en ruta de colisión teniendo en cuenta la velocidad y dirección de las fuerzas y, una vez estén próximos al contacto, computará la respuesta teniendo en cuenta los valores indicados en los atributos de *MinProximityIntersection*.

C.2. Selección de la malla

Respecto a la selección de la malla, teniendo en cuenta que este proyecto no abarca el ámbito del modelado 3D, se ha optado por la búsqueda en la web de diferentes modelos

¹³Pueden consultarse en el siguiente enlace los *plugins* adicionales disponibles junto con su documentación e instrucciones de instalación: SOFA plugins.

de intestino disponibles, usando finalmente aquel aportado por uno de los tutores dado la falta de modelos gratuitos precisos y sin errores en la topología.

Para llevar a cabo un estudio adecuado, sí resulta necesaria la manipulación del modelo aportado con el objetivo de realizar pruebas de eficiencia con geometrías de distinta cantidad de elementos (vértices, aristas y caras). Con este fin se ha usado el *software* Blender¹⁴ para reducir dicha cantidad de elementos en diferentes proporciones y obteniendo distintas mallas de prueba.

La malla elegida para el modelo visual contiene una mayor cantidad de elementos que la usada en los demás modelos. Esto es así porque el coste computacional en dicho modelo es mucho menor en comparación con los otros, permitiendo un resultado preciso en cuanto al apartado visual. El formato elegido para esta malla es obj, ya que su utilidad está enfocada a la visualización 3D de los objetos.

La malla seleccionada para el modelo físico y de colisiones ha sido la misma. Esto es así ya que carece de sentido el uso de una malla diferente en ambos: la detección de una colisión ubicada en determinados componentes, e.g., vértices, que no se encuentran en el modelo físico o viceversa puede provocar que el objeto se comporte de una manera irreal durante la simulación. El formato elegido en este caso es msh, ya que es el más eficiente para la representación de volúmenes con mayor densidad de componentes geométricos.

Se usarán las clases *MeshObjLoader* y *MeshGmshLoader* para importar los archivos obj y msh correspondientes. La ruta del equipo donde se encuentra el archivo se indicará en el atributo *filename*.

C.3. Selección y parametrización de los componentes

Una vez importada la malla del intestino, debe decidirse qué información va a ser usada para modelar su topología. Si se busca que el objeto contenga un volumen y no este representado únicamente por su superficie, las opciones más apropiadas son tetraedros o hexaedros. En el caso de un objeto con una forma compleja, como el que atañe a este proyecto, el uso de tetraedros es más apropiado.

Para definir el modelo físico del intestino con la geometría mencionada se utilizarán las siguientes clases de los componentes anteriormente detallados y se asignarán, de ser necesario, los atributos apropiados:

- Entre los *ODESolvers* disponibles, *EulerImplicitSolver* es el más usado habitualmente por su equilibrio entre estabilidad y bajo coste computacional. Está enfocado a obtener una simulación fluida en entornos que puedan tener cambios rápidos en breves instantes de tiempo, como escenarios con colisiones entre cuerpos rígidos o deformables. Sus principales atributos que deben definirse apropiadamente son:

¹⁴Blender es un conjunto de herramientas de creación y modelado 3D gratuito y de código abierto. Web oficial: blender.org

1. *rayleighMass*: Regula la disipación de fuerzas asociada a la velocidad, permitiendo que los objetos se comporten de manera realista simulando la pérdida de energía a lo largo del tiempo.
 2. *rayleighStiffness*: Regula la disipación de fuerzas asociada a la rigidez del objeto simulando la pérdida de energía debida a la fricción con el aire y otros elementos, evitando que los objetos oscilen o se muevan de manera irrealista.
- La colección de *linear solvers* dispone de una serie de clases entre las que destaca, por su uso enfocado a entornos con colisiones y objetos con un comportamiento complejo como tejidos biológicos, la clase *CGLinearSolver* como la más apropiada para el objetivo de este proyecto. Esta clase intenta resolver de una manera aproximada el sistema de ecuaciones, alcanzando un resultado aproximado a lo largo de una serie de iteraciones. Sus principales atributos a definir son:
 1. *iterations*: Define la cantidad de iteraciones de cada paso temporal. A mayor cantidad de ellas, el sistema tendrá un realismo más preciso pero la velocidad podrá verse perjudicada si son excesivas. El margen de valores habitual suele estar en el rango de 5 a 30 iteraciones.
 2. *threshold*: Determina si las iteraciones deben terminar cuando sobrepasan un determinado margen de error. Si el *solver* encuentra una situación en la que el resultado obtenido es menos preciso en cada iteración, detiene el proceso. Los valores más apropiados están entre los margenes 10^{-3} y 10^{-15} .
 3. *tolerance*: Indica el margen de precisión que debe obtenerse en cada iteración. A mayor precisión, el resultado será más óptimo pero se requerirá mas cantidad de iteraciones. Sus valores se encuentran en los mismos margenes que el *threshold*.
 - *MechanicalObject*: Se asignará, como ya se ha mencionado, el valor *Vec3d* al atributo *template* para indicar que la información representa las coordenadas tridimensionales de los vectores de estado del objeto.
 - *TetrahedronSetTopologyContainer* contendrá la información de los vectores de estado uniéndolos para crear figuras tetraédricas que conformen el volumen del intestino. Deberá indicarse en el atributo *src* el nombre identificativo del componente *MeshLoader* con el que está asociado.
 - De forma congruente, *TetrahedronSetGeometryAlgorithms* indicará que los algoritmos computados en la simulación deberán considerar que el objeto estará conformado por tetraedros.

Cabe mencionar que en los componentes *MechanicalObject* y *OglModel* de los tres modelos es posible manipular la escala, rotación y traslación del objeto en los ejes mediante los atributos *scale*, *rotation* y *translation*. Por defecto, la posición y orientación del objeto está definida en la información del archivo importado y puede modificarse mediante los atributos indicados.

A la hora de asignar la masa, la elección de la clase *DiagonalMass* es la más equilibrada en cuanto a precisión y eficiencia y tiene en cuenta la densidad del objeto para una distribución de masas no uniforme, por lo que será la más adecuada. Permite asignar, alternativamente, la masa o la densidad del objeto mediante los atributos *totalMass* y *massDensity*.

Resta, para el modelo físico, entender qué parámetros modelan las propiedades elásticas asociadas a la deformación del intestino. A este efecto, se pueden encontrar estudios y artículos, como el publicado en Journal of Biomechanics sobre las propiedades biomecánicas de diferentes órganos [21], que parametrizan dichas propiedades mediante los siguientes parámetros:

- Módulo de Young: indica la capacidad de un material de ser deformado de manera longitudinal al aplicar una tensión sobre su superficie en una dirección determinada. En otras palabras, define la rigidez del material.
- Coeficiente de Poisson: define la capacidad del material de expandirse o contraerse en dirección perpendicular a la deformación provocada por la aplicación de una tensión sobre su superficie.

Estas características permiten modelar dicho comportamiento elástico mediante la clase *TetrahedronFEMForceField* asignándole los valores correspondientes mediante los atributos *youngModulus* y *poissonRatio*. También debe indicarse el valor *Vec3d* en su atributo *template* y es posible, a mayores, indicarle que use mayor o menor precisión a la hora de computar los decimales de los cálculos asignándole el valor *large* en el atributo *method*.

Respecto a los modelos de colisiones y visual, el componente *Mapping*, ya mencionado en las secciones 3.5 y 3.6 , dispone de dos únicas clases para utilizar con unos casos de uso bien diferenciados: *RigidMapping*, para la simulación de cuerpos rígidos, y *BarycentricMapping*, para la simulación de cuerpos deformables. En el caso que abarca el proyecto, se utilizará la segunda.

Queda finalmente, para acabar de definir los modelos visual y de colisiones, asignar las clases correspondientes a las primitivas del modelo de colisión. Se elegirá la clase *TriangleCollisionModel*, que utilizará las caras triangulares de los tetraedros que coincidan con la superficie del objeto para la detección. Además, en cualquier primitiva se podrá definir un atributo *contactStiffness* que determine la rigidez de contacto, permitiendo ajustar la fuerza que ambos objetos ejercen entre sí al contactar para alcanzar el nivel de realismo deseable.

Finalmente, la estructura del *scene graph* se ha concluido y las distintas clases a utilizar han sido seleccionadas. Aun así, deberán añadirse una serie de componentes a mayores a la hora de implementar los siguientes aspectos:

1. El colonoscopio, o cualquier componente de la simulación que se deseé manipular deberá poder moverse manualmente de alguna forma. Para este efecto, se pueden crear los siguientes componentes:
 - La creación de una clase *ConstantForceField*, que permite asignar mediante su atributo *totalForce* la fuerza constante aplicada a cada eje de un objeto.
 - Gracias al uso de python, podrá implementarse una clase que acceda a los atributos de cualquier componente en tiempo real durante la simulación, permitiendo manipular los valores del componente creado.
 - Ha de tenerse en cuenta que, a la hora de mover el objeto, la inercia podrá dificultar su manipulación mediante el teclado debido a la falta de fuerzas que simulen el rozamiento con el aire. Por ello, será necesario para frenar el objeto acceder a la velocidad del mismo y reducirla a cero. Este valor es accesible a través del atributo *velocity* del *MechanicalObject*.
2. El intestino deberá mantenerse fijo en su posición, ya que en un entorno natural el resto de órganos y estructuras del cuerpo ejercen una serie de fuerzas que así lo mantienen. Además, esto también deberá implementarse dada la capacidad del intestino de volver a su estado de reposo tras sufrir deformaciones. Para ello, se debe utilizar la clase *RestShapeSpringsForceField*, la cual permite aplicar una fuerza que mantenga al objeto fijo en el espacio, representando su atributo *stiffness* la fuerza con la que el objeto regresa a su estado de reposo., y el atributo *angularStiffness* el comportamiento mencionado ante deformaciones de torsión.
3. Deberá eliminarse la fuerza de la gravedad de la simulación, ya que queremos manipular uno de los objetos directamente, y deberá ser asignada dicha gravedad al intestino mediante la clase *ConstantForceField* con los valores correspondientes para que su comportamiento mantenga el realismo de sus condiciones naturales.
4. Si se desea implementar una vista desde la perspectiva de un objeto, simulando una cámara asociada a la posición del mismo, puede hacerse a través de la clase *OglViewport*. Por una parte, sus atributos *screenPosition* y *screenSize* permitirán definir el tamaño y la posición en la pantalla de simulación de la pantalla secundaria con la nueva vista. Por otra, el atributo *cameraOrientation* definirá la orientación inicial de la vista y el atributo *cameraPosition* referenciará la posición del *MechanicalObject* al que está asociada. Se ha añadido también la posibilidad de modificar la orientación de la cámara en tiempo real de simulación mediante el teclado.
5. Por último, para aumentar la precisión de realismo del intestino, habrá de tenerse en cuenta que sus propiedades elásticas no son idénticas en todas sus secciones, teniendo diferentes valores del módulo de Young y del coeficiente de Poisson. Para solucionar esto, es posible añadir un componente de la clase *BoxRoi* que permite crear una caja ficticia y manipular exclusivamente la porción del volumen del objeto encerrado en dicha caja, asignándole a cada una diferentes *TetrahedronFEMForceField*. A

este efecto, la clase *BoxRoi* contendrá el atributo *box* en el cual se indicarán las dimensiones de la caja. Deberá indicarse, además, el componente *MechanicalObject* al que está asociada mediante el atributo *position*. También deberá indicarse la *BoxRoi* asociada a cada *TetrahedronFEMForceField* mediante el atributo *initialPoints*.

C.4. Escala y unidades de medida

SOFA no utiliza unidades de medida por defecto para ningún parámetro, ni si quiera para evaluar la escala del modelo que se está usando. En consecuencia, todos los valores del modelo deben ser congruentes entre sí. Por ejemplo, si la escala del objeto está representada en centímetros en lugar de metros, la densidad de masa debe ser asignada en kg/cm^3 , y la fuerza de la gravedad en cm/s^2 . Esto debe ser así para todos los valores asignados.

Por otra parte, asumir una escala real de longitud no siempre es óptimo en la simulación. Esto podría provocar errores de convergencia en el *linear solver* al usar valores numéricos demasiado elevados o pequeños, pudiendo llevar también a aproximaciones inexactas o incluso errores en la simulación. Tal es el caso de valores como el módulo de Young, que usualmente opera con valores en el margen de varios mega o gigapascales.

Dado lo mencionado hasta ahora, cabe mencionar que la naturaleza de SOFA implica generalmente una gran cantidad de ensayos de prueba y error tratando de ajustar determinados parámetros como la masa, el paso temporal o la aplicación de fuerzas en la simulación; partiendo de un valor aproximado al real y ajustando valores y escalas.

C.5. Valor de los parámetros

Al margen de la problemática que se plantea en el anterior apartado, sí resulta necesario partir de unos valores apropiados y que se aproximen a la realidad. Para ello se han obtenido los valores medios de los siguientes parámetros: un módulo de Young y coeficiente de Poisson de $2,63 MPa$ y $0,33$ [21] y una densidad de masa de $1088 kg/m^3$ [22]. También se ha comprobado que, al margen de la escala final, la malla del modelo respeta la proporción de las dimensiones de las distintas secciones del intestino [23]. Por otra parte, como se había mencionado en el apartado 4.3, pueden definirse de manera más precisa y al margen de su valor medio los parámetros del módulo de Young y el coeficiente de Poisson para las diferentes secciones del intestino [24].

Otros valores que afectan a la simulación pero son intrínsecos del escenario son el *dt* y los parámetros de los *solvers*. Para asignarles un valor en el margen correcto, se han usado como base varios ejemplos de simulaciones disponibles en el GitHub de SOFA¹⁵ que modelan un hígado. El objetivo principal de las pruebas recae en el ajuste preciso de estos parámetros para simular un comportamiento real. De esta manera, tras realizar

¹⁵SOFA GitHub web

las pruebas que se mencionarán a continuación, los valores que han demostrado ser más estables son:

- Un paso temporal de 0,4.
- Unos valores de 0,5 y 0,3 para los atributos *alarmDistance* y *contactDistance* de la clase *MinProximityIntersection*.
- Un valor de 0,5 para los atributos *rayleighStiffness* y *rayleighMass* de la clase *EulerImplicitSolver*.
- Una cantidad de iteraciones entre 15 y 20 junto con un valor tanto de *tolerance* como de *threshold* de 10^{-6} para el *CGLinearSolver*.

D. Desarrollo completo de las pruebas realizadas

Una vez realizado el estudio de las clases más adecuadas para los componentes de la simulación y su adecuada parametrización, cabe realizar las pruebas pertinentes para verificar que las clases seleccionadas son las más adecuadas. Se observará el comportamiento y rendimiento de diferentes clases, el impacto del ajuste de los parámetros más relevantes y el efecto de usar diferentes mallas en la simulación.

Como punto de partida se ha utilizado uno de los ejemplos de libre uso puestos a disposición por parte de los desarrolladores, como ya se ha mencionado en el apartado 4.5.

Por otro lado, para poder comprobar el rendimiento de los componentes durante una simulación, el *framework* dispone de una herramienta que plasma en un gráfico circular el porcentaje del tiempo de simulación total dedicado a procesar cada uno, destacando aquellos que implican un mayor esfuerzo computacional. También es posible observar dicho gráfico exclusivamente para el paso temporal que más esfuerzo ha requerido, ya que puede haber instantes críticos en la simulación que impliquen un coste mayor, como aquel en el que se produce una colisión.

En función de los gráficos, los fotogramas por segundo y el comportamiento observable, se realizarán las comparativas en las pruebas que se realicen para concluir cuales comprometen en mayor medida al rendimiento y cuales logran un comportamiento más realista, buscando el equilibrio que permita la mayor eficiencia posible.

D.1. Simulación inicial

- Descripción: Ejemplo de simulación proporcionada por los desarrolladores del *framework* usada de modelo base para iniciar el proyecto y las pruebas por sus similitudes con el mismo. Muestra la colisión entre un hígado y una esfera, aplicando

una fuerza al primero que le mantiene sujeto a su posición inicial mientras que la segunda cae por gravedad. Las principales características que definen la simulación son:

- El entorno está definido con un paso temporal de 0,01 mientras que la gravedad no se aplica a toda la simulación si no que se asigna exclusivamente a la esfera mediante una fuerza constante a través de la clase *ConstantForceField*, con un valor de $[0, -1, 0]$.
- Para evitar que el hígado se desplace tras el impacto, se usa la clase *FixedConstraint* para fijarlo a su posición inicial. Dicha clase permite fijar los vértices indicados del objeto a través del atributo *indices*¹⁶. En este caso, están fijados únicamente los vértices de los extremos del eje longitudinal del hígado.
- El *AnimationLoop*, los *solvers* y los componentes que definen las fases del *CollisionPipeline* han sido comentados anteriormente y se realizarán algunas pruebas más adelante con el objetivo de mejorar su impacto en el rendimiento.
- La esfera esta compuesta por un modelo físico carente de topología que representa un objeto rígido. Como en cualquier caso que se modela un objeto no deformable, no sólo carece de sentido si no que no es posible añadir información topológica asociada, ya que esta es usada exclusivamente en la simulación para conformar la capacidad de deformación de un objeto y, por tanto, el propio *framework* no soporta dichos componentes en objetos rígidos.
- A la hora de definir un objeto rígido, asignando el valor *Rigid3d* al atributo *template* del *MechanicalObject*, este es conformado por un vector tridimensional, que indicará su posición en los ejes, y un cuaternión, que indicará la orientación del mismo; siendo asignados ambos en el atributo *position*.
- Su masa está definida mediante la clase *UniformMass*, con un valor de *totalMass* de 1.
- El modelo de colisiones es esférico, permitiendo ajustar el tamaño de la propia esfera asignándole el radio en dicho componente mediante un atributo.
- El hígado es modelado como un objeto volumétrico compuesto por tetraedros. Posee una densidad de valor 1 asignada mediante la clase *DiagonalMass*, y su elasticidad está modelada por la clase *TetrahedronFEMForceField* con unos valores de coeficiente de Poisson y módulo de Young de 0,4 y 1000, respectivamente. Su modelo de colisiones está definido mediante triángulos.
- No han sido implementados modelos visuales en esta simulación.
- Resultados: La simulación muestra una colisión suave, en la que la esfera apenas se hunde en el hígado y rebota levemente para seguir cayendo por la gravedad aplicada,

¹⁶La información de una malla también contiene una enumeración de cada componente topológico de la misma, pudiendo acceder a las propiedades de cada uno (posición, velocidad, masa, etc.) a través de dicho índice. SOFA puede mostrar su numeración mediante una opción en sus ajustes.

como se aprecia en las figuras 4 y 5. La cantidad de fotogramas por segundo es bastante alta, mayor a 100.



Figura 4: Prueba 1 - Momento de la simulación previo a la colisión.



Figura 5: Prueba 1 - Paso temporal posterior al momento de la colisión.

Como se puede observar, la gráfica del tiempo total de la simulación en la figura 6 muestra que los componentes *EulerImplicitSolver* y *BarycentricMapping* requieren un mayor coste. Es posible concluir que el resultado es razonable, ya que el primero debe procesar los elementos de la simulación para conformar el sistema de ecuaciones

y el segundo se encarga de la comunicación entre el modelo físico y de colisiones para procesar la respuesta a la colisión de manera adecuada en el modelo físico.

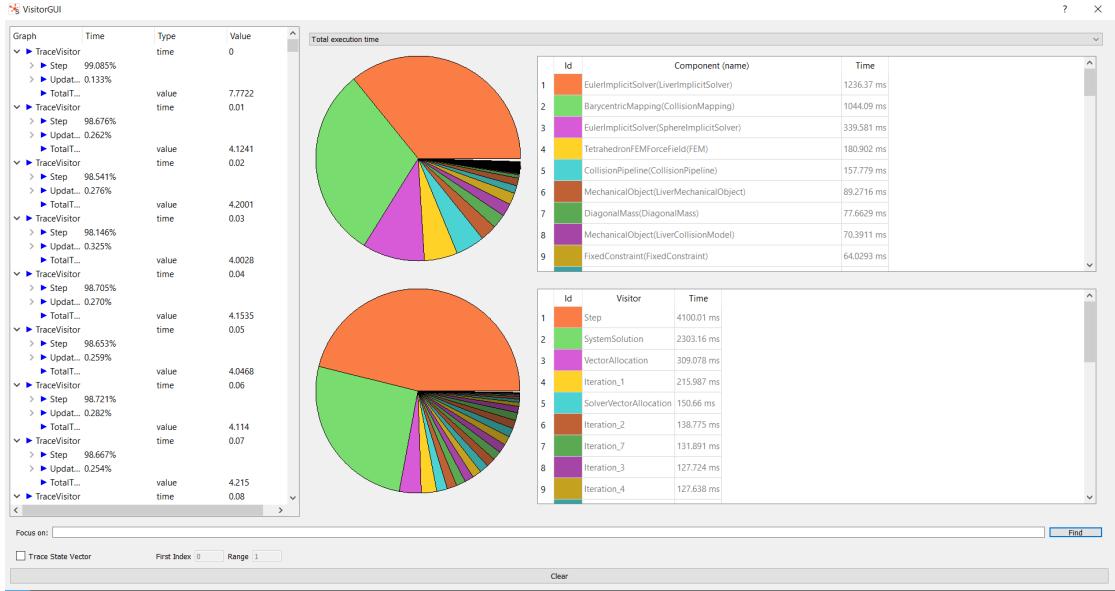


Figura 6: Prueba 1 - Media del coste computacional de cada componente en el tiempo total de simulación.

En el paso temporal de mayor coste, visualizando la figura 7, podemos ver como aumenta proporcionalmente el coste de conformar el sistema de ecuaciones. Cabe destacar que hay un *solver* independiente para cada componente de la simulación en lugar de uno global, siendo el mencionado hasta ahora el relativo al hígado. Más adelante se ahondará en ello en la prueba específica.

D.2 Uso de diferentes escalas en la simulación

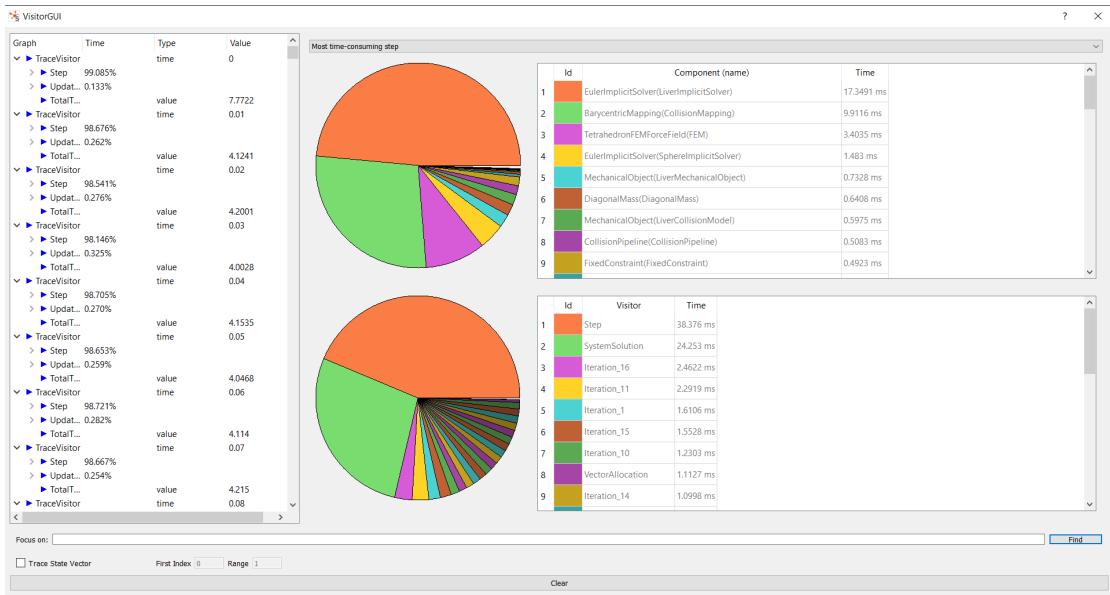


Figura 7: Prueba 1 - Coste computacional de cada componente en el paso temporal de mayor carga.

- Conclusiones: Es un punto de partida razonablemente estable en términos de eficiencia. A pesar de que el modelado de la esfera carece de similitud con un objeto real, sirve de base para establecer un entorno estable ante este tipo de colisiones con un cuerpo volumétrico deformable.

D.2. Uso de diferentes escalas en la simulación

Al intercambiar las mallas del hígado por las del intestino, se puede percibir una gran diferencia en términos de escala entre el intestino, cuya malla posee unas dimensiones definidas dentro de los márgenes de su tamaño real [23], y la esfera, cuyo radio se ha mantenido a 1. Además, midiendo las dimensiones de la malla mediante una herramienta externa como Blender, ya que SOFA no permite hacerlo, se comprueba que la malla utilizada para el hígado estaba escalada a un tamaño bastante menor al real, con un diámetro transversal de apenas 10mm.

- Descripción: Se procederá a comprobar el efecto de aplicar diferentes escalas al intestino, observando las diferencias expuestas y escalando el radio de la esfera de manera proporcional a la que presentaba respecto al hígado en la simulación de prueba. En concreto, se realizará la prueba con el propio tamaño definido para la malla, con una dimensión diez veces mayor y con una dimensión diez veces menor. Ha de tenerse en cuenta que cualquier dimensión que dependa de la longitud, como la densidad o el módulo de Young, deberá escalarse de manera proporcional para que las condiciones del entorno sean idénticas.

- Resultados: Sin modificar los parámetros físicos, que todavía corresponden al hígado, se observa que la colisión no provoca ninguna respuesta en el intestino, saliendo la esfera rebotada como si colisionase con un cuerpo totalmente rígido. Al aumentar o disminuir la escala, modificando los parámetros de manera proporcional, el comportamiento y recorrido de la esfera es totalmente idéntico salvo por su velocidad y la distancia de contacto. Se puede observar el momento de la colisión en la figura 8, donde se aprecia que el intestino no muestra deformación alguna.



Figura 8: Prueba 2 - Paso temporal posterior al momento de la colisión.

En el caso de la velocidad de la esfera, se observa que aumenta a menor escala y disminuye a mayor de manera proporcional a la escala; mientras que a menor escala la colisión se produce sin que ambos objetos lleguen a hacer contacto. La distancia de contacto, que se definía en el componente *MinProximityIntersection* de la *CollisionPipeline*, tiene un valor de 2,5 que no ha sido escalado para esta prueba y se observará más adelante el efecto de modificar tal valor en la prueba correspondiente a la *CollisionPipeline*.

Por otro lado, se pueden observar en las gráficas correspondientes a las figuras 9 y 10 un incremento notorio en el coste del componente que define el modelo elástico, obteniendo una cantidad de casi 80 fotogramas por segundo. Este resultado es notoriamente menor en comparación con la anterior prueba debido al aumento de carga computacional exigido por una malla con bastantes más elementos. Las gráficas son similares en las tres simulaciones. Podrán encontrarse en el anexo I la cantidad de componentes que conforman las mallas utilizadas a lo largo de todas las pruebas.

D.2 Uso de diferentes escalas en la simulación

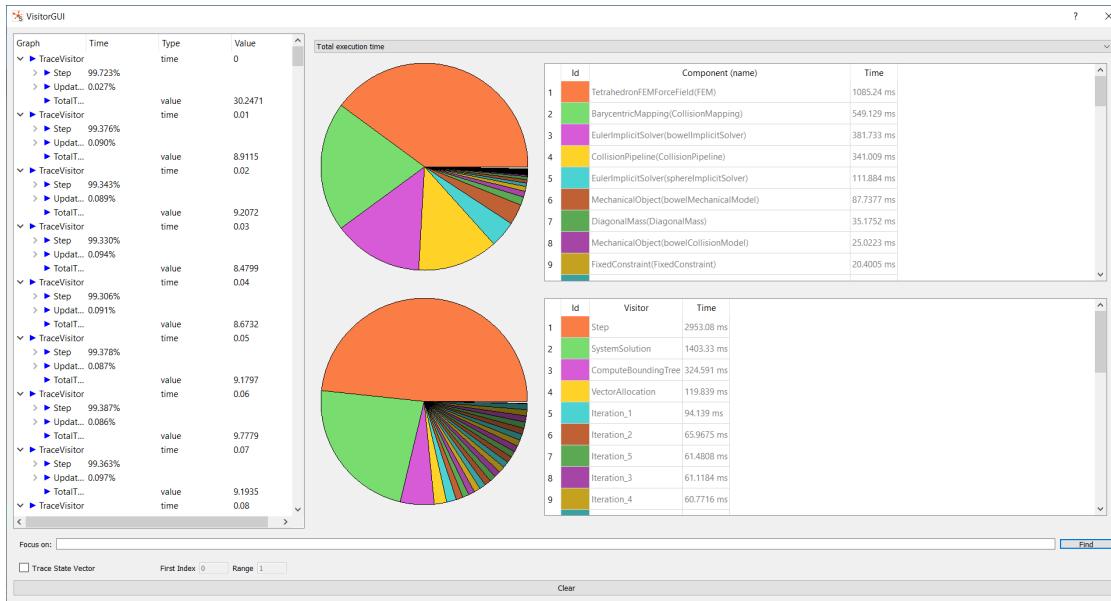


Figura 9: Prueba 2 - Media del coste computacional de cada componente en el tiempo total de simulación.

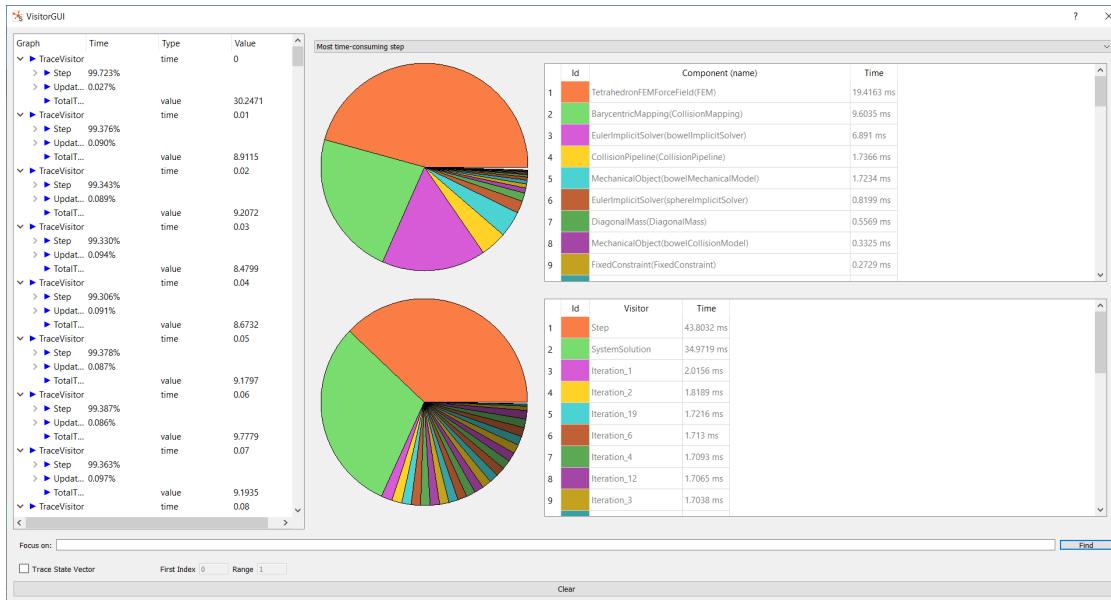


Figura 10: Prueba 2 - Coste computacional de cada componente en el paso temporal de mayor carga.

- Conclusiones: Se ha podido comprobar como el *framework*, siendo adimensional, requiere que los parámetros de medida sean asignados congruentemente. De esta forma, si todos los parámetros son escalados de manera acorde, el comportamiento de la simulación parecería ser el mismo. En la siguiente prueba se ajustarán los

parámetros del intestino para una correcta deformación y se repetirá para las tres escalas, pudiendo confirmar el efecto de la escala de manera más precisa, ya que en este caso sólo podían observarse los efectos viendo que la trayectoria de la esfera tras la colisión era idéntica en todas las escalas.

El comportamiento adimensional implica que, al escalar toda la simulación, la esfera tenga que desplazarse una distancia también escalada proporcionalmente, por lo que necesitará incrementar la fuerza aplicada a la misma para compensarlo.

La distancia de contacto, que se definía en el componente *MinProximityIntersection* de la *CollisionPipeline*, tiene un valor de 2,5 que no ha sido escalado para esta prueba y se observará más adelante el efecto de modificar tal valor en la prueba correspondiente a la *CollisionPipeline*.

D.3. Parametrización del intestino

- Descripción: A continuación, se asignarán los parámetros adecuados al intestino para su correcta deformación y se observará de nuevo el efecto de ajustarlos a diferentes escalas. Ya mencionados los valores de dichos parámetros en el apartado 4.5, deben ajustarse adecuadamente a la escala simulada: decímetros, centímetros o milímetros.

Para solventar el efecto de la velocidad a diferentes escalas, se aplicará una fuerza escalada también diez veces mayor y diez veces menor para que la velocidad de la esfera en las tres simulaciones sea idéntica. Ya que la fuerza usada en la simulación inicial era de valor 1 y la escala del hígado estaba en milímetros, se usarán valores de 1, 10 y 100 de menor a mayor escala.

- Resultados: Se puede observar en la escala real, con una magnitud de longitud de la malla de centímetros, que el intestino se deforma ligeramente como en el caso del hígado. Además, la sección en la que colisiona la esfera, la transversal en este caso, es desplazada ligera y lentamente tras la colisión debido a que no está sujeto apropiadamente a una posición fija, como se puede observar en las figuras 11 y 12.

D.3 Parametrización del intestino

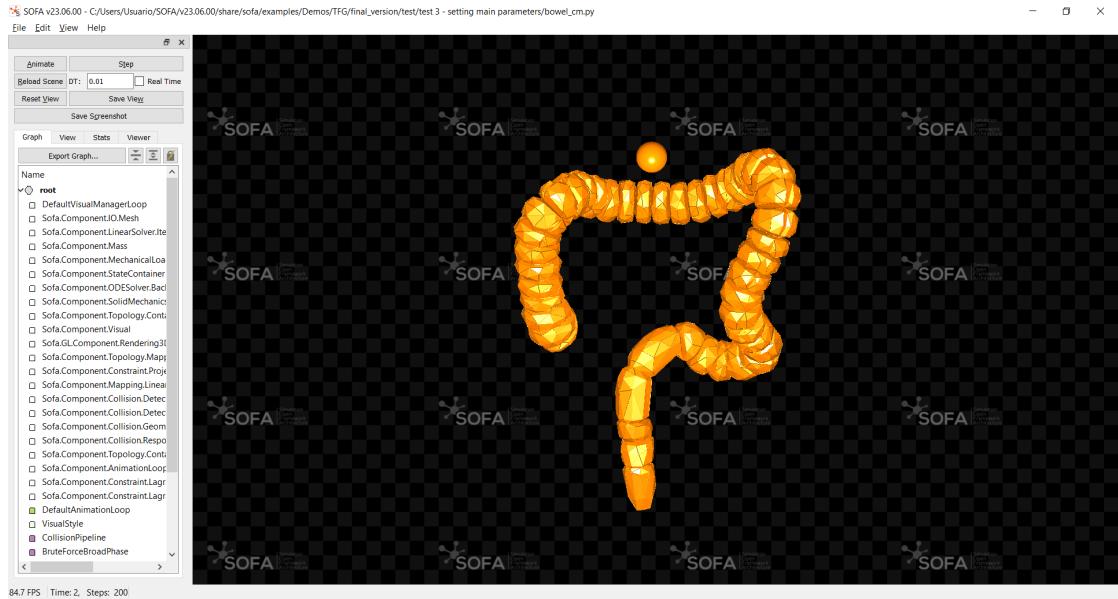


Figura 11: Prueba 3 - Momento de la simulación previo a la colisión.



Figura 12: Prueba 3 - Varios pasos temporales después del instante de la colisión.

Por otro lado, la misma simulación escalada a decímetros ya no parece comportarse de manera idéntica si no que el intestino se muestra rígido igual que en la prueba anterior.

La simulación a escala milimétrica muestra una colisión irreal en la que la esfera colisiona de forma excesivamente fuerte provocando que el intestino se deforme de manera inadecuada, como se puede apreciar en la figura 13.

D.3 Parametrización del intestino

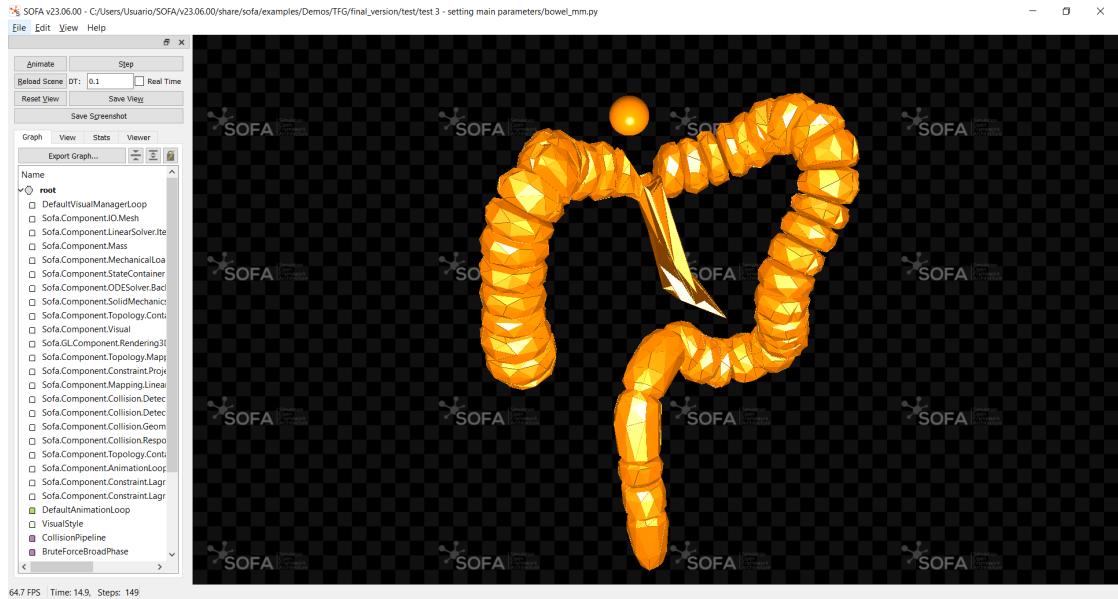


Figura 13: Prueba 3 - Respuesta anómala de la colisión en escala milimétrica.

Las gráficas muestran un rendimiento idéntico a la prueba anterior a excepción de la gráfica a escala milimétrica, cuyo tiempo total de simulación, que se puede apreciar en la figura 14, muestra una distribución de componentes errática en la que se observa que un mismo componente llega a estar duplicado, siendo esto causado por una simulación inadecuada.

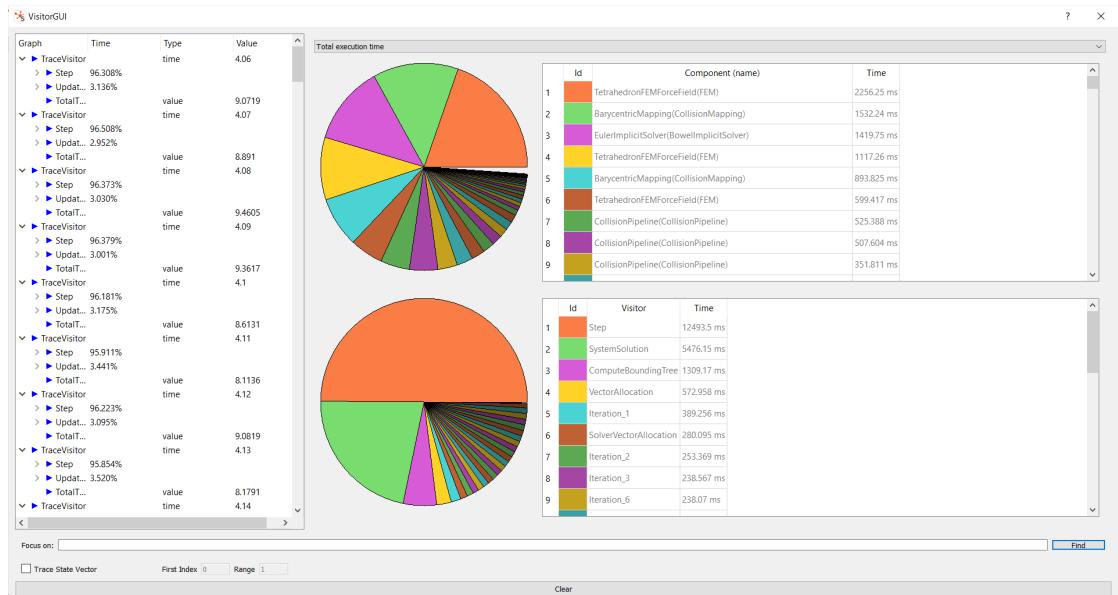


Figura 14: Prueba 3 - Gráfica del tiempo total de simulación para la escala en milímetros.

- Conclusiones: Se puede apreciar que, como se mencionó en el apartado 4.4, no

todas las escalas son adecuadas y pueden ocasionar problemas debido al propio funcionamiento del *framework*. Por ahora, se concluye que la más adecuada es aquella que se encuentra entre las dimensiones medias de un intestino.

D.4. Ajuste de parámetros

Antes de proseguir, cabe analizar el efecto de algunos parámetros que puedan solventar el mal funcionamiento de las dos simulaciones escaladas con respecto a la original.

Por una parte, es posible que una fuerza excesiva en la escala pequeña o una fuerza demasiado leve en la mayor afecten a la colisión. Esto podría intentar resolverse manteniendo la fuerza original y ajustando el paso temporal para obtener una velocidad similar.

Como ya se ha mencionado en la sección 4.3, el parámetro *contactStiffness* puede ser de utilidad a la hora de ajustar la fuerza del impacto de una colisión. Puede que sea posible que, modificando su valor por defecto de 100, se logre un efecto diferente que se adecúe más a la simulación escalada en centímetros.

- Descripción: Se intentará por ultima vez, modificando el paso temporal y el atributo *contactStiffness* de las primitivas de colisiones, el ajuste de las escalas en decímetros y milímetros como último método para lograr un comportamiento análogo a la simulación en centímetros.
- Resultados: Tras realizar la prueba con varios valores, el resultado obtenido para la escala en decímetros ha sido idéntico al anterior. Por otro lado, la simulación en milímetros muestra una colisión mucho menos rígida que en centímetros, llegando a hundirse la esfera en el intestino y desplazándolo una distancia considerable, tal y como se aprecia en la figura 15.

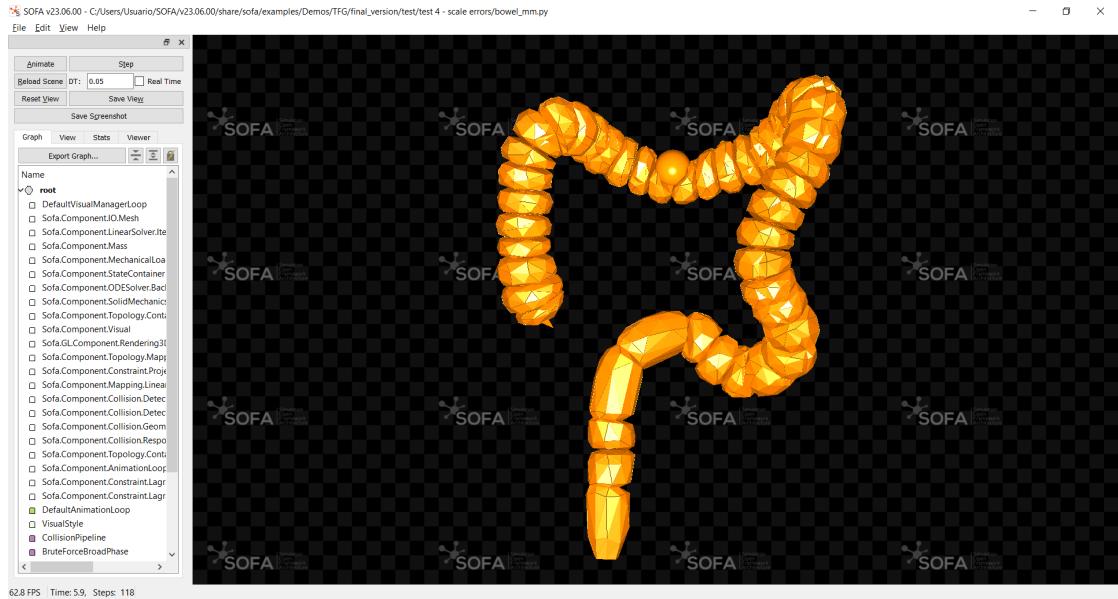


Figura 15: Prueba 4 - Resultado de la colisión a escala milimétrica.

No hay diferencias notorias en las gráficas en esta prueba, son similares a las obtenidas en la simulación a escala original.

- Conclusiones: No es posible ajustar adecuadamente las escalas con una longitud demasiado elevada, del orden de decenas de decímetros. Esto puede deberse a la incapacidad de computar parámetros con un valor excesivamente alto, pudiendo solucionarse de manera sencilla disminuyendo la escala.

Por otra parte, sí resulta eficaz la modificación de determinados atributos del entorno a la hora de estabilizar una simulación a pequeña escala. Además, según aumenta la escala las colisiones resultan en una interacción más rígida a pesar de modificar los parámetros precisos de manera congruente, por lo que el objetivo de lograr unas deformaciones realistas resulta más complejo.

D.5. Sujeción del intestino

Para poder observar la colisión de una forma más adecuada, reajustando los parámetros de ser necesario, cabe mantener el intestino en una posición inicial fija igual que lo estaría en un entorno real debido a las fuerzas ocasionadas por el resto de órganos circundantes. Para ello, no es suficiente mantener fijos únicamente una serie de puntos, como ocurría en la simulación inicial, si no la totalidad del intestino.

Una de las opciones viables sería la utilización del mismo componente, *FixedConstraint*, fijando el total de los vértices de la malla.

Si se buscan otras opciones, la única posibilidad sería utilizar un componente *ForceField* que aplique la fuerza necesaria para mantenerlo fijo. En este caso, la mayoría de clases

disponibles modelan las fuerzas internas, tal como sucede con *TetrahedronFEMForceField*, y no es posible utilizar más de una clase con esa funcionalidad para la misma malla, ya que no es soportado por SOFA. Queda, mediante el descarte de las opciones incompatibles, una clase denominada *RestShapeSpringsForceField* cuyo objetivo es mantener un objeto fijo en una posición de reposo. Mediante sus atributos *stiffness* y *angularStiffness* se definen la resistencia del objeto a ser deformado y su capacidad de volver a su posición inicial, con la diferencia de que el segundo atributo define el valor aplicado a la deformación angular del objeto.

- Descripción: Se repetirán las pruebas para las escalas en milímetros y centímetros manteniendo al intestino en una posición fija mediante las clases mencionadas, estudiando los parámetros de las mismas, si los tuvieran, y observando su efecto en la simulación.
- Resultados: Utilizando la clase *FixedConstraint* la colisión observada muestra un comportamiento del intestino totalmente rígido, independientemente de la escala. Dicha clase no contiene ningún atributo que permita ajustar la rigidez o la fuerza de la sujeción, por lo que resulta inadecuada para el objetivo buscado. Además, si se intenta manipular la fuerza aplicada a la esfera para forzar alguna respuesta diferente en el intestino, esta llega a atravesarlo directamente debido a unos valores que exceden los márgenes adecuados, como se comprueba en la figura 16.



Figura 16: Prueba 5 - Resultado de aplicar fuerzas excediendo el límite de los valores tolerables.

La clase *RestShapeSpringsForceField* sí muestra una deformación mayor ajustando sus parámetros adecuadamente, como se muestran en las figuras 17 y 18 para el

caso de milímetros y centímetros, respectivamente. Si el valor de los parámetros es demasiado bajo el comportamiento será similar al caso de no aplicar el componente, mientras que si es demasiado alto se comportará como un objeto rígido.

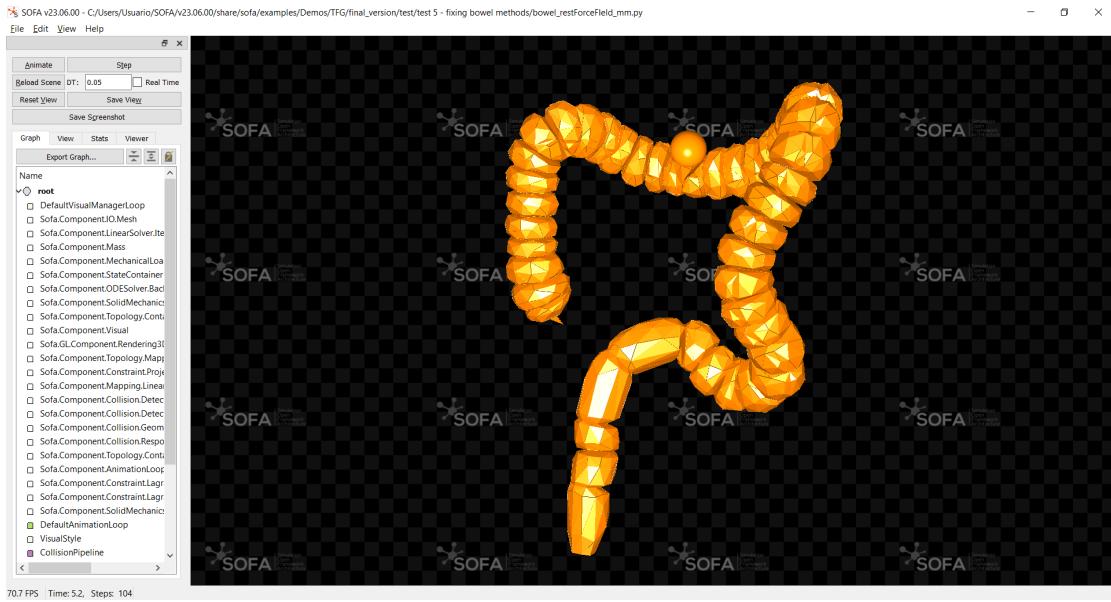


Figura 17: Prueba 5 - Momento posterior a la colisión en la escala a milímetros.



Figura 18: Prueba 5 - Momento posterior a la colisión en la escala a centímetros.

Cabe destacar en las simulaciones que, mientras que a escala a milímetros si se aprecia como la esfera deforma el intestino y se hunde ligeramente en él de manera

suave, en la escala a centímetros se percibe que el resultado de la colisión es más similar a un rebote, pareciendo un material más elástico de lo que en realidad es.

Analizando las gráficas se observa que ambas resultan idénticas entre sí y en comparación con la prueba anterior, con la salvedad de que el componente *RestShapeSpringsForceField* parece tener un coste computacional ligeramente mayor, ya que ahora está entre los diez componentes que más coste exigen a pesar de que no sea suficiente para suponer una diferencia en la simulación. En las figuras 19 y 20 pueden verse las gráficas en el caso de la escala milimétrica.

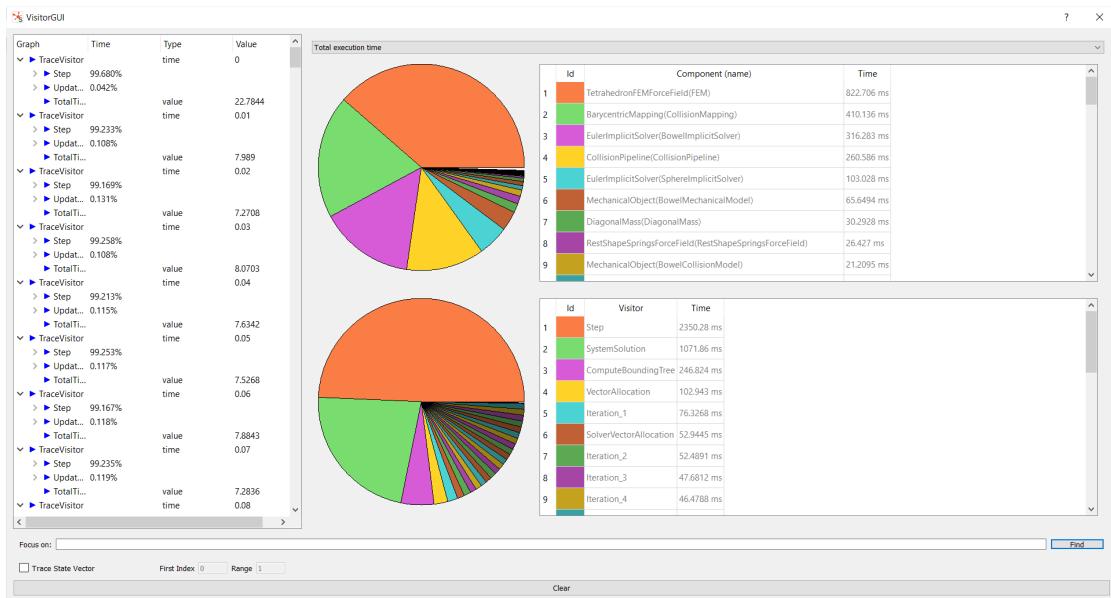


Figura 19: Prueba 5 - Gráfica del tiempo total de simulación en la escala a milímetros.

D.6 Comparación entre una malla cerrada y una hueca

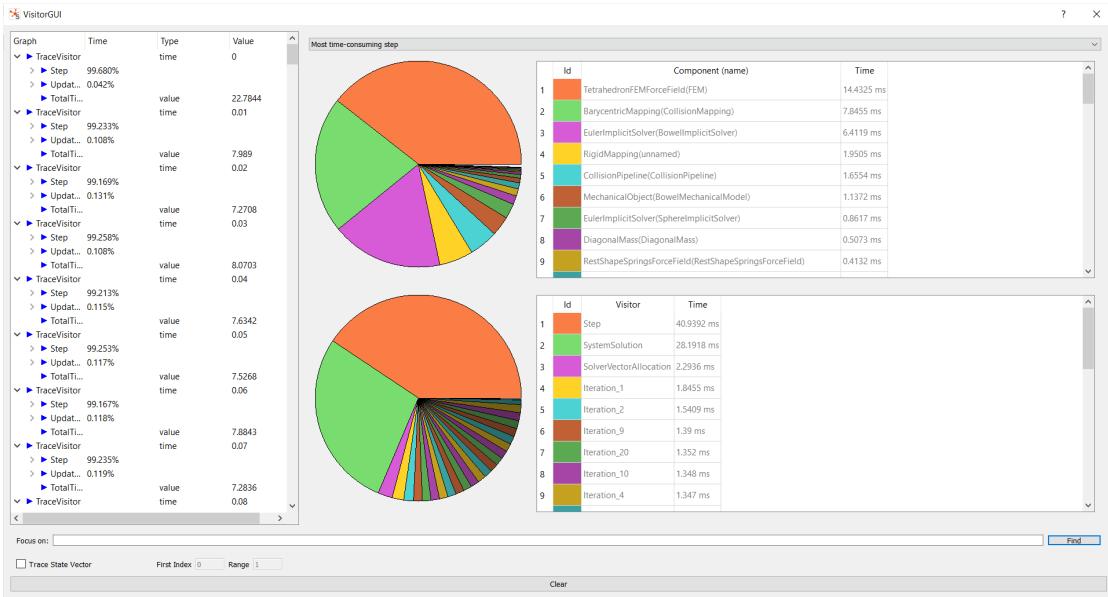


Figura 20: Prueba 5 - Gráfica del paso temporal con mayor coste en la escala a milímetros.

- Conclusiones: Respecto a la primera clase, se puede concluir que retiene los vectores indicados en su posición de manera totalmente rígida, impidiendo su desplazamiento independientemente de la fuerza aplicada. No se ajusta en absoluto al propósito del proyecto.

La segunda opción sí es más adecuada y no sólo permite la sujeción del intestino si no también modificar ligeramente la deformación causada en el intestino logrando una respuesta más realista.

D.6. Comparación entre una malla cerrada y una hueca

Inicialmente, antes de documentar de manera rigurosa las pruebas realizadas, la malla utilizada para el intestino era una malla cerrada, sin abertura en el recto. La utilización de este modelo para el proyecto no hubiese tenido recorrido debido a la imposibilidad de realizar pruebas moviendo un objeto a lo largo del interior del mismo, por lo que su uso fue descartado. Se procedió a modificar dicha malla mediante Blender, creando la abertura con unas dimensiones adecuadas y generando una malla que tuviese un grosor de la pared intestinal adecuado.

No obstante, resulta un objeto de estudio interesante la comparación del comportamiento que tendría dicha malla, que conforma el volumen de un intestino cerrado, en comparación con la malla hueca que conformaría únicamente el volumen de la pared intestinal.

- Descripción: A partir de las modificaciones realizadas en la prueba anterior, se realizará una comparativa entre la malla de un intestino con un volumen cerrado y

otra cuyo volumen se limita al grosor de la pared intestinal, observando el efecto de la diferencia considerable de volumen y el coste exigido. De nuevo, se realizará con ambas escalas.

- Resultados: En el caso de la escala en centímetros la esfera rebota de forma clara en la malla, como si esta fuera exageradamente elástica, a gran velocidad. Si se observan las figuras 21 y 22 con detenimiento, en la esquina inferior izquierda se pueden ver los fotogramas por segundo, el tiempo de simulación transcurrido y el paso temporal. Entre las dos imágenes se aprecia una diferencia de tan sólo 2 pasos temporales, dando una clara impresión de la velocidad a la que sale despedida la esfera tras el impacto.



Figura 21: Prueba 6 - Paso temporal N° 150 de la simulación en la escala a centímetros.

D.6 Comparación entre una malla cerrada y una hueca

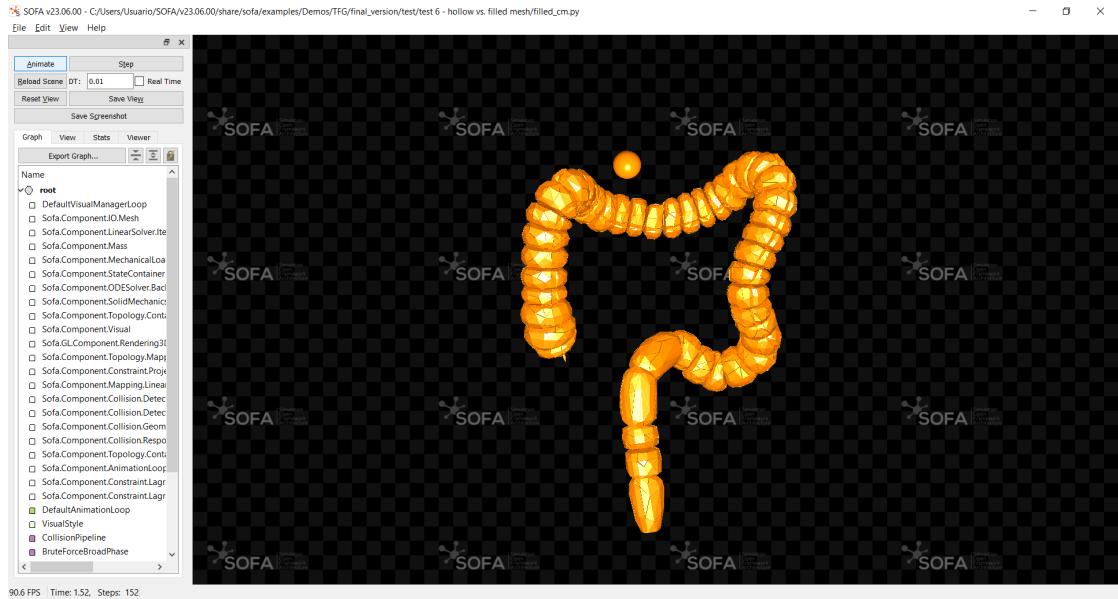


Figura 22: Prueba 6 - Paso temporal N° 152 de la simulación en la escala a centímetros.

En el caso de la escala a milímetros la esfera se hunde en el intestino mucho más levemente que en la prueba anterior, como se percibe en la figura 23.

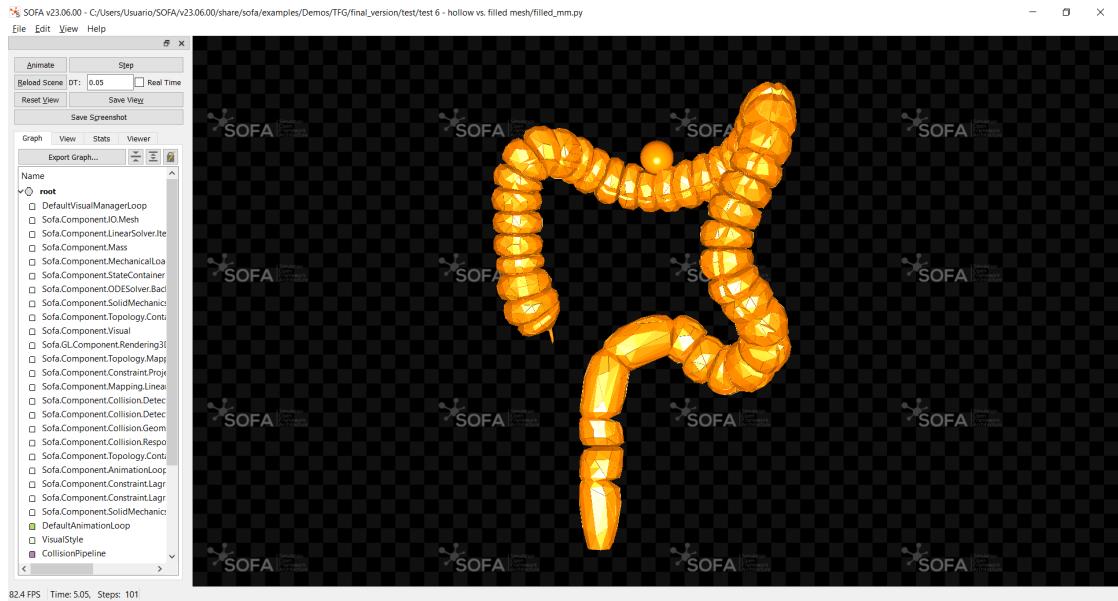


Figura 23: Prueba 6 - Momento de colisión en la escala a milímetros.

Las gráficas no muestran diferencia alguna respecto a la prueba anterior.

- Conclusiones: La simulación de un cuerpo que ocupa mucho más volumen, como se ha podido apreciar, implica un comportamiento muy distinto en el que el propio

cuerpo resiste con más fuerza una deformación. Ha de tenerse en cuenta que la densidad, el módulo de Young y el coeficiente de Poisson no han sido modificados, y el valor que deberían llegar a tener para que el intestino se comportase de manera adecuada son desconocidos al llegar a variar el volumen del cuerpo, por lo que se dificultaría enormemente el estudio de su parametrización adecuada.

D.7. Comparación con una malla triangular

Ya que la malla utilizada es hueca, puede llegar a ser factible que una malla exclusivamente superficial compuesta por triángulos permita una simulación adecuada, obviando el posible efecto que la falta de grosor de la pared del intestino pudiera tener de ser el caso. La manera de lograr esto de una forma adecuada sería sencillo: utilizar un formato de archivo que contenga únicamente información superficial, como stl, y utilizar una topología triangular mediante las clases *TriangleSetTopologyContainer*, *TriangleSetGeometryAlgorithms* y *TriangleFEMForceField*.

Es necesario utilizar una malla en formato stl¹⁷, ya que definir una topología y unas fuerzas elásticas internas basadas en triángulos en una malla con tetraedros supondrá un error importante en la simulación. Esto se debe a que el *framework* es incapaz de reconocer qué triángulos de cuáles tetraedros coinciden con la superficie del objeto, dando lugar a un error de simulación.

- Descripción: Con el objetivo de comprobar que la malla tetraédrica es la más adecuada para representar los modelos físico y de colisiones, se realiza una prueba sustituyendo los componentes asociados a la topología y el comportamiento elástico del intestino para que se adecúen a una malla superficial triangular. Una vez más, se probará en las mismas escalas utilizadas hasta el momento.
- Resultados: La prueba realizada no sólo no muestra resultados satisfactorios en ambas escalas si no que provoca deformaciones inestables e incluso la ruptura de la malla, como se puede observar en las figuras 24 y 25 en la escala a centímetros y a milímetros, respectivamente.

¹⁷El formato de archivo msh se obtiene directamente de un archivo stl, generándola de manera automática mediante herramientas como Gmsh, por lo que ya se dispone de los archivos en el formato necesario.

D.7 Comparación con una malla triangular

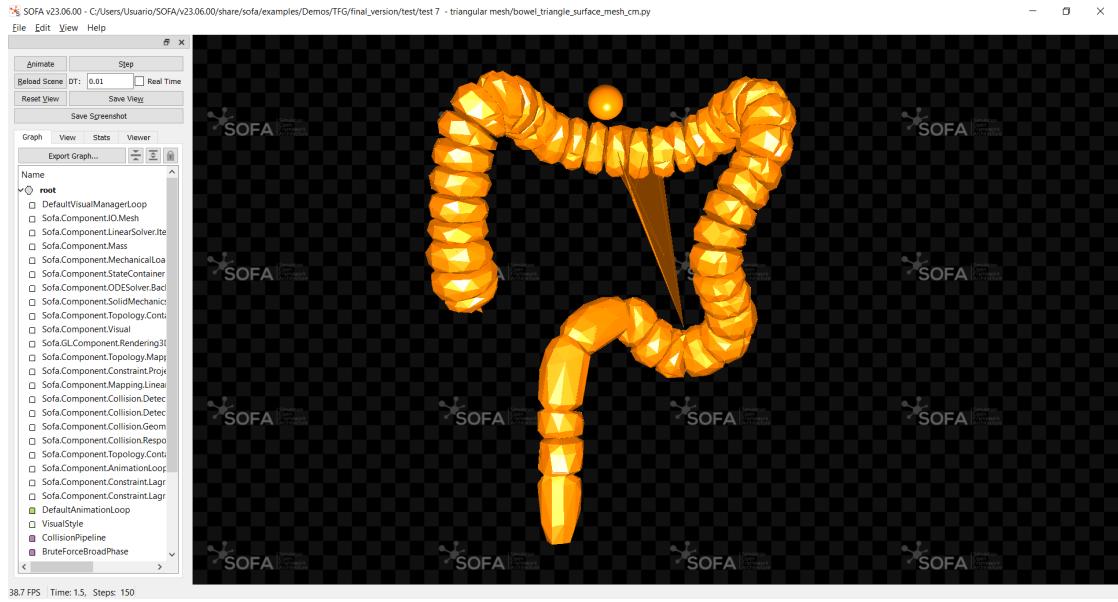


Figura 24: Prueba 7 - Deformación inestable provocada por la colisión en la escala a milímetros.

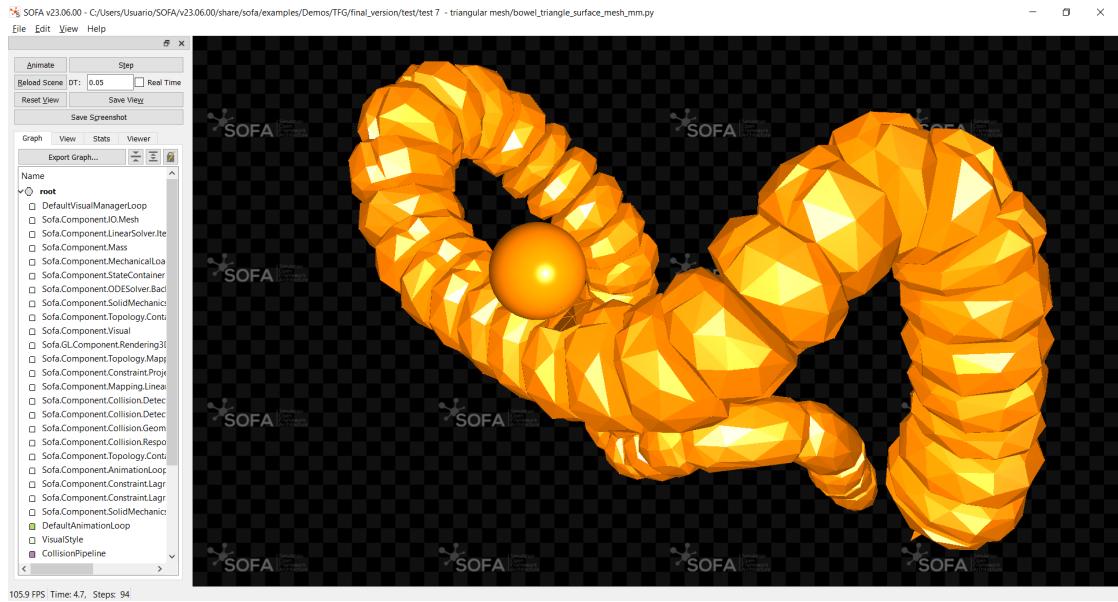


Figura 25: Prueba 7 - Imagen ampliada de la simulación en la escala a milímetros que muestra una ruptura de la malla tras el impacto.

- Conclusiones: El uso de una malla triangular no es eficiente para objetos volumétricos o, al menos, conformados por un volumen que presente una forma compleja como es este caso.

D.8. Esfera volumétrica

El siguiente paso a tratar será definir una esfera como un objeto volumétrico y, por tanto, capaz de comportarse como un objeto deformable en lugar de rígido. Esto aportará una colisión más realista y precisa.

- Descripción: Se modificarán los componentes que definen la esfera para dotarla de un volumen y un comportamiento deformable. Para ello, se utilizarán las mismas clases en el modelo físico de la esfera para definir la masa y elasticidad. Por parte del modelo de colisiones se utilizará el mismo que para el intestino, dejando para más adelante la comparativa entre los componentes *TriangleCollisionModel* y *SphereCollisionModel*.
- Resultados: A la hora de seleccionar unos valores adecuados para la masa y las propiedades elásticas se ha observado que una densidad y/o un módulo de Young demasiado bajos provocarán que la esfera atraviese el intestino sin colisionar, simplemente reduciendo la velocidad de la misma como si atravesase algún tipo de fluido, como se puede apreciar en la imagen 26.



Figura 26: Prueba 8 - Colisión fallida por una densidad y módulo de Young excesivamente bajos en la esfera.

Por otra parte, resulta indispensable experimentar con diferentes valores de *contactStiffness* que permitan una colisión suave y natural, sin un contacto excesivamente rígido.

Tras ir variando el valor de los diferentes parámetros mencionados, se ha logrado una colisión en la que, a diferencia de las anteriores pruebas, la esfera deforma de

manera mucho más clara el intestino en el caso de la escala en milímetros, como se puede apreciar en las figuras 27, 28 y 29.

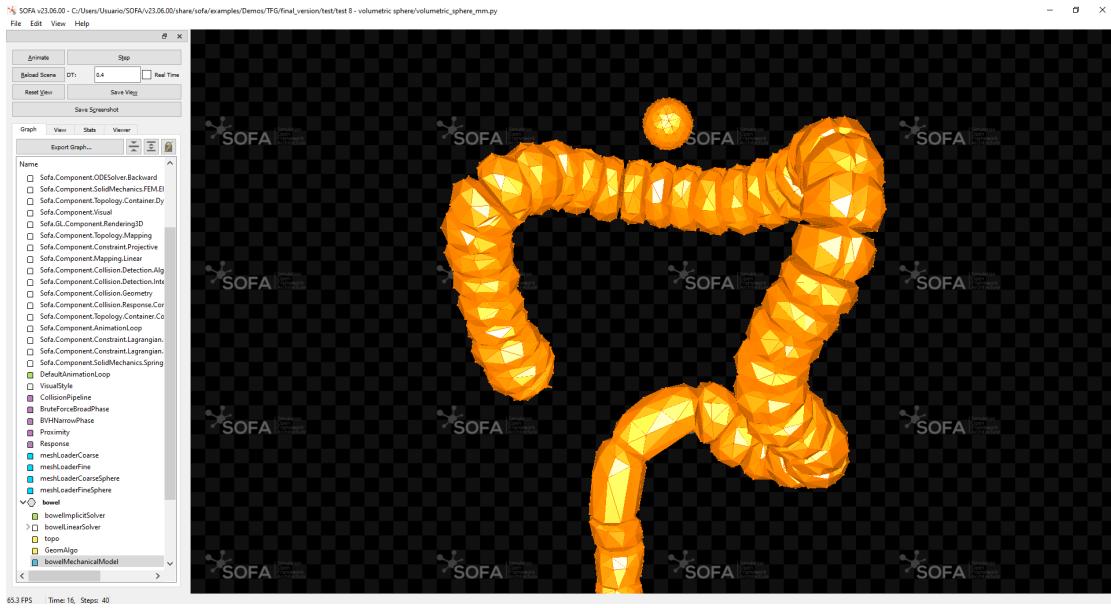


Figura 27: Prueba 8 - Momento previo a la colisión del intestino.



Figura 28: Prueba 8 - Inicio de la colisión, donde se puede apreciar como el intestino comienza a deformarse debido a la fuerza aplicada por la esfera.

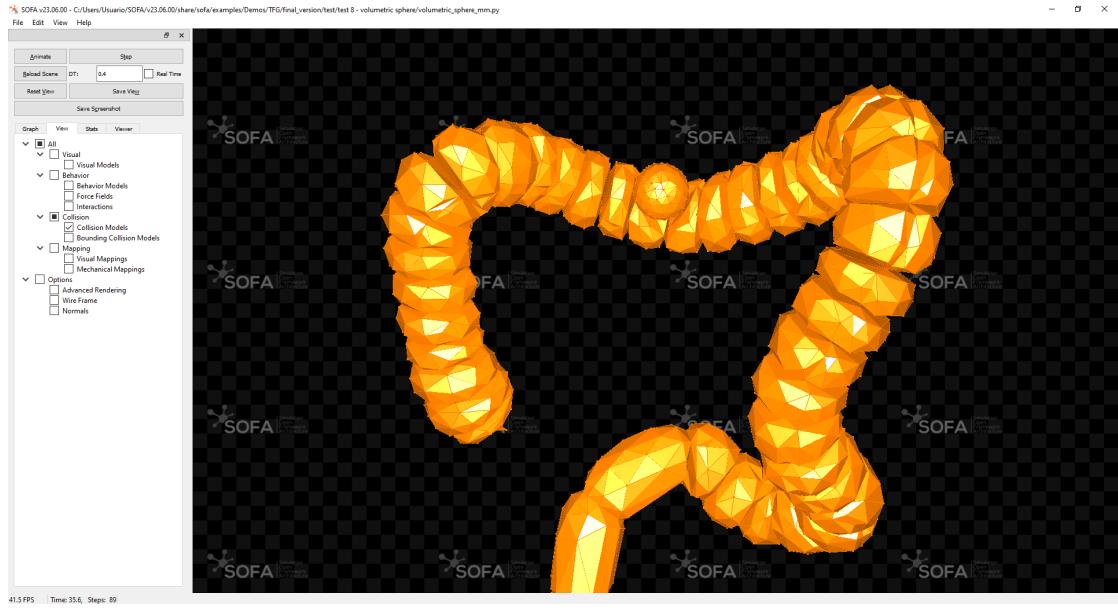


Figura 29: Prueba 8 - Punto de la colisión en la que la esfera se hunde a la profundidad máxima posible en el intestino antes de que éste aplique una fuerza de resistencia intentando recuperar su forma original.

Además, es posible apreciar en la figura 30 cómo la fuerza de la esfera es contrarrestada por las fuerzas internas del intestino, siendo impulsada en el eje horizontal por la misma, ya que la fuerza en el eje vertical ha sido anulada y todavía no se está aplicando una fuerza gravitatoria adecuada.



Figura 30: Prueba 8 - Instante en el que la esfera, al verse sometida a las fuerzas internas del intestino, comienza a desplazarse en sentido horizontal a falta de fuerzas gravitatorias.

En el caso de la escala en centímetros, no se ha podido lograr un ajuste adecuado de los parámetros: para que la esfera colisione adecuadamente requiere un valor de *contactStiffness* que provoca una colisión excesivamente rígida, incapaz de deformar el intestino adecuadamente aunque se reduzcan el valor de los parámetros del componente *RestShapeSpringsForceField* al mínimo.

Analizando el rendimiento se puede apreciar que, tras la colisión, la simulación llega a caer cerca de los 30 fps. Con respecto al coste de computación, se puede percibir de forma clara el impacto de simular dos cuerpos con volumen y fuerzas elásticas, reflejando en la figura 31 el efecto de los componentes añadidos a la simulación.

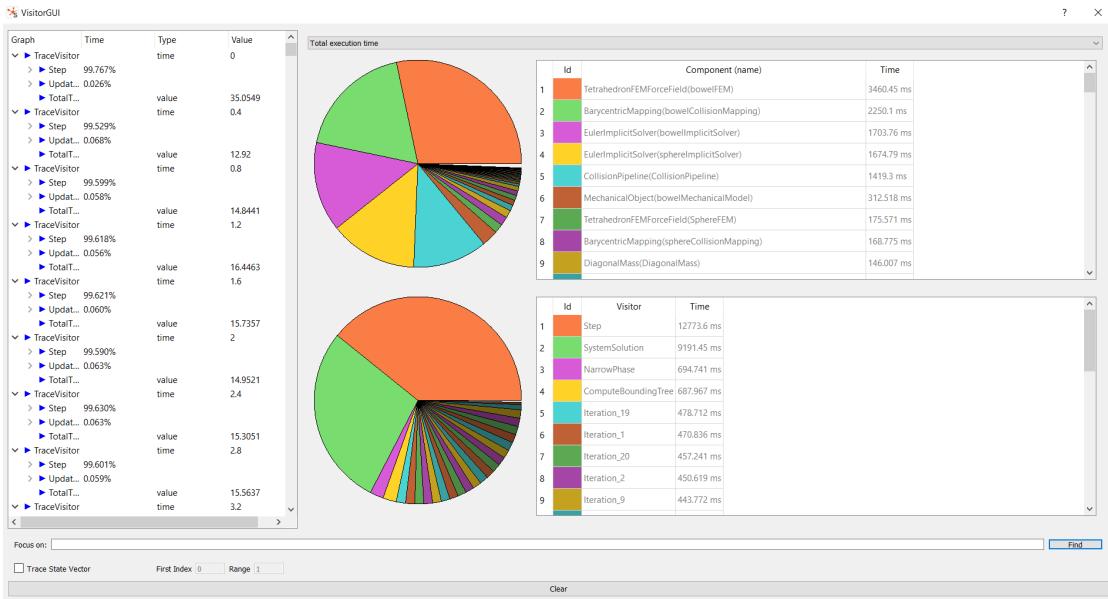


Figura 31: Prueba 8 - Gráfica del tiempo total de simulación en la escala a milímetros en la colisión de dos cuerpos con volumen y fuerzas elásticas modeladas.

- Conclusiones: Queda descartada, tras los resultados observados, la simulación a escala en centímetros, ya que no ha sido posible lograr una parametrización adecuada por los motivos ya mencionados anteriormente referentes a la escala. Resultan ahora más evidentes los motivos que provocaron que la malla utilizada como modelo en la simulación del hígado no representase su escala real.

A la hora de modelar la simulación entre dos cuerpos deformables, resulta de gran importancia asignar un valor adecuado a los parámetros de la masa, la elasticidad y la rigidez de contacto; ya que son responsables en su mayor parte de un comportamiento adecuado frente a una colisión.

Cabe remarcar una vez más la importancia de experimentar con diferentes valores en ensayos de prueba y error, ya que son muchas las variables de las que depende una adecuada simulación y no siempre los valores más apropiados resultan los más intuitivos.

D.9. Incorporación de modelos visuales

Tras la prueba anterior comienza a ser necesario tratar de disminuir el coste computacional antes de que la calidad de la simulación se reduzca en exceso, pero antes debe añadirse un último modelo, el visual, para que su coste se tenga en cuenta a la hora de optimizar la simulación.

- Descripción: Se añadirán los modelos visuales apropiados a la simulación, tanto para el intestino como para la esfera. Para ello se utilizarán los componentes *OglModel*

y *BarycentricMapping*, mencionados en los apartados 3.6 y 4.3, y las mallas en formato obj de los dos objetos. Cabe rememorar que las mallas en formato obj tienen una mayor cantidad de componentes para mejorar la precisión visual de la simulación.

- Resultados: La calidad visual de la simulación aumenta, permitiendo ver de forma más precisa la colisión y la deformación del intestino, como se puede apreciar en las figuras 32 y 33. Además, el impacto en los fps es imperceptible y los componentes no tienen gran impacto en la simulación, como se observa en la gráfica de la figura 34.

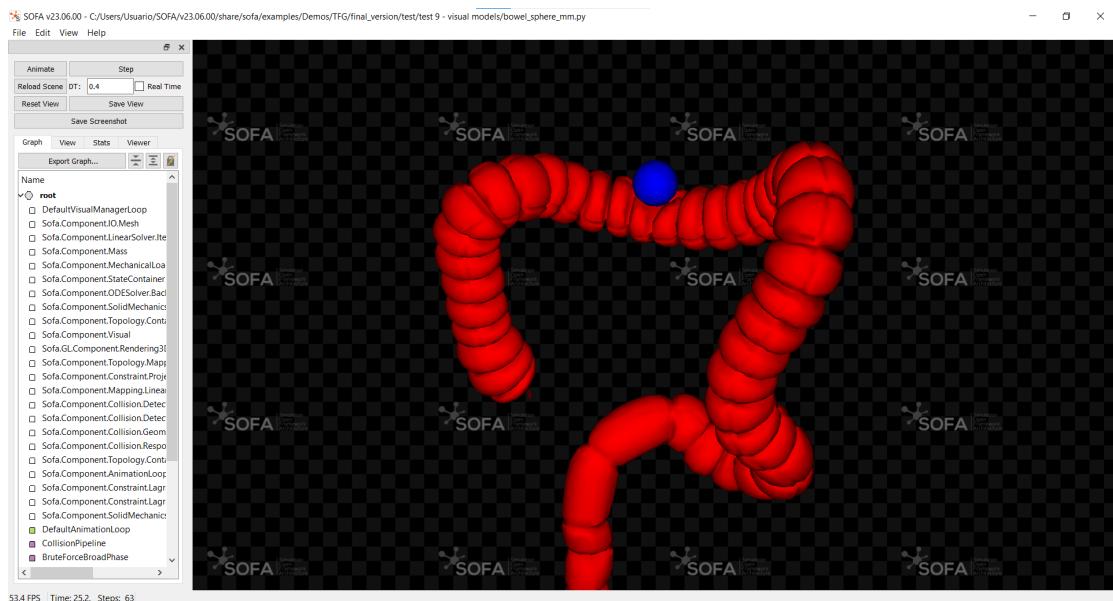


Figura 32: Prueba 9 - Inicio de la colisión añadiendo modelos visuales, cuando el intestino comienza a deformarse.

D.9 Incorporación de modelos visuales

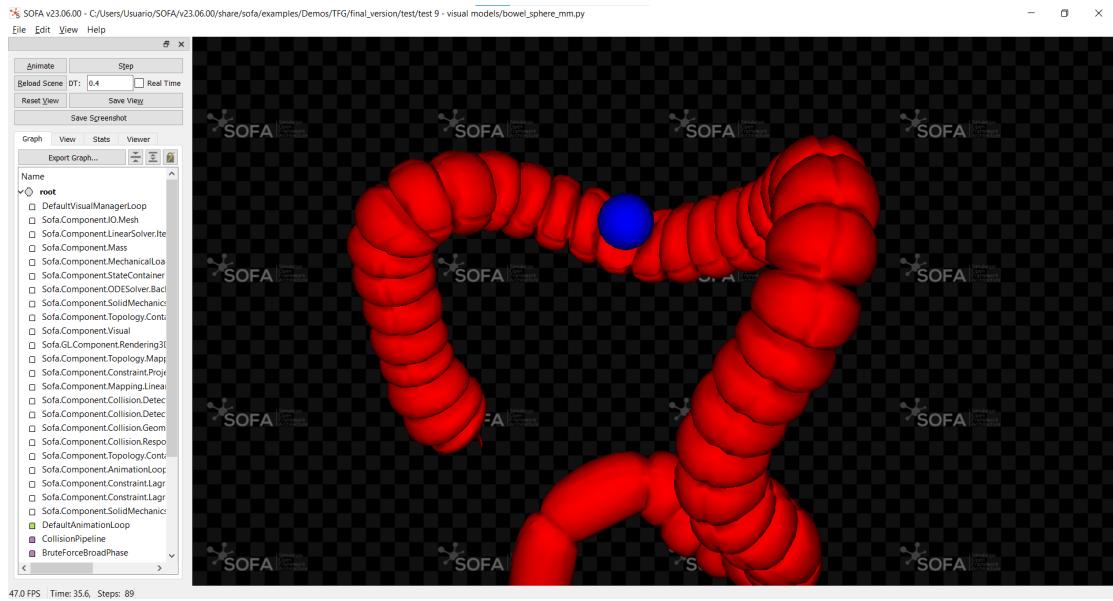


Figura 33: Prueba 9 - Punto de la colisión añadiendo modelos visuales en la que la esfera se hunde a la profundidad máxima posible en el intestino antes de que éste aplique una fuerza de resistencia intentando recuperar su forma original.

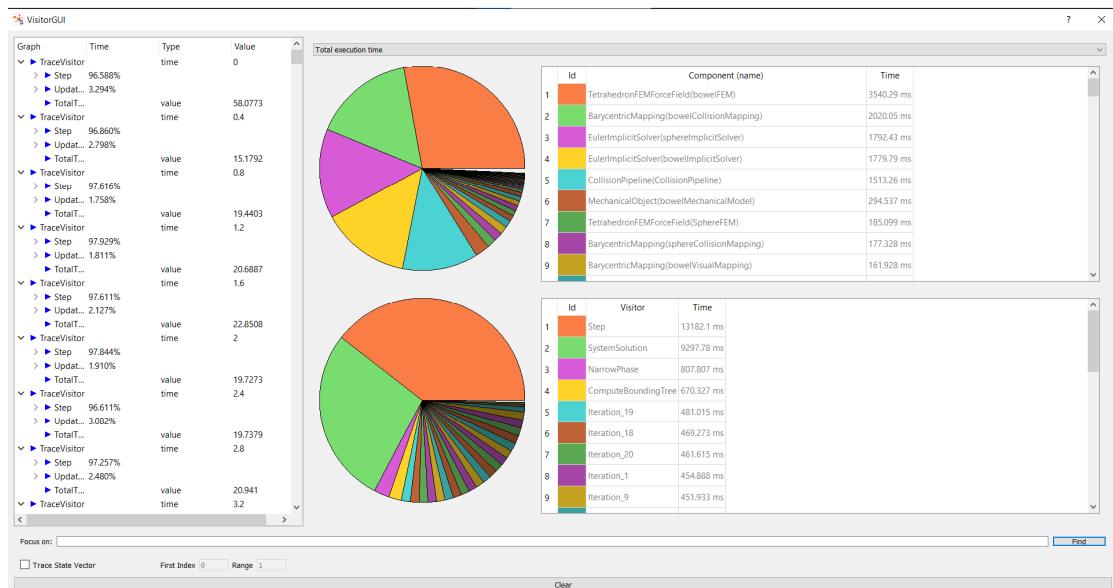


Figura 34: Prueba 9 - Gráfica del tiempo total de simulación al agregar modelos visuales.

- Conclusiones: La incorporación de modelos visuales es, en términos de eficiencia, aparentemente mínimo, al menos en este caso. Además, aporta una calidad visual a la simulación que permite apreciar de manera más natural las colisiones y las deformaciones de los objetos.

D.10. Reducción de elementos de la malla

Como se ha podido apreciar en las anteriores pruebas, la calidad de la simulación ha ido disminuyendo llegando a obtener unos mínimos de entre 30 y 50 fps, dependiendo del instante en la simulación.

A pesar de que se conservan una calidad y estabilidad adecuadas, es deseable mantener los fps a un nivel alto antes de que suponga una inefficiencia notoria. Uno de los puntos más importantes a tales efectos es regular la cantidad de elementos que componen las mallas, debiendo alcanzar un equilibrio entre calidad y eficiencia.

- Descripción: Se modificará la cantidad de componentes de las mallas para realizar la prueba anterior con cada una de ellas y estudiar el impacto de la cantidad de componentes de las mallas en los fps.
- Resultados: En la tabla 2 podemos observar la cantidad de fps que supone el uso de una malla con distinta densidad de elementos en términos de coste computacional. Dicho valor ha sido medido en el momento que mayor coste computacional requiere, durante la colisión, teniendo en cuenta que durante el resto de la simulación la cantidad sigue manteniéndose alta, entre 80 y 120 fps. La cantidad concreta de elementos de cada malla puede ser consultada en el anexo I, siendo la malla denominada "Intestino 2" la utilizada hasta ahora en las pruebas. Además ha de tenerse en cuenta que se están manipulando las mallas del modelo visual y los modelos físico y de colisiones de manera proporcional.

Rendimiento de las mallas	
Malla	Mínima cantidad de fotogramas por segundo alcanzada
Intestino	27 fps
Intestino 2	31 fps
Intestino 3	42 fps
Intestino 4	51 fps
Intestino 5	61 fps

Tabla 2: Fotogramas por segundo en el momento de mayor carga computacional según la cantidad de componentes de las mallas, ordenadas de mayor a menor cantidad.

En el caso de las mallas de los modelos visuales, se han escalado también proporcionalmente y usado con la correspondiente malla del modelo físico y de colisiones. Como ya se ha visto en los resultados de la anterior prueba, no sólo afectan mínimamente a la eficiencia si no que además se ha probado cada una de forma independiente con cada una de las mallas físicas y de colisiones, causando apenas una caída de 10 fps entre la malla con más elementos y la malla con menos.

- Conclusiones: Reducir la cantidad de componentes de la malla, siempre y cuando se mantenga una cantidad adecuada para una correcta deformación, supone una gran

mejora en la calidad de la simulación, logrando duplicar en este caso la cantidad mínima de fps alcanzada en los instantes críticos. A partir de este punto, las siguientes pruebas serán realizadas con la versión más reducida de las mallas, la correspondiente en la tabla al Íntestino 5".

Hasta ahora se ha analizado con detenimiento a lo largo de distintas pruebas el uso de diferentes topologías, la parametrización más adecuada o la incorporación y utilización de los diferentes modelos (físico, de colisiones y visual). Antes de proseguir con las pruebas restantes de cara a la optimización y mejora del modelo, es preciso detenerse y analizar otras clases y parámetros para determinados componentes esenciales que hasta ahora se han supuesto adecuados, como la clase para el componente que define la masa de los objetos o los *solvers*. En concreto, se realizarán varias pruebas con los siguientes componentes:

- Comparativa de diferentes clases del componente *Mass*, que define la masa y/o densidad de un objeto.
- Comparativa de diferentes clases del componente *ForceField*, en concreto las clases que permiten definir las fuerzas elásticas internas de un objeto.
- Estudio de modificar los valores de los parámetros de los *solvers*, encargados de la resolución de las ecuaciones que conforman las variables de la simulación en cada paso temporal.
- Utilización de diferentes primitivas de colisión, que definen qué elementos de la malla interceden en el cálculo de la colisión.

D.11. Pruebas con diferentes clases del componente *Mass*

- Descripción: Se realizarán las pruebas con las dos clases restantes: *UniformMass* y *MeshMatrixMass*, cuyo comportamiento y características se describen brevemente en el anexo F.2.
- Resultados: Visualmente ambas clases mantienen un comportamiento similar con la utilizada hasta ahora, siendo apenas perceptible algún cambio en la deformación del intestino. A nivel de eficiencia, si supone una disminución drástica en los fps el uso del componente *MeshMatrixMass*, llegando a descender a 24 fps. Además, la clase *UniformMass* sólo permite asignar la masa del componente, no su volumen.
- Conclusiones: Por una parte, la clase *MeshMatrixMass* afecta de manera crítica a los fps sin aportar mejoras visuales significativas. Por otro lado, la clase *UniformMass* mantiene un buen rendimiento pero implica una distribución de masa uniforme, lo que puede impedir el comportamiento adecuado frente a determinadas fuerzas que afecten al intestino en cuestión. La clase utilizada hasta ahora, *DiagonalMass*, sigue siendo la más adecuada.

D.12. Pruebas con diferentes clases del componente *ForceField*

- Descripción: Existen diversas clases que pueden modelar las fuerzas internas que representen el comportamiento elástico de un objeto. A pesar de ello, la mayoría no son compatibles con un objeto volumétrico conformado por tetraedros. En el caso que atañe a la simulación realizada, cabe comprobar si la clase *TetrahedralCorotationalFEMForceField* implica una mejora considerable a bajo costo respecto a la clase utilizada. Algunas de las clases más comunes del componente son comentadas en el anexo F.4.
- Resultados: En este caso los fps de la simulación no se ven afectados y, de la misma manera, a nivel visual tampoco surte un cambio notorio.
- Conclusiones: La clase *TetrahedralCorotationalFEMForceField* puede llegar a ser esencial en otro tipo de deformaciones más complejas que sometan a un objeto a una tensión mucho más grave y prolongada, pero en el caso que atañe a este proyecto no parece implicar un cambio de ningún tipo.

D.13. Pruebas con *solvers*

- Descripción: Hasta ahora se han estado utilizando *solvers* independientes para cada uno de los objetos de la simulación, y los valores de sus parámetros se han mantenido constantes. El motivo de utilizar *solvers* independientes, teniendo en cuenta la función y parámetros de los mismos analizados en el apartado 4.3, radica en la posibilidad de dedicar más recursos computacionales a los objetos que más importancia requieran en la simulación. De esta forma, un aumento de los parámetros del *linear solver* en el intestino podrían mejorar la precisión de su comportamiento, mientras que mantener los parámetros del mismo a niveles bajos en la esfera no implicarían un cambio notorio, ya que no sufriría de deformaciones.

En esta prueba se comprobará si dicho razonamiento es verídico con el objetivo de mejorar ligeramente la calidad de la simulación ajustando los parámetros de los *solvers*, sin perder de vista la necesidad de mantener unos fps altos para no perjudicar el nivel de eficiencia de la misma.

- Resultados: Aumentando el valor de los parámetros del *linear solver* del intestino y disminuyendo los de la esfera, puede observarse cómo la colisión afecta de manera levemente más amplia al intestino, afectando a toda la sección transversal del mismo. Por otra parte, la deformación en el punto de colisión es menor. En la figura 35 puede verse el mismo paso temporal que en la figura análoga de la prueba 9: 33.

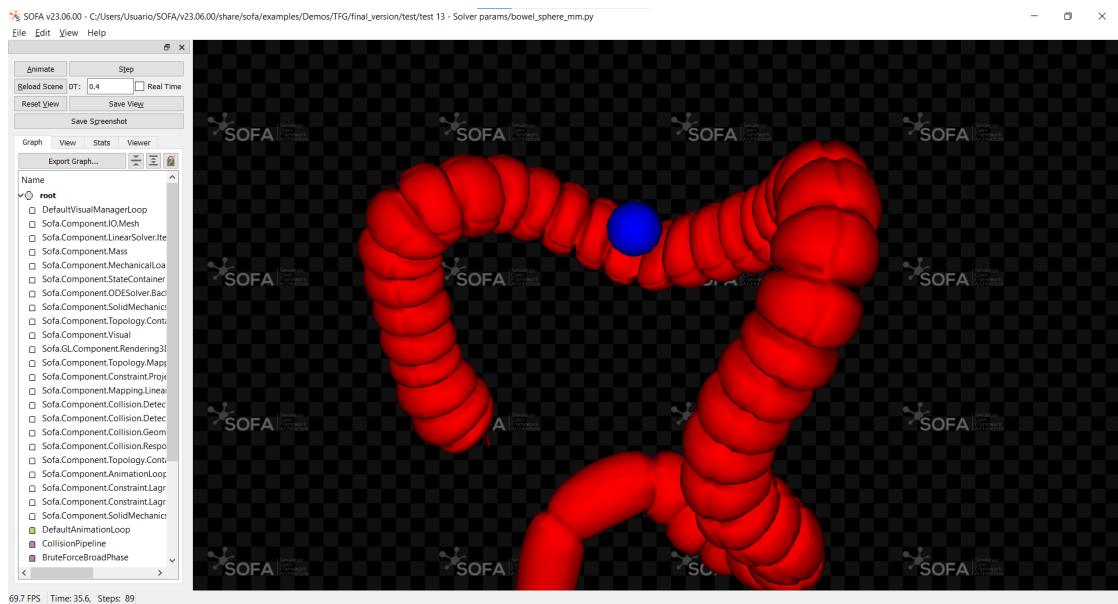


Figura 35: Prueba 13 - Instante de la colisión en el que se puede apreciar cómo la sección transversal del intestino se ve más afectada que en las anteriores pruebas, a costa de un menor efecto en la zona puntual de la misma.

En cuanto al impacto en el rendimiento, podemos observar una caída del mínimo de fps en el momento crítico a casi 45, lo cual implica un descenso moderado. Las gráficas muestran una diferencia significativa, duplicando el tiempo utilizado para el procesamiento del *TetrahedronFEMForceField* del intestino, como se puede visualizar en la figura 36

D.14 Pruebas con diferentes primitivas de colisión

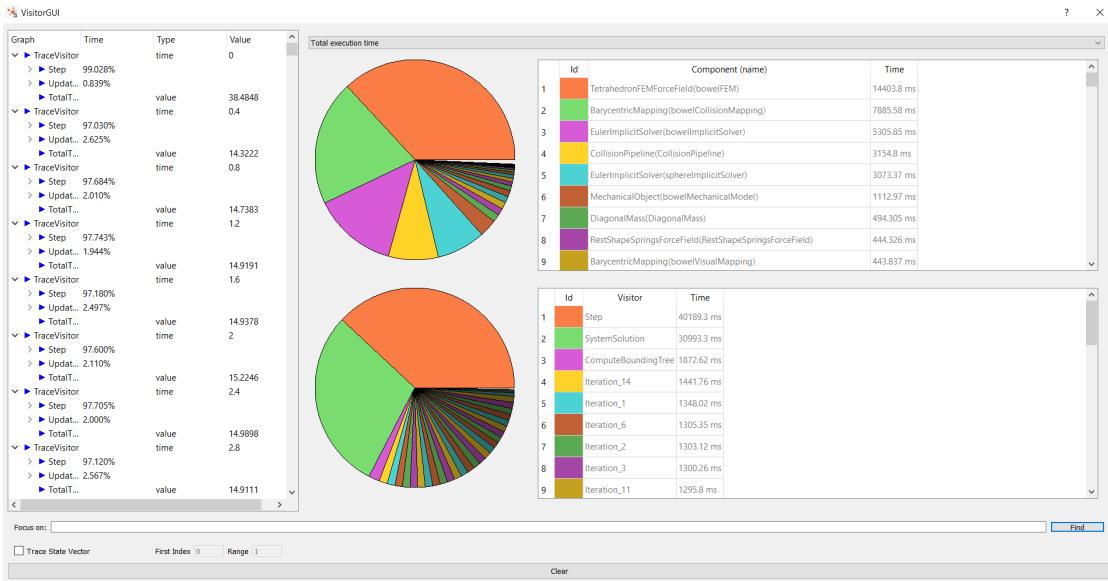


Figura 36: Prueba 13 - Gráfica del tiempo total de simulación en la que se puede apreciar un incremento del tiempo de procesamiento dedicado al componente *TetrahedronFEM-ForceField* del intestino.

El uso de los mismos *solvers* para la esfera y el intestino no resulta en una diferencia notoria, ni a nivel visual ni a nivel de eficiencia.

- Conclusiones: La utilización de diferentes *solvers* y valores para distintos objetos en la simulación es una buena decisión cuando alguno de ellos requiere un interés mayor a costa de los demás, a pesar de que puede afectar significativamente al coste y el comportamiento.

A falta de las últimas modificaciones en la simulación, se mantendrán los valores actuales y se modificarán en última instancia de ser necesarios, ya que hacerlo ahora afectaría de manera significativa al comportamiento.

D.14. Pruebas con diferentes primitivas de colisión

En la prueba 8, donde se parametrizaba la esfera con un volumen, se sustituyó la primitiva de colisión basada en esferas por la misma que la del intestino, basada en triángulos. Cabe recordar, como se mencionó en el apartado 3.5, que las primitivas del modelo de colisiones definen qué componentes de la malla se utilizan para computar una colisión. En el caso del componente *SphereCollisionModel*, que está basado en esferas, su utilización en la ausencia de una esfera volumétrica representaba una única esfera a la cuál se le podía asignar el radio. Al utilizar un cuerpo volumétrico, dicha primitiva procede a llenar todo el volumen con esferas del radio indicado, en lugar de recrear una única, como se puede apreciar en las figuras 37 y 38.

D.14 Pruebas con diferentes primitivas de colisión

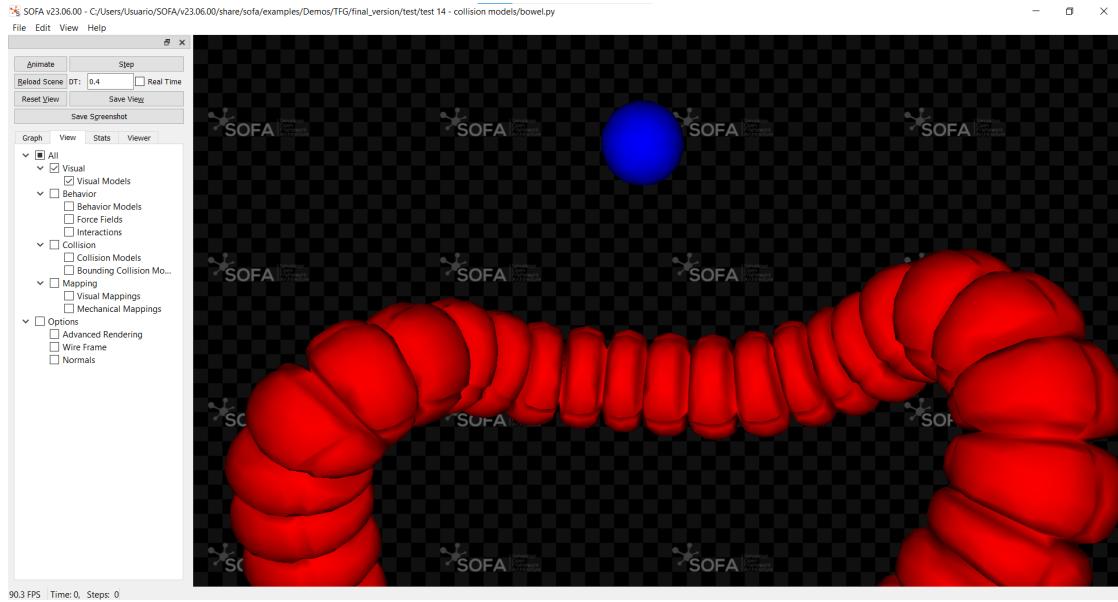


Figura 37: Prueba 14 - Modelo visual de la simulación.

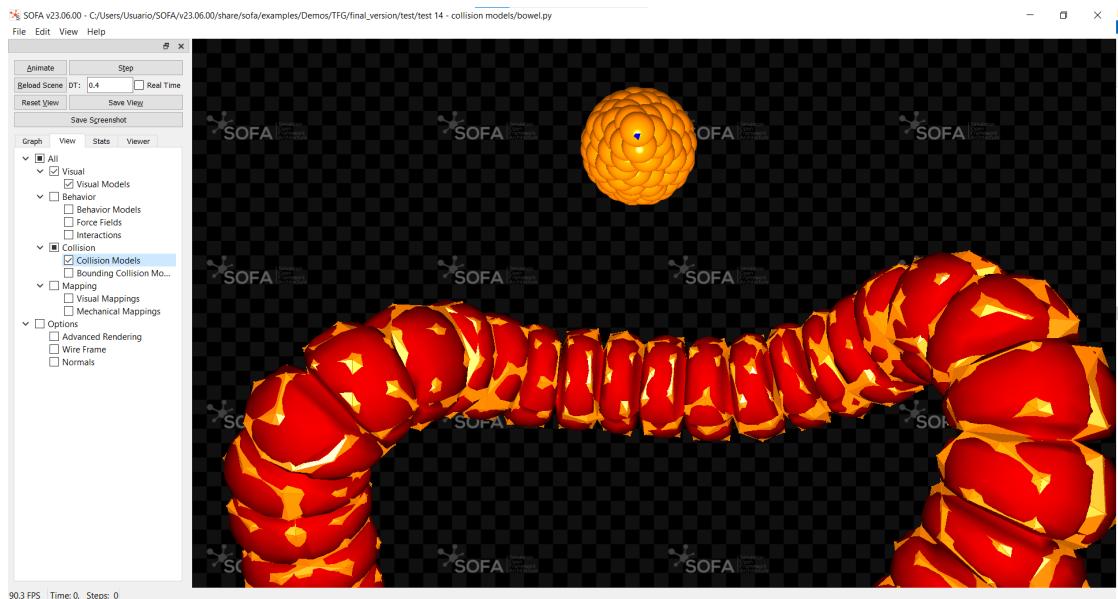


Figura 38: Prueba 14 - Modelo de colisiones superpuesto al modelo visual, donde se puede observar la comparativa de la primitiva de colisiones de la esfera con respecto al visual de la figura anterior.

- Descripción: Se comprobará la viabilidad de la utilización de la primitiva de colisiones *SphereCollisionModel* para la esfera en comparativa con la *TriangleCollisionModel*, utilizada hasta ahora tanto en el intestino como en la esfera volumétrica.
- Resultados: El radio mínimo posible que se puede asignar a las esferas del componente

SphereCollisionModel cuando se trata de un cuerpo volumétrico es 1. Esto implica que la propia distancia de colisión, independientemente del impacto en la eficiencia, se vea aumentada en superposición al tamaño real de la esfera, como se puede comprobar en las figuras 39 y 40.

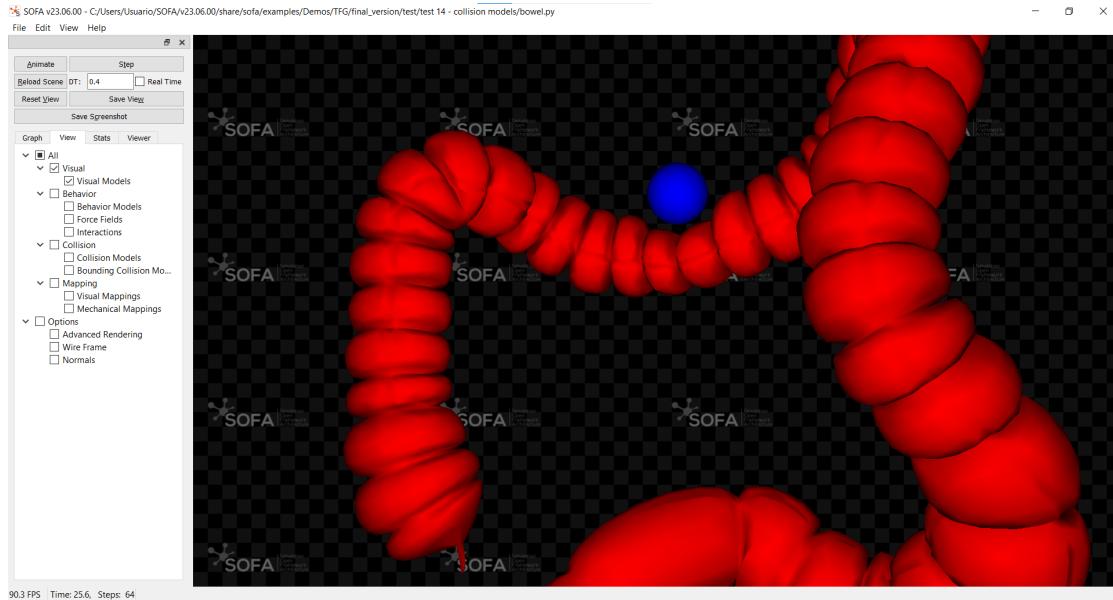


Figura 39: Prueba 14 - Vista del modelo visual en el que se puede observar como el intestino se deforma dejando un espacio considerable entre la esfera y el intestino.



Figura 40: Prueba 14 - Vista del modelo de colisiones superpuesto al modelo visual, donde se puede observar que entre los modelos de colisiones sí hay contacto, a diferencia de lo percibido en la figura anterior.

En contraparte, se puede observar en la figura 41 como el modelo de colisiones utilizando el componente *TriangleCollisionModel* no se ve afectado de la misma manera.

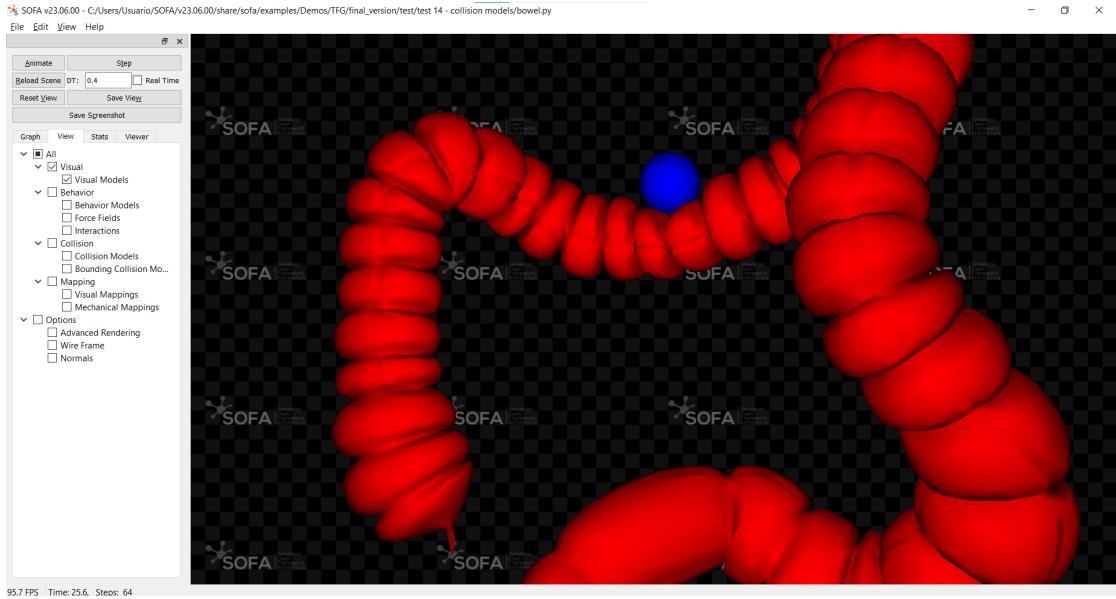


Figura 41: Prueba 14 - Momento de la colisión utilizando el modelo basando en triángulos. El paso temporal de la simulación corresponde a su análogo en las dos anteriores figuras.

La modificación del tamaño de la esfera no resulta viable ya que el modelo de colisiones sigue siendo proporcionalmente más grande, tratando de llenar los pequeños espacios sobrantes del volumen con esferas que sobresalen del mismo.

- Conclusiones: El componente *SphereCollisionModel* es inviable para representar un cuerpo volumétrico esférico en la escala en la que se realiza la simulación.

D.15. Simulación de la gravedad y ajuste de parámetros

Hasta ahora, se ha estado aplicando una fuerza a la esfera mediante el componente *ConstantForceField* que simulase la gravedad del entorno, provocando la colisión de la misma con el intestino. Dado que el órgano se ve afectado de manera natural por la gravedad, carece de sentido no simular dicha fuerza en la totalidad de la simulación.

- Descripción: Se asignará la gravedad a la totalidad de la simulación mediante el componente *gravity* y, de ser necesario, se ajustarán los valores de los parámetros que lo requieran.
- Resultados: Tras la adición de la gravedad, el intestino se ha visto afectado en exceso por la misma, provocando su desplazamiento. Esto ha implicado la modificación de los valores del componente *RestShapeSpringsForceField*, entre otros, obteniendo

tras los cambios una simulación más precisa en la que la sección transversal del intestino se ve afectada en mayor medida tras la colisión, mientras que la deformación provocada por la esfera no disminuye como ocurría en los cambios realizados durante la prueba de *solvers*.

El resultado final es visible en las figuras 42, 43 y 44. También se puede apreciar como ahora la esfera continúa su trayectoria de caída tras la colisión en la figura 45, en lugar de desviarse hacia el eje horizontal.

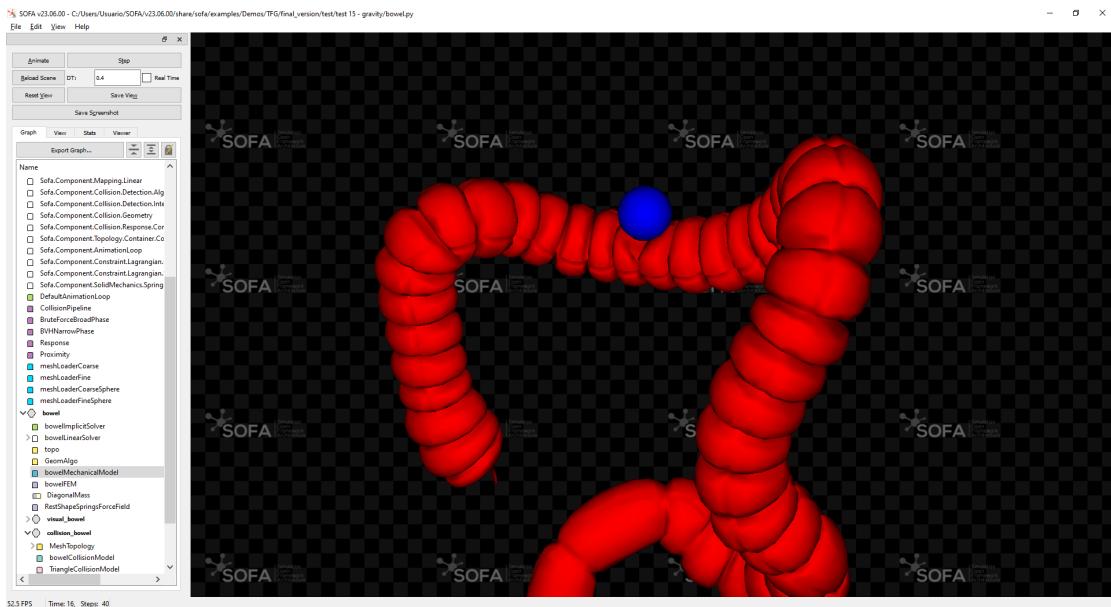


Figura 42: Prueba 15 - Punto de inicio de la colisión tras la adición de la gravedad y los consiguientes ajustes.

D.15 Simulación de la gravedad y ajuste de parámetros

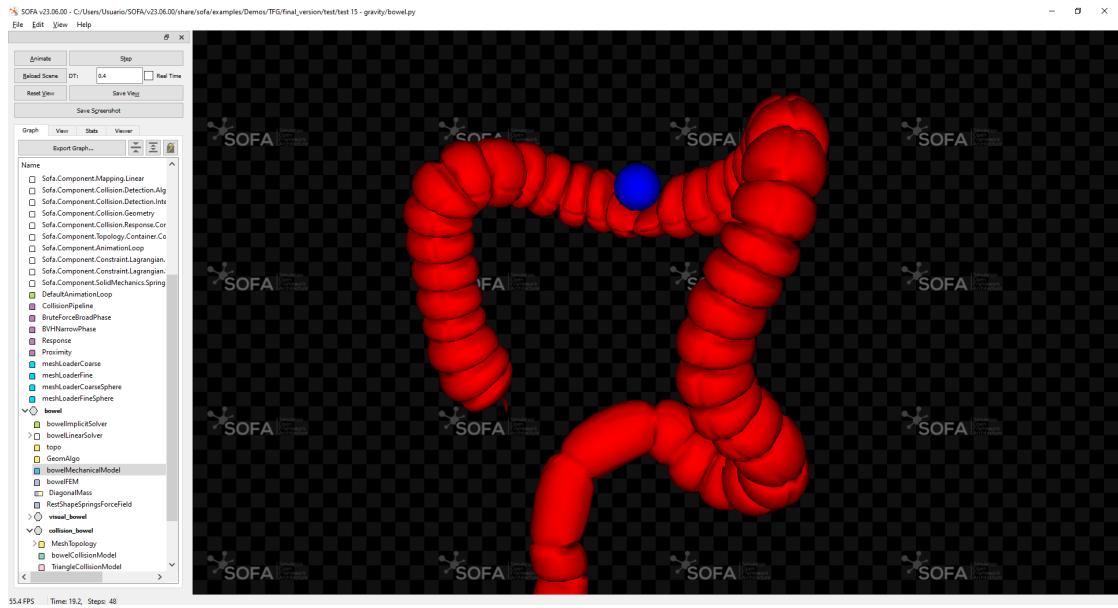


Figura 43: Prueba 15 - Instante de la colisión en la que la esfera deforma a más profundidad el intestino, antes de que las fuerzas internas del intestino la desplacen.

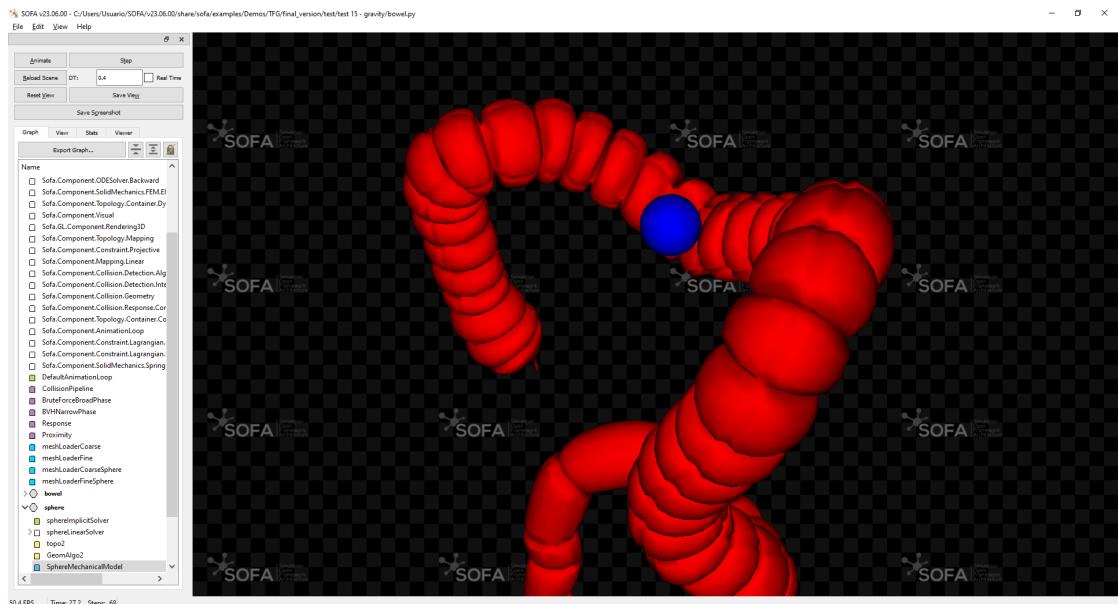


Figura 44: Prueba 15 - Momento de la colisión en la que las fuerzas internas del intestino desplazan la esfera hacia un lateral, viéndose afectada la sección transversal por el efecto de la colisión.

D.15 Simulación de la gravedad y ajuste de parámetros

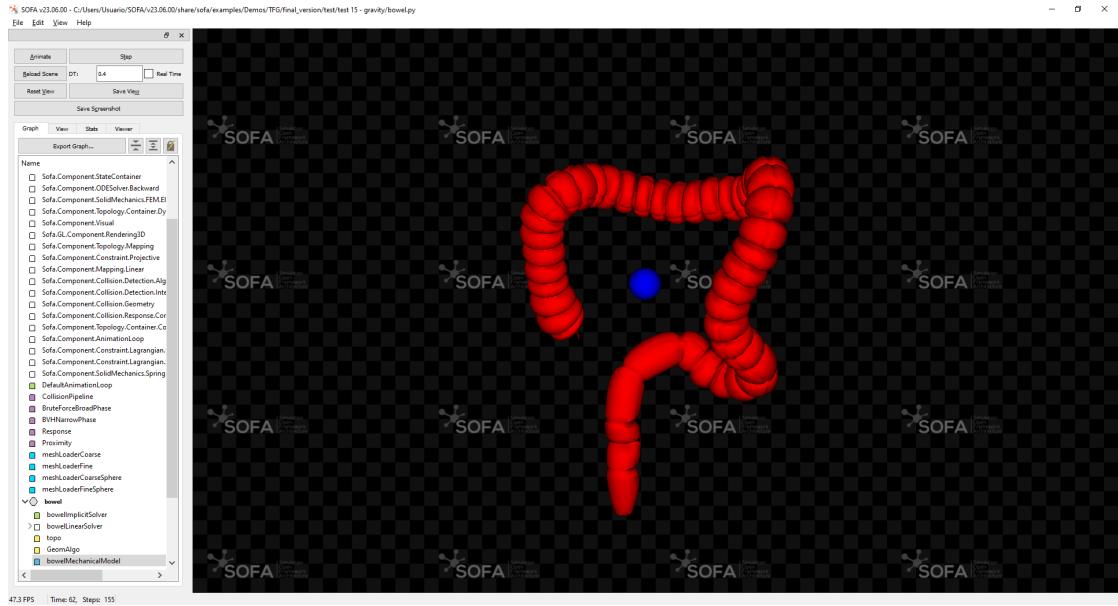


Figura 45: Prueba 15 - La esfera continúa su trayectoria cayendo hacia la parte inferior, en lugar de siendo desplazada en el eje horizontal como en anteriores pruebas.

El coste actual de la simulación se mantiene relativamente estable, sin descender de los 45 fps, mientras que las gráficas (figuras 46 y 47) continúan reflejando que los componentes de los *linear solver* de ambos objetos y aquel que representa las fuerzas internas del intestino siguen siendo los que tienen mayor impacto en la simulación.

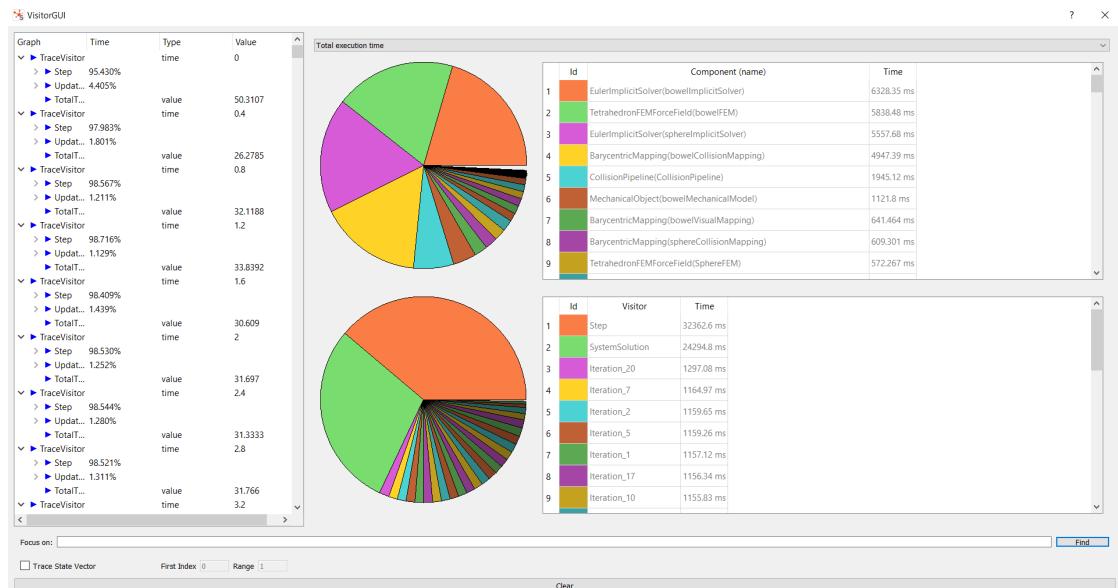


Figura 46: Prueba 15 - Gráfica del tiempo total de simulación.

D.16 Manipulación del movimiento de la esfera y adición de una cámara secundaria

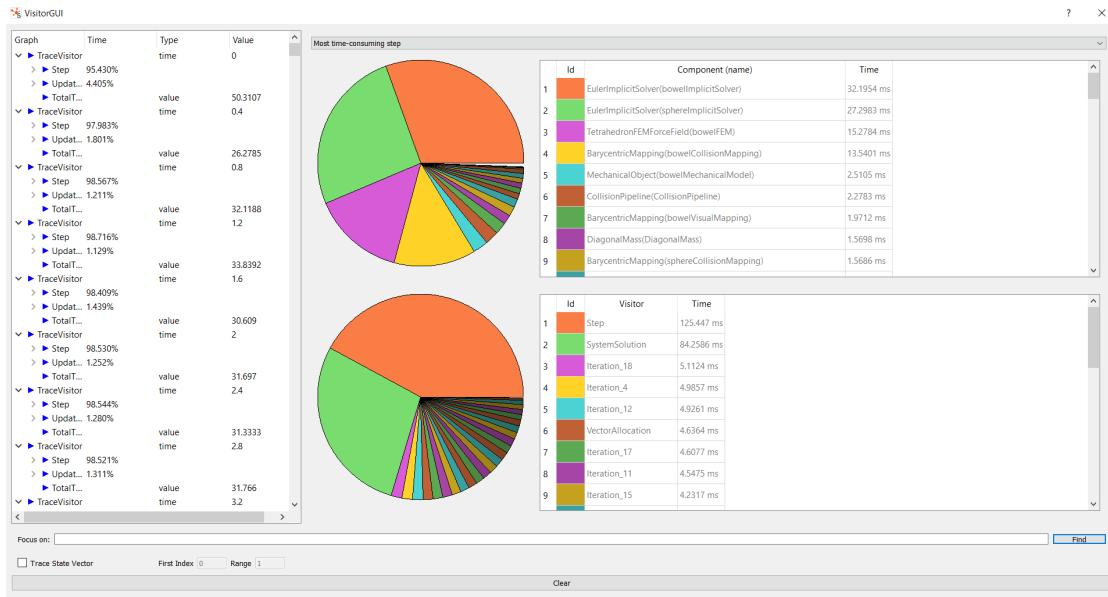


Figura 47: Prueba 15 - Gráfica del paso temporal de mayor coste de la simulación.

- Conclusiones: La adición de la gravedad a la totalidad de la simulación no sólo es necesaria para la imitación de un entorno más realista si no que ha permitido el ajuste de parámetros en aras de obtener un comportamiento más adecuado del intestino, notándose los efectos de la colisión no sólo en el punto de impacto si no en toda la sección afectada.

D.16. Manipulación del movimiento de la esfera y adición de una cámara secundaria

Es posible, gracias a la utilización de *python* para el desarrollo de la simulación, añadir una funcionalidad en el código que permita controlar ciertas variables de la simulación en tiempo real a través del teclado del dispositivo.

Con el propósito permitir al usuario manipular directamente el movimiento de la esfera, se debe implementar dicha funcionalidad de manera que resulte sencilla e intuitiva para el mismo.

A mayores y, ya que la finalidad es poder desplazar el objeto por el interior del propio órgano y visualizarlo, es posible añadir una segunda vista desde la perspectiva de la esfera para alcanzar dicho objetivo.

- Descripción: Se añadirá al código la funcionalidad de poder desplazar libremente la esfera mediante el uso del teclado por parte del usuario. A tales efectos, deberán tenerse en cuenta varios aspectos:

D.16 Manipulación del movimiento de la esfera y adición de una cámara secundaria

1. Ya que se busca la manipulación directa de uno de los objetos de la simulación, debe eliminarse el efecto de gravedad sobre el mismo, de forma que sea posible para el usuario mantener dicho objeto en una posición fija en cualquier instante de la simulación.
2. Se debe tener en cuenta, a raíz del punto anterior, la posibilidad no sólo de movilizar el objeto en ambos sentidos de cada ejes si no también de dejarlo inmóvil en cualquier punto.

Se añadirá también una vista secundaria a la simulación que muestre el punto de vista del objeto secundario (la esfera en este caso).

- Resultados: Se ha logrado, mediante la clase *Controller* del plugin *SofaPython3*, asociar la fuerza aplicada a la esfera en ambos sentidos de cada eje al teclado. Por una parte, se ha asignado el control de las teclas de dirección a los respectivos ejes X e Y, mientras que las teclas K y J corresponden al sentido positivo y negativo del eje Z. Para eliminar las fuerzas que afectan a la esfera en cualquier dirección, basta con pulsar la tecla P.

Para lograr la visualización desde el punto de vista de la esfera, se ha logrado asociar un vértice de la superficie exterior de la misma con una cámara fija cuya perspectiva será visible en la esquina inferior izquierda de la pantalla, como se puede observar en la figura 48. Ya que no existe la posibilidad de cambiar el vértice de la esfera al que se asocia la cámara, se ha añadido la funcionalidad de rotar la visión en cada uno de los ejes mediante las teclas numéricas de la 1 a la 6.

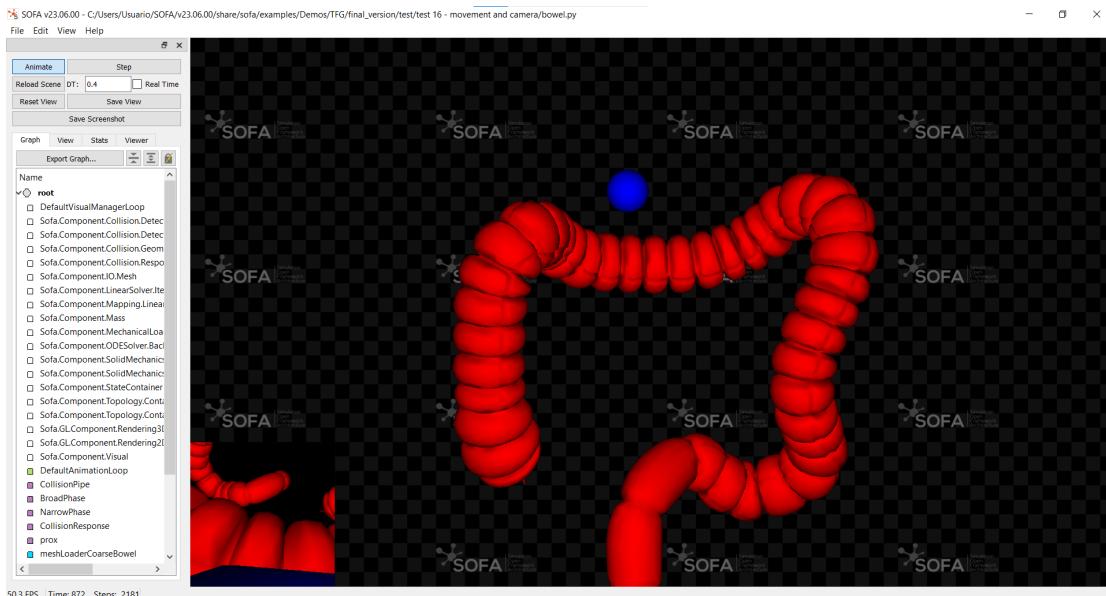


Figura 48: Prueba 16 - Implementación de una vista secundaria asociada a la esfera.

- Conclusiones: Tanto la manipulación del movimiento de la esfera como la vista

D.17 Uso de *BoxRoi* para modelar las secciones del intestino con más precisión

desde una segunda perspectiva y la posibilidad de rotar la cámara de la misma han sido implementadas con éxito.

D.17. Uso de *BoxRoi* para modelar las secciones del intestino con más precisión

En aras de mejorar la precisión del modelo, ha de tenerse en cuenta que las diferentes secciones del órgano poseen propiedades ligeramente distintas [24]. Es posible reflejar dichas disparidades mediante el componente *BoxRoi*.

Dicho componente permite definir una serie de cubos o prismas rectangulares en la simulación, asociando las propiedades que se le indiquen sólo al volumen del objeto contenido dentro del mismo.

- Descripción: Se utilizará el componente *BoxRoi* para definir el módulo de Young y coeficiente de Poisson adecuados a cada sección del intestino.
- Resultados: A pesar de ser necesaria la diferenciación de las diferentes secciones del intestino, el uso del componente *BoxRoi* supone un aumento del coste computacional muy elevado. La simulación se mantiene por debajo de los 30 fps incluso en los instantes donde no se produce ninguna colisión, como se puede apreciar en la figura 49, donde también se ha añadido la visualización de los límites del *BoxRoi* definido para cada sección.



Figura 49: Prueba 17 - Implementación de un *BoxRoi* para cada sección del intestino con propiedades diferenciadas.

Si se analizan las gráficas, también es posible observar en la figura 50 que el coste del

D.18 Pruebas de optimización: uso del plugin Multithreading

componente *TetraHedronFEMForceField* se multiplica por la cantidad de secciones diferenciadas.

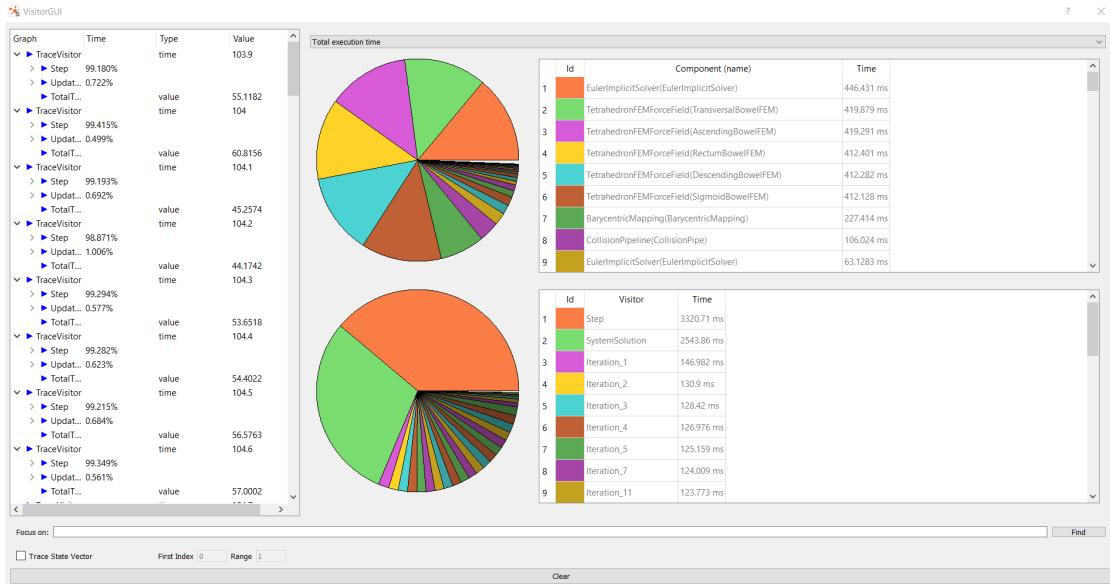


Figura 50: Prueba 17 - Gráfica del tiempo total de la simulación con la implementación de un *BoxRoi* para cada sección del intestino con propiedades diferenciadas.

- Conclusiones: La implementación del *BoxRoi* no es viable si no se optimiza adecuadamente la simulación. Avanzado el proyecto hasta su estado actual, queda someter la simulación a una última optimización mediante la utilización de *plugins* enfocados a tal aspecto. De ser su implementación no viable o insuficiente, solo cabe descartar la adición del componente *BoxRoi*.

D.18. Pruebas de optimización: uso del plugin Multithreading

El *plugin* Multithreading permite explotar la cantidad de núcleos disponibles en la CPU para realizar las tareas más costosas de manera simultánea, mejorando la eficiencia de la simulación.

- Descripción: Se sustituirán los componentes de la simulación por sus análogos del *plugin* multithreading. En el caso de la simulación realizada, están disponibles los componentes *ParallelBruteForceBroadPhase* y *ParallelBVHNarrowPhase* de la *CollisionPipeline*, y el componente *ParallelTetrahedronFEMForceField*.
- Resultados: El uso del componente *BoxRoi* sigue implicando un impacto excesivo en la simulación, a pesar de que el *plugin* utilizado consigue mantener los fps ligeramente por encima de los 30. Si se analiza el impacto del uso del *plugin* en ausencia de los *BoxRoi*, el mínimo se eleva de los 50 fps a los 60.

- Conclusiones: El uso del *plugin* supone una mejora, aunque insuficiente si se desea mantener unos fps altos.

D.19. Pruebas de optimización: uso del *plugin* CUDA

El *plugin* CUDA permite al equipo ejecutar la simulación mediante la GPU en lugar de la CPU, aumentando la eficiencia en consonancia.

- Descripción: Se sustituirán, igual que en el caso anterior, los componentes disponibles por sus análogos, sin necesidad de añadir nuevos valores o modificar los existentes.
- Resultados: El *plugin* no parece funcionar de manera correcta en la simulación, provocando que la caída de fps sea aun peor debido al mal funcionamiento o implementación del *plugin*. En aras de localizar el error, se han realizado las siguientes pruebas:
 - Se han localizado diversos ejemplos que utilicen CUDA y se ha ejecutado su versión con y sin el uso de dicho *plugin*. El aumento de los fps en las simulaciones es visible y considerable, por lo que el *plugin* en sí está correctamente instalado y configurado en el equipo.
 - Se ha partido desde el inicio añadiendo las clases del *plugin* de una en una y realizando la simulación en cada sustitución para focalizar la zona de error. El problema que sucede en este punto se debe a que hay diversos componentes (*MechanicalObject*, *TetrahedronFemForceField*, *TetrahedronSetTopologyContainer*, etc.) que requieren del cambio simultáneo a la utilización de CUDA, provocando un error que impide la simulación en caso contrario, con su correspondiente aviso de dicha incompatibilidad.
 - Se han intentado sustituir sólo los mismos componentes utilizados en ejemplos con CUDA que sí funcionan, siendo este cambio insuficiente o ineffectivo debido a que los componentes con más coste en la simulación son exclusivamente los *TetrahedronFEMForceField* de los *BoxRoi*.
 - La ejecución del *plugin* eliminando los *BoxRoi* tampoco parece ser el motivo, descartando la posible incompatibilidad entre componentes.
- Conclusiones: No ha sido posible ejecutar la simulación con CUDA de manera adecuada, de la misma forma que tampoco se ha logrado encontrar el error que lo causa. A efectos de dejar constancia de ello, el equipo utilizado para el desarrollo del proyecto es un portátil que dispone de una GPU NVIDIA GeForce RTX 3050.

Finalmente se ha optado por tratar de implementar el componente *BoxRoi* con algunas limitaciones, reduciendo a un sólo *BoxRoi* aquellas secciones con mayor similitud entre ellas. Esto permite una simulación con una mayor cantidad de fps permitiendo a su vez una mínima diferenciación entre las zonas del intestino más dispares.

E. Formatos de archivo de las mallas

Los principales formatos de archivo que contienen mallas son:

- stl: Contiene únicamente información geométrica de la superficie del objeto. Las aristas sólo pueden unir vértices conformando caras triangulares. Eficiente para representar superficies simples.
- obj: Contiene únicamente información geométrica de la superficie del objeto. Las aristas pueden unir vértices conformando caras triangulares u otros polígonos. También puede contener información de texturas y color.
- off: Puede contener información geométrica del volumen y la superficie del objeto. Las aristas sólo pueden unir vértices conformando caras triangulares y un volumen compuesto por tetraedros. Eficiente para representar superficies y volúmenes simples.
- msh: Puede contener información geométrica del volumen y la superficie del objeto. Es el formato más eficiente para volúmenes con una cantidad elevada de componentes.
- vkt: Puede contener información geométrica del volumen y la superficie del objeto. También puede contener información de datos escalares, vectores y tensores. Es el formato más versátil.

Cada formato de archivo usa una clase de *MeshLoader* diferente (e.g., para los archivos obj se usaría *MeshOBJLoader*).

F. Colección de clases disponibles

A continuación se mencionan algunas de las clases más utilizadas de diversos componentes junto con su descripción y casos de uso. Cabe mencionar que el continuo avance y desarrollo del *framework* causa que la colección de clases disponibles para cada componente varíe de manera constante¹⁸.

F.1. *AnimationLoop*

- *DefaultAnimationLoop*: Es la clase estándar del componente que permite simular de manera estable y adecuada uno o varios objetos con colisiones entre ellos.
- *MultiTagAnimationLoop*: Permite etiquetar diferentes fases de la simulación permitiendo que se ejecuten en un orden determinado. Es útil para realizar simulaciones que requieran diferentes pasos temporales para distintas fases o evitar que determinadas acciones se simulen de manera simultánea.

¹⁸Se puede consultar la versión completa actual de todas ellas en el siguiente enlace: SOFA API Documentation.

- *MultiStepAnimationLoop*: Utilizada si se requiere la realización de diferentes acciones de manera paralela en un único paso temporal permitiendo, de manera opuesta a la clase anterior, definir pequeñas subfases dentro de un mismo paso temporal. Es apropiado para simulaciones que impliquen computaciones a alta velocidad como la simulación de fluidos, fenómenos electromagnéticos o vibraciones de alta frecuencia.
- *FreeMotionAnimationLoop*: Permite computar de manera separada la resolución del sistema y el cálculo de fuerzas y velocidades del mismo, aumentando la estabilidad y fluidez de la simulación de objetos con movimientos complejos como la ropa o las articulaciones en robots blandos.

F.2. *Mass*

- *UniformMass*: Es una representación simplificada que no tiene en cuenta el volumen de la geometría ni la densidad, y la masa total se distribuye entre todos los vértices por igual. Es usado para representar cuerpos no deformables o con una densidad muy homogénea. Permite asignar el valor de la masa que se distribuirá entre sus vértices. Coste computacional bajo.
- *DiagonalMass*: Usa la información geométrica para una distribución de la masa no uniforme teniendo en cuenta la densidad. Tiene limitaciones a la hora de computar deformaciones que impliquen la torsión y rotación de un cuerpo sobre sí mismo. Permite asignar el valor de la densidad o la masa del objeto. Coste computacional medio.
- *MeshMatrixMass*: Suple las limitaciones de la clase *DiagonalMass* a la hora de simular torsiones y rotaciones de un objeto sobre sí mismo. Permite asignar el valor de la densidad o la masa del objeto. Coste computacional elevado.

F.3. *CollisionPipeline*

Para el funcionamiento del modelo de colisiones, deberá declararse el componente *CollisionPipeline* con la única clase disponible a tal efecto, *DefaultCollisionPipeline*, y podrán usarse para cada uno de sus componentes las diferentes clases disponibles, entre las que destacan:

1. Para el componente que representa la fase *Broad detection phase*:

- *BruteForceBroadPhase*: implica una aproximación directa en la que se comprueba si existen potenciales colisiones entre cada par de objetos. Es el más simple y eficiente en el caso de simulaciones con una cantidad pequeña de objetos.

- *ParallelBruteForceBroadPhase*: es una versión optimizada de *BruteForceBroadPhase* que permite el procesamiento multihilo en el caso de utilizar el *plugin* correspondiente.
 - *BruteForceDetection*: una versión de *BruteForceBroadPhase* más imprecisa y simple, que usualmente es utilizada en fases de pruebas para las primeras versiones de una simulación donde se busca la mayor simplicidad posible.
 - *THMPGSpatialHashing*: integrado en el *plugin* con su mismo nombre, es utilizado para la simulación de objetos con formas regulares y simétricas permitiendo alcanzar una mayor eficiencia y simplicidad mediante la división de los mismos en componentes idénticos.
 - *BulletCollisionDetection*: integrado en el *plugin* con su mismo nombre, permite implementar un modelo de colisiones altamente eficiente específico para cuerpos rígidos y enfocado a la simulación de entornos virtuales interactivos en tiempo real o robótica, entre otros.
2. Para el componente que representa la fase *Narrow detection phase*:
 - *BVHNarrowPhase*: descompone el volumen de un objeto en componentes idénticos que conforman el mismo, como triángulos, permitiendo la detección de colisiones directamente de las formas geométricas simplificadas para mejorar la precisión y calidad en la detección. Es el apropiado para mallas con formas más complejas.
 - *ParallelBVHNarrowPhase*: es una versión optimizada de *BVHNarrowPhase* que permite el procesamiento multihilo en el caso de utilizar el *plugin* correspondiente.
 - *DirectSAPNarrowPhase*: utiliza la información de las posiciones que ocupan las superficies de los objetos en los ejes para detectar colisiones. Es un algoritmo más costoso pero muy eficiente en entornos donde los objetos de la simulación están en constante movimiento.
 3. Para el componente que define el método de intersección, es decir, el comportamiento que permite evitar que los objetos que participan en una colisión lleguen a atravesarse.
 - *MinProximityIntersection*: realiza un cálculo simplificado para detectar la colisión entre los diferentes objetos simulados. Está enfocado a un cálculo eficiente con cierto margen de error permitiendo priorizar una mayor eficiencia frente a la precisión.

- *LocalMinDistance*: calcula la distancia entre todos los puntos de los objetos para detectar la colisión. Es mucho más precisa y, por tanto, exige mayor coste computacional y no es recomendado su uso para objetos con formas complejas.
4. El componente *ContactManager* dispondrá de la clase *DefaultContactManager* y el uso del valor *PenaltyContactForceField* para su atributo *response* es el comúnmente usado en las versiones estables de SOFA.

F.4. ForceField

- *ConstantForceField*: Permite aplicar una fuerza constante a todos los vértices que conforman el objeto. Puede aplicarse a la totalidad de ellos o a un conjunto determinado. La fuerza aplicada se distribuye entre todos los vértices del objeto sobre el que se aplica.
- *PolynomialSpringsForceField*: Permite simular modelos elásticos no lineales mediante el uso de funciones polinomiales. Útil para la simulación de objetos en los que la fuerza que aplican en oposición a la fuerza de deformación es mayor y no proporcional, obteniendo un comportamiento propio de gomas elásticas o determinados tejidos biológicos.
- *TetrahedronFEMForceField*: Permite simular la deformación y el comportamiento físico de topologías tetraédricas usando el método de elementos finitos. Pueden definirse determinados atributos que modelan las propiedades del material y su resistencia a la deformación como el coeficiente de Poisson o el módulo de Young.
- *TetrahedralCorotationalFEMForceField*: Es una extensión de la clase *TetrahedronFEMForceField* que tiene en cuenta los efectos de torsión y rotación del objeto, permitiendo mantener la estabilidad de la simulación ante deformaciones de larga duración. Su coste computacional es más elevado y posee los mismos atributos que la anterior clase.
- *TetrahedronHyperelasticityFEMForceField*: Mucho más precisa que la clase *TetrahedronFEMForceField*. Permite deformaciones mayores e integra comportamientos físicos no lineales. Puede usar una cantidad extensa de algoritmos que definen su comportamiento, cada uno especializado en un tipo de material distinto como goma, plásticos, piel humana, etc. Su caracterización es bastante compleja y el coste computacional, elevado.
- *QuadBendingFEMForceField*: Es utilizado para modelar el comportamiento de flexión en objetos de topologías compuestas por cuadriláteros. A diferencia del método *TetrahedronFEMForceField*, se enfoca en las fuerzas de flexión resultantes de las cargas o deformaciones aplicadas al objeto.

G. Funcionamiento del modelo de colisiones

La detección de colisiones entre objetos está dividida en diferentes fases, cada una implementada en un componente distinto que forma parte del *collision pipeline*:

1. *Collision reset phase*: elimina toda la información procesada por el *collision pipeline* en el anterior paso temporal.
2. *Collision detection phase*: determina si dos o más objetos de la simulación están en colisión. Recibe como información de entrada todos los datos geométricos de los modelos de colisión de cada par de objetos que pueden colisionar.
 - a) *Broad detection phase*: Descarta de la información de entrada los pares que no están en colisión y detecta aquellos que se encuentran en situación de inminente colisión. Para ello, computa una caja alrededor de cada modelo y toma como una posible colisión aquellos cuyas cajas estén en contacto. Devuelve los pares de posible contacto.
 - b) *Narrow detection phase*: Examina los pares que en la *Broad detection phase* se han detectado en una posible intersección y comprueba si realmente lo están. Para ello existen diferentes mecanismos denominados *intersection methods*. La salida consistirá en una serie de pares de primitivas, esto es, conjuntos de vectores, aristas, triángulos y otro tipo de componentes geométricos que están en colisión.
3. *Collision response phase*: Procesa la respuesta de la colisión. Existen diferentes tipos de respuesta que pueden definirse en el componente *collision manager*.

Se pueden observar de manera global los componentes que interactúan en cada fase de la *collision pipeline* en la figura 51. Las distintas clases disponibles para cada componente podrán consultarse en el apéndice F.3.

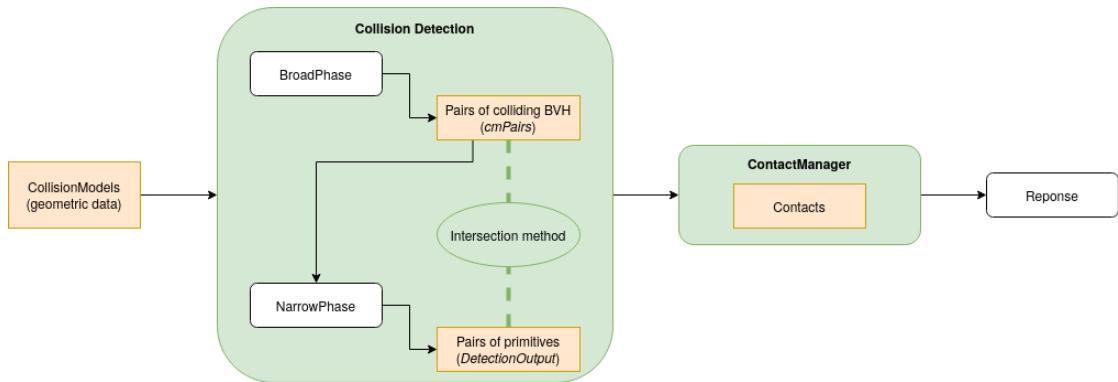


Figura 51: Esquema de las diferentes fases de la *collision pipeline*.

H. Modelos de estructuras de datos

En función del tipo de información que contengan los vectores de estado podrán usarse vectores de diferentes dimensiones para representar dicha información y su utilización estará enfocada a distintos ámbitos.

- Vec1d: Contiene información de la temperatura, usada para representaciones termo-dinámicas de objetos.
- Vec2d: Enfocado a la representación de modelos usados para electrofisiología cardíaca.
- Vec3d: Posee información de coordenadas de los vértices y permite representar modelos físicos.
- Rigid3d: A mayores de las coordenadas, contiene también información de la orientación de los vértices. Permite representar cuerpos no deformables.

I. Mallas utilizadas

En las siguientes tablas podemos observar el número de componentes aproximado de las mallas utilizadas a lo largo de las pruebas, teniendo en cuenta que aquellas categorizadas como mallas finas en la tabla 3 son las utilizadas generalmente en el modelo visual, mientras que su versión reducida en la tabla 4, con menos componentes, es la utilizada en los modelos físico y de colisiones.

Conviene indicar que las mallas finas, al ser superficiales y no contener información del volumen, no disponen de información relativa a poliedros.

Mallas finas		
Nombre	Nº de vértices	Nº de triángulos
Intestino	16300	32600
Intestino 2	12200	24400
Intestino 3	9800	19500
Intestino 4	7800	15600
Intestino 5	6300	12500
Esfera	200	1440

Tabla 3: Mallas utilizadas en las pruebas para los modelos visuales y su cantidad de componentes

I Mallas utilizadas

Mallas reducidas				
Nombre	Nº de vértices	Nº de aristas	Nº de triángulos	Nº de tetraedros
Intestino	4100	20400	28600	12300
Intestino 2	3300	16500	23100	10000
Intestino 3	2600	13200	18500	7900
Intestino 4	2100	10500	14800	6400
Intestino 5	1700	8400	11900	5000
Intestino cerrado	2700	11900	15600	6500
Hígado	200	900	1300	600
Esfera	100	600	900	400

Tabla 4: Mallas utilizadas en las pruebas para los modelos físicos y de colisiones y su cantidad de componentes