UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

GENERACIÓN DE SKELETONS A PARTIR DE MALLAS DE SUPERFICIE

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL EN COMPUTACIÓN

LILIANA FRANCISCA ALCAYAGA GALLARDO

PROFESOR GUÍA:
SR. STEFFEN HÄRTEL GRUNDLER

MIEMBROS DE LA COMISIÓN:
SRA. NANCY HITSCHFELD KAHLER
SR. PATRICIO INOSTROZA FAJARDIN

SANTIAGO DE CHILE
ABRIL 2012

Resumen

El modelamiento y análisis de estructuras biológicas microscópicas 3D de alta ramificación es una tarea desafiante debido a su alta complejidad. Un método para abordar esta tarea corresponde a la generación de *skeletons*, como modelos de dimensión reducida. El *skeleton* de un objeto 3D es una representación 1D del mismo, aproximadamente equidistante a los bordes y que busca conservar sus propiedades geométricas y topológicas. Los *skeletons*, aplicados a estructuras biológicas complejas de interés, requieren satisfacer las siguientes propiedades: ser unidimensionales, invariantes bajo transformaciones isométricas, aproximadamente centrados y homeotópicos.

El objetivo de este trabajo de título fue implementar un algoritmo de esqueletonización correcto y robusto, tomando como base un método descrito para mallas triangulares de superficie 3D.

El algoritmo implementado considera tres etapas: una de contracción de la geometría, una de remoción de todas las caras de la malla que la transforma en una estructura unidimensional, y una de centrado para la corrección del *skeleton* resultante. Para garantizar la robustez del algoritmo, se añadió una rutina de preprocesamiento que verifica que la malla de entrada sea válida; además se realizaron pruebas unitarias para validar los distintos escenarios posibles en las tres etapas del método.

En esta implementación se utilizó el paradigma de programación orientada a objetos y de patrones de diseño para facilitar la extensión y modificación del *software*. Se evaluó la implementación del algoritmo y de las mejoras propuestas utilizando (i) mallas simples de figuras de fantasía, presentadas en trabajos previos sobre esqueletonización , y (ii) mallas complejas de estructuras biológicas observadas por microscopía confocal. En el último caso se recurrió a biólogos expertos para evaluar los resultados.

Al emplear este método con mallas biológicas de distinto tamaño y complejidad, se obtienen *skeletons* correctos que satisfacen las propiedades requeridas. En particular se utilizaron mallas de superficie de: red de retículo endoplasmático de células de cultivo COS-7, cuerpo y soma de neuronas pertenecientes al órgano parapineal del pez cebra y conglomerados de membranas plásticas de células de cresta neural de pez cebra. Al emplearlo con las mallas de fantasía se obtienen *skeletons* aproximadamente centrados en casi todos los casos, y en un caso se observó una región del *skeleton* que quedó ubicada fuera de la figura original. Además, en este último tipo de mallas, se observa que el tener una malla simétrica no implica que el *skeleton* resultante sea simétrico. Para todos los casos la aplicación realizada cumple los requerimientos de robustez en las tres etapas del algoritmo.

Finalmente, la extensión del trabajo realizado en proyectos futuros abarca temas como: la paralelización de la aplicación, y mejoras para garantizar que el *skeleton* se encuentre siempre dentro de la malla y aproximadamente centrado con respecto a ésta.

Agradecimientos

Quisiera agradecer a Steffen Härtel y a Nancy Hitschfeld por la disposición y el apoyo prestado durante el desarrollo de esta memoria.

También quisiera agraceder el financiamiento de este trabajo a la Comisión Nacional de Investigación Científica y Tecnológica CONICYT a través del proyecto Fondecyt 1090246 y al Instituto de Ciencias Biomédicas BNI a través del proyecto ICM P09-015-F.

Para terminar, quisiera agradecer a todas las personas que me han apoyado incondicionalmente durante mis años de estudio, especialmente a mis padres Jorge y Juany, a mi hermano Matías, a mis amigos y a las familias Gallardo Araya y Díaz Céspedes por su comprensión y cariño.

Índice General

Índice de Figuras

Índice de Algoritmos

Capítulo 1

Introducción

1.1. Antedecentes Generales

Una de las principales aplicaciones de la computación gráfica es modelar y representar la realidad lo más fielmente posible. Para ello, es necesario contar con herramientas matemáticas y computacionales e infrastructura apropiadas que permitan realizar un correcto análisis e interpretación del problema a modelar.

El campo de la geometría computacional estudia algoritmos y estructuras de datos eficientes para resolver problemas geométricos computacionalmente. Las estructuras biológicas representan modelos geométricos donde caracterizar sistemas cuya complejidad no puede ser estimada a simple observación es una tarea desafiante. Un enfoque propuesto para acceder a la información en estructuras biológicas complejas radica en la simplificación de formas reduciendo su dimensionalidad a través de la esqueletonización.

El *skeleton* o *curve-skeleton* [7] de un objeto bidimensional o tridimensional es una representación unidimensional del mismo, aproximadamente equidistante a los bordes que busca conservar sus propiedades geométricas, morfológicas y topológicas (Apéndice A, Figura 9.1).

Los *skeletons* unidimensionales han sido definidos en más de una manera, es por esto que existen muchos algoritmos de esqueletonización, basados en distintas estrategias y algunos orientados a funcionar mejor en casos específicos. Usualmente dan resultados muy parecidos, pero a veces pueden presentar diferencias dependiendo del algoritmo que se utilice, en particular al aplicarlos a datos discretos. Las dos definiciones más comunes son:

- Eje y superficie medial: de acuerdo a la definición de [4], en el caso bidimensional o tridimensional el eje medial es el locus de puntos de la figura que tiene al menos dos puntos más cercanos con el borde de la figura (Apéndice A, Figura 9.2). En el caso tridimensional, esto se extiende a la superficie medial, ya que además de curvas puede contener superficies. Por este motivo, no puede ser utilizado directamente en el caso tridimensional, sin embargo, sí es posible utilizarla como punto de partida para generar un *skeleton* unidimensional.

- Centro de discos o bolas maximales: esta es la definición más aceptada de un *skeleton*. Consiste en el locus de puntos que corresponden a centros de discos (en el caso

bidimensional) o esferas (en el caso tridimensional) maximales abiertos inscritos en el objeto. Formalmente: sea $X \in R^3$ una forma tridimensional[1]. Se define una bola abierta centrada $x \in X$ y de radio r como $Sr = \{y \in R^3, distancia(x, y) < r\}$. Una bola $Sr(x) \subseteq X$ es maximal si no está en ninguna otra bola maximal (Apéndice A, Figura 9.2).

El hecho que existan diversas maneras de definir un *skeleton* produce algunos inconvenientes, por ejemplo, no siempre el *skeleton* obtenido por medio de la aplicación estricta de una definición es deseable, ya que se puede ver muy afectado por ruido en la superficie del modelo tridimensional, o también puede no ser el mismo si se aplica el algoritmo al modelo en una diferente orientación. Además, son muy sensibles a pequeños cambios en la superficie del objeto, lo que resulta poco práctico para aplicaciones donde es necesario tener robustez frente a la presencia de ruido o pequeñas variaciones en la superficie.

Algunas de las propiedades deseables de un *curve-skeleton* son las siguientes [7]:

- Homeotópico: el *skeleton* debe ser topológicamente igual al objeto original, es decir, debe tener el mismo número de componentes conectados, túneles y cavidades. Además, el *skeleton* debe estar contenido completamente en el objeto original; de manera contraria, se puede cumplir la condición anterior aunque el *skeleton* tenga una geometría diferente. Como no es posible tener cavidades en un objeto unidimensional como un *skeleton*, en este caso se puede utilizar una definición relajada de topología, y decir que el *skeleton* debe tener el mismo número de componentes conectados, y al menos un *loop* por cada túnel y cavidad del objeto original (entendiéndose *loop* en el *skeleton* como un conjunto de puntos en el cual es posible partir desde uno de sus puntos y llegar al mismo recorriendo todos los demás sin repetir ninguno).

- Invariante bajo transformaciones isométricas: dada una transformación isométrica, el *skeleton* de un objeto transformado debe ser igual a la transformación de un *skeleton* del objeto original. Formalmente: dada una transformación isométrica T y un método para extraer el *skeleton* de un objeto Sk, el resultado debe ser el mismo si se aplican en cualquier orden a un objeto O; es decir, $T(Sk(O)) = Sk(T(O))$.

- Unidimensional: el *skeleton* debe ser unidimensional y en el caso discreto a lo más debe tener un voxel de espesor en todas direcciones, excepto en las uniones donde convergen varias ramas.

- Centrado: el *skeleton* debe estar aproximadamente centrado repecto al objeto que representa. Esto se puede cuantificar por ejemplo, midiendo la distancia de cada punto del *skeleton* al borde del objeto a lo largo de varios segmentos perpendiculares a la dirección del *skeleton*; los puntos centrados deberían tener la misma distancia al borde en cada par de segmentos colineales. En algunos casos, como compresión de forma, es importante que el *skeleton* esté perfectamente centrado, pero en la mayoría de los casos un *skeleton* perfectamente centrado no es bueno ya que se reflejarían en él pequeñas imperfecciones indeseadas de la superficie del objeto; en estos casos es mejor contar con un *skeleton* aproximadamente centrado.

[1]Es decir, su borde y su interior.

- Robusto: el *skeleton* debe ser poco sensible al ruido en el borde del objeto. Esta propiedad está en conflicto con la anterior; es decir, un *skeleton* robusto no puede ser perfectamente centrado. En la mayoría de los casos, un *skeleton* robusto y aproximadamente centrado es lo mejor.

- Suave: se dice que un *skeleton* es suave cuando al moverse a lo largo de él no existe mucha variación en la dirección de la curva tangente. Se puede medir la suavidad como la desviación estándar de ángulos entre direcciones tangentes en ubicaciones sucesivas del *skeleton*; para asegurar suavidad, esta variación al moverse de un punto a otro debe ser lo menor posible [7]. Esta propiedad es útil en aplicaciones como navegación virtual, en donde el skeleton, que es utilizado como un camino de navegación, debe ser lo más suave posible para evitar cambios abruptos en la imagen mostrada.

El actual estado del arte de los distintos enfoques para la esqueletonización dan lugar a herramientas eficaces, precisas y robustas que son escenciales para el estudio de complejas estructuras biológicas altamente ramificadas. Resulta entonces interesante realizar la implementación de un algoritmo de esqueletonización para apoyar este tipo de estudios.

1.2. Motivación

El *SCIAN-lab* (*Scientific Image Analysis*)[2] es un laboratorio de procesamiento de imágenes científicas, ubicado en la Facultad de Medicina de la Universidad de Chile y dirigido por el profesor Steffen Härtel[3]. Forma parte del Programa de Anatomía y Biología del Desarrollo y de las iniciativas interdisciplinarias del Instituto Milenio de Neurociencia Biomédica (BNI)[4] y de la Advanced Imaging and Bioinformatics Initiative (AIBI)[5], en colaboración con el Departamento de Ciencias de la Computación y la Facultad de Ciencias Físicas y Matemáticas. El *SCIAN-lab* desarrolla herramientas matemáticas y algoritmos computacionales para acceder a las funciones dinámicas, morfológicos y topológicos en sistemas experimentales en el ámbito biofísicos, biológicos o médicos. Microscopía confocal de alta velocidad en combinación con procesamiento de imágenes revela la interacción entre estructura y función en sistemas modelo de lípidos y fenómenos a niveles sub-celular, celular, y supra-celular en el campo de la biología del desarrollo, neurobiología, o la biofísica de membranas.

De la diversidad de sistemas biológicos estudiados en *SCIAN-lab*, los que presentan mayor complejidad y ramificación no disponen en la actualidad de herramientas para su análisis y cuantificación. Entre ellos podemos considerar: la relación estructural y funcional de circuitos neuronales del cerebro del pez cebra [14], la dinámica y cambios morfológicos presentes en el retículo endoplasmático [21] y los cambios morfológicos en el conglomerado de membranas plasmáticas de células de cresta neural de pez cebra [22].

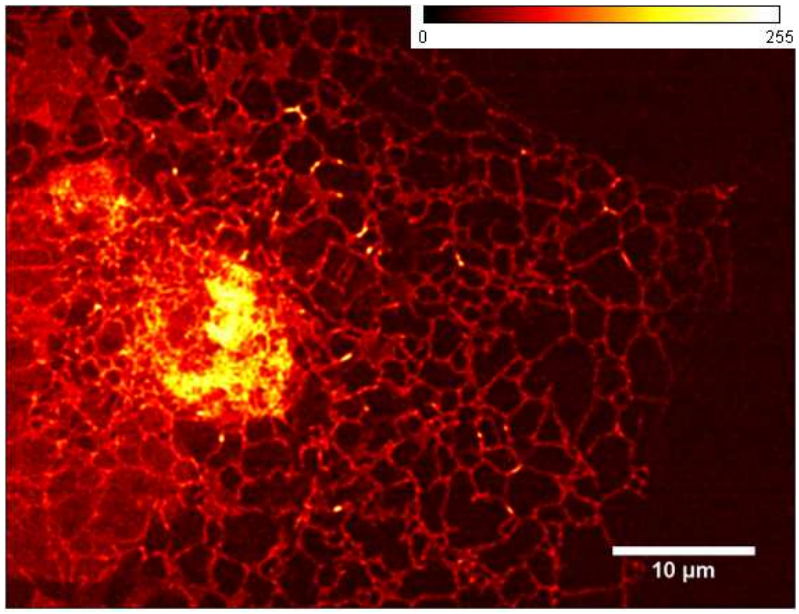


Figura 1.1: Red de retículo endoplasmático de células de cultivo COS-7 obtenidas a partir de riñon de mono verde africano (Cercopithecus aethiops) observado por microscopía confocal. El marcador utilizado corresponde a una proteína fluorescente llamada RFP (*red fluorescent protein*).

[2] www.scian.cl
[3] shartel@med.uchile.cl
[4] www.bni.cl
[5] www.aibi.cl

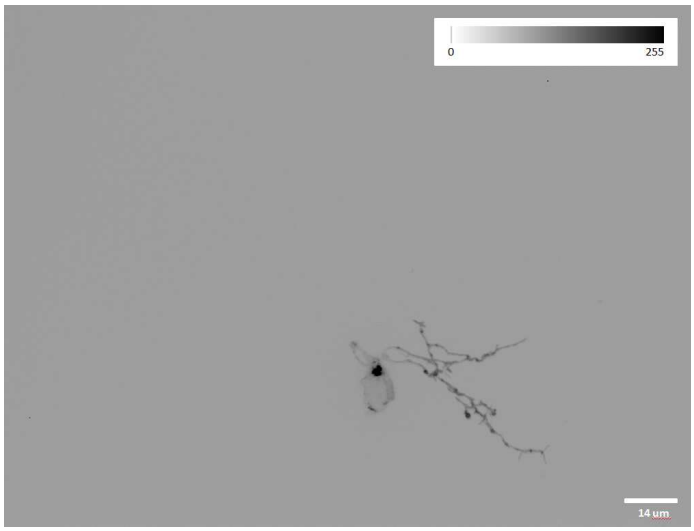


Figura 1.2: Proyección 2D de una imagen tridimensional de neurona perteneciente al órgano parapineal del pez cebra obtenida mediante microscopía confocal. El marcador utilizado corresponde a una proteína fluorescente llamada *m-cherry*.

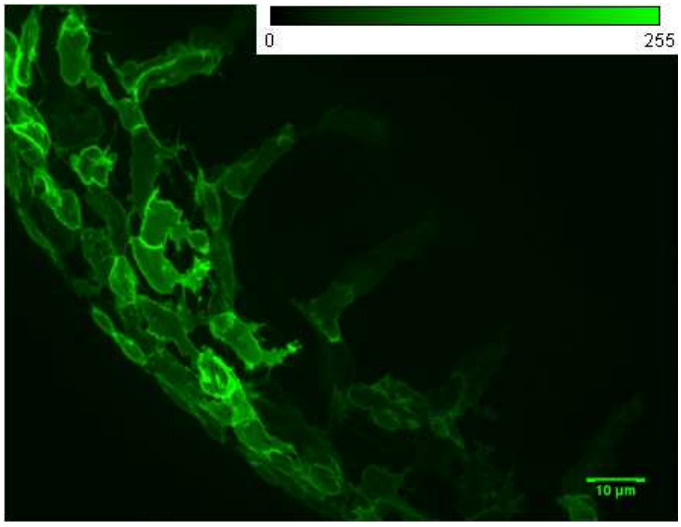


Figura 1.3: Proyección 2D de una imagen tridimensional de un conglomerado de las membranas plasmáticas de células de cresta neural de un pez cebra obtenida mendiante microscopía confocal. El marcador utilizado corresponde a una proteína fluorescente llamada GFP (*green fluorescent protein*).

Para el estudio de estos sistemas biológicos se emplean series de datos 2D y 3D obtenidas a partir de microscopía confocal. Estas series están compuestas de imágenes que tienen una resolusión de unos cientos de nanómetros por pixel. Con el equipamiento actual de *SCIAN-lab* las imágenes 2D obtenidas tienen una resolución de 1024×1344 pixeles ($\sim$ 1MB). En el caso de las series 3D se observan hasta 70 cortes ($1024 \times 1344 \times 70$, $\sim$ 70 MB), 8 bits TIFF sin comprensión.

Este trabajo de título busca apoyar la labor de estos profesionales y aborda el problema de análisis morfológico y topológico de estos sistemas altamente ramificados mediante

esqueletonización [1]. El modelamiento de estos sistemas y sus componentes a través de una estructura más simplificada que una malla geométrica logra una mejor comprensión de sus asimetrías, proyecciones y conectividades. Los *skeletons* permiten cuantificar estos sistemas biológicos mediante la obtención de información relevante como: número y nivel de bifurcaciones de los nodos, largo de los segmentos, ángulo entre nodos, etc. Estas características logran describir de mejor forma los cambios morfológicos de estos sistemas biológicos, junto con proporcionar estructuras simplificadas que facilitan su visualización.

Al inicio del proyecto, se disponía de un prototipo para esqueletonización incompleta no funcional, que sólo implementaba un criterio de contracción [1], y parcialmente integrado al software de tratamiento y análisis de imágenes de *SCIAN-lab*. En adición, los casos de aplicación biológicos existentes conforman el marco de trabajo para el presente proyecto.

1.3. Objetivos

1.3.1. Objetivos Generales

- Implementar un algoritmo correcto y robusto para la generación de un *skeleton* 1D de un objeto 3D a partir de su malla de superficie. Correcto quiere decir que entregue una respuesta que satisfaga las propiedades definidas para el *skeleton* cuando la entrada es válida y robusto quiere decir que el algoritmo no falle con entradas no esperadas o por problemas de precisión[6].

- Visualizar la estructura resultante para el *skeleton* en *SCIAN-Soft*.

1.3.2. Objetivos Específicos

- Mejorar el prototipo existente (*skeleton extraction by mesh contraction*) [2] e implementar las etapas faltantes para generar el *skeleton*. Éstas son:

  1. Contracción de la malla (*geometry contraction*).
  2. Colapso de la malla contraída (*connectivity surgery*).
  3. Corrección del *skeleton*, si es que hay partes que quedan ubicadas fuera de la malla original (*embedding refinement*).

- Validar el prototipo con mallas de superficie generadas artificialmente y con mallas reales para estructuras 3D a nivel subcelular y supracelular, de distinta complejidad y tamaño.

- Implementar la visualización del *skeleton* de un objeto en *SCIAN-Soft*.

[6] Algunos de los problemas de precisión y robustez en el campo de la Geometría Computacional son mencionados en [23].

1.4. Contenido de la Memoria

1. Introducción: Se presenta una mirada general al tema y la motivación. Se señalan los objetivos de este trabajo.

2. Marco Teórico: Se presentan los elementos necesarios para el entendimiento del problema a solucionar. Estos incluyen: el procesamiento de imágenes y generación de mallas de superficie, revisión del *software* recibido y revisión del método parcialmente implementado.

3. Análisis: Se exponen una revisión detallada de los problemas de implementación del método parcialmente implementado y las alternativas analizadas para mejorar y extender el software existente.

4. Diseño: Se presentan las estructuras de datos utilizadas, el detalle del diseño del algoritmo y el diseño de la aplicación.

5. Implementación: Se exponen los detalles de la implementación de la aplicación.

6. Evaluación: Se expone la evaluación del diseño de la aplicación y de la implementación realizada.

7. Discusión: Se presenta la discusión de todos los aspectos significativos en este trabajo. Estos son: las características de las mallas de entrada, la implementación del algoritmo y su rendimiento.

8. Conclusiones: Se presentan las conclusiones obtenidas en este trabajo y los posibles trabajos futuros sobre la aplicación desarrollada.

9. Trabajo Futuro: Se presentan distintos aspectos para continuar el trabajo realizado en próximos proyectos.

10. Referencias: Se exponen las fuentes de información utilizadas en el marco teórico y en el diseño del algoritmo.

11. Apéndice: Se presenta información adicional de interés sobre el trabajo realizado.

Capítulo 2

Marco Teórico

En este capítulo se entregan antecedentes para facilitar la comprensión del trabajo realizado. Estos son: el contexto de este trabajo de memoria, la revisión del *software* existente y la revisión bibliográfica de éste.

2.1. Procesamiento de Imágenes y Generación de Mallas de Superficie

2.1.1. Captura de Imágenes

En el contexto del *SCIAN-lab*, para estudiar el comportamiento de estructuras biológicas se utiliza microscopía confocal. La microscopía confocal permite obtener y cuantificar datos asociados a características físicas de las estructuras biológicas en observación. El microscopio captura imágenes de las muestras observadas sobre varios planos (x, y) a través del eje z, almacenándolas en forma de *stack* en el computador.

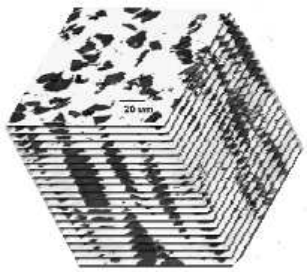


Figura 2.1: Representación de un *stack* de imágenes apiladas en el eje z obtenidas mediante microscopía confocal.

2.1.2. Generación de Mallas de Superficie

SCIAN-Soft es un programa para tratamiento y análisis de imágenes científicas desarrollado en el *SCIAN-lab*. Este *software* se encuentra implementado en IDL[1] y proporciona soluciones *ad hoc* a las necesidades del laboratorio, por ejemplo, análisis sobre imágenes de

[1]Interactive Data Language. El detalle de la descripción de IDL se encuentra en el Capítulo 5.3.

estructuras celulares.

SCIAN-Soft permite crear un volumen 3D de vóxeles a partir de un *stack* de imágenes que son tomadas a las muestras biológicas de interés a través del eje *z*. Luego, se utiliza este volumen para producir una lista de vértices y polígonos que describen el contorno de una superficie. *SCIAN-Soft* utiliza el método *getSurfaceModel* para generar mallas de superficie. *PolyPaint*, el método que utiliza *getSurfaceModel* para generar mallas de superficie, está basado en el algoritmo *Marching Cubes* descrito en [17].

El algoritmo *PolyPaint* (y también *Marching Cubes*) puede generar mallas de superficie no conformes, en particular, las mallas presentan elementos no conectados entre sí. Para solucionar este problema, las mallas de superficie son reparadas, obteniendo mallas aptas para utilizar en este *software*.

2.2. Revisión del Método de Esqueletonización

En el *software* existente se intentó implementar el método *skeleton extraction by mesh contraction* [2], que a partir de una malla de superficie busca obtener el *skeleton* de un objeto. Este método está compuesto de tres etapas:

- contracción de vértices (*geometry contraction*): proceso basado en el operador de Laplace que contrae y suaviza la malla de superficie hasta alcanzar un volumen cercano a cero.
- colapso de aristas (connectivity surgery): a través de un proceso de colapso de arcos se remueven todas las caras de la malla contraída hasta obtener un *skeleton* unidimensional.
- centrado de nodos del skeleton (*embedding refinement*): se realiza un proceso que centra los nodos del *skeleton* que se encuentren fuera de la malla original.

El *software* recibido sólo intentó implementar la etapa de contracción (*geometry contraction*) utilizando el método de contracción basado en el Laplaciano [2]. Los detalles del diseño de este método son descritos a continuación.

2.2.1. Método de Contracción Basado en el Operador de Laplace

Consiste en un proceso de suavizado y contracción que remueve detalles y ruido de la malla de superficie desplazando cada uno de los vértices a través de su dirección normal. El enfoque sugerido en [2] consiste en una contracción mediante la aplicación del método *Laplacian smoothing*. En esta propuesta, los vértices de posiciones $V^{'}$ son (suavemente) contraídos a través de sus direcciones normales para resolver la ecuación discreta de Laplace:

$$LV^{'} = 0 \tag{2.1}$$

donde L es una matrix n x n llamada operador curvatura-flujo de Laplace con elementos

$$L_{i,j} = \begin{cases} w_{i,j} = cot(\alpha_{i,j}) + cot(\beta_{i,j}) & \text{si } e_{(i,j)} \in E \\ \sum_{e_{(i,k)} \in E}^{k} -w_{i,k} & \text{si } i = j \\ 0 & \text{en otro caso} \end{cases} \tag{2.2}$$

donde $\alpha_{i,j}$ y $\beta_{i,j}$ son los ángulos opuestos correspondientes al arco $e_{(i,j)}$ [9] (Figura 2.2).

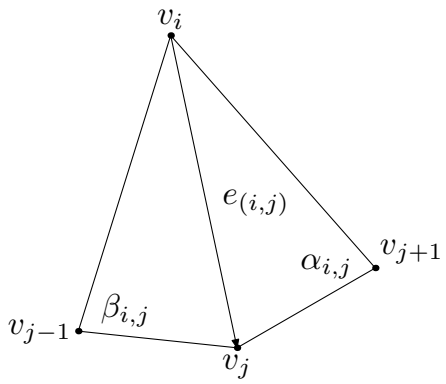


Figura 2.2: Ángulos $\alpha_{i,j}$ y $\beta_{i,j}$ correspondientes al arco $e_{(i,j)}$

De esta forma, las coordenadas del Laplaciano $\delta = LV = [\delta_1^T, \delta_2^T, ..., \delta_n^T]^T$ aproximan interiormente las normales curvatura-flujo, es decir,

$$\delta_i = -4A_i\kappa_i \mathrm{n}_i \tag{2.3}$$

donde A_i, κ_i y n_i son el área local del anillo (*one-ring* en inglés), la aproximación de curvatura media local y la aproximación de la normal exterior del vértice i, respectivamente [9]. En el contexto de mallas geométricas, se define el anillo (*one-ring*) [13] de un vértice v_i como el conjunto de todos los vértices que son vecinos de v_i. Esto se puede extender a la vecindad de dos anillos de un vértice (*two-ring*) agregando todos los vecinos de los vértices del primer anillo. Luego, el área local del anillo A_i corresponde al área de todos los triángulos que contienen el vértice v_i.

Para el cálculo del vector normal, en [2] se sugiere utilizar la propuesta de [9]. Este enfoque propone obtener la curvatura media y una aproximación al vector normal a partir de la siguiente definición:

$$\frac{\bigtriangledown A}{2A} = \overline{\kappa}\mathrm{n} \tag{2.4}$$

donde A es el área del anillo de v_i y $\bigtriangledown$ es la derivada con respecto a las coordenadas (x, y y z) de v_i. La versión discreta de la ecuación anterior es equivalente a:

$$-\kappa_i \mathrm{n}_i = \frac{1}{4A_i} \sum_{j \in N_1(i)} (cot(\alpha_{i,j}) + cot(\beta_{i,j}))(v_j - v_i) \tag{2.5}$$

donde $N_1(i)$ es el anillo (*one-ring*) o vecindad del vértice v_i.

Por lo tanto, resolver la ecuación $LV^{'} = 0$ significa también contraer la malla geométrica en función de sus componentes normales. Las filas del sistema $LV^{'} = 0$ corresponden a las restricciones de contracción porque ellas definen la fuerza para contraer la malla. Además, se introducen restricciones de atracción que tienen por objetivo atraer los vértices a la geometría original de la figura.

Se define el siguiente sistema para las posiciones de los vértices:

$$\begin{bmatrix} W_L L \\ W_H \end{bmatrix} V^{'} = \begin{bmatrix} 0 \\ W_H V \end{bmatrix} \tag{2.6}$$

donde W_L y W_H son las matrices diagonales de n x n con los coeficientes que ponderan las restricciones de contracción y atracción, respectivamente. El elemento i-ésimo de la diagonal de las matrices W_L y W_H se denota $W_{L,i}$ y $W_{H,i}$, respectivamente. Además, la ecuación (6) es equivalente a

$$W_L L V^{'} = 0 \tag{2.7}$$
$$W_H V^{'} = W_H V \tag{2.8}$$

Dado que el sistema de la ecuación (6) esta sobre-determinado[2], una posible solución se obtiene al emplear la técnica de los mínimos cuadrados, que equivale a minimizar la siguiente expresión cuadrática:

$$\left\| W_L L V^{'} \right\|^2 + \sum_i W_{H,i}^2 \left\| v_i^{'} - v_i \right\|^2, \tag{2.9}$$

donde el primer término corresponde a las restricciones de contracción y el segundo término corresponde a las restricciones de atracción.

Se requieren varias iteraciones con las ponderaciones adecuadas para que el proceso converga a una figura delgada. Para incrementar la velocidad de colapso, se puede incrementar la ponderación de contracción $W_{L,i}$ para todos los vértices i después de cada iteración. Además, para evitar una sobrecontracción de las características importantes de la malla, se actualizará la ponderación de la atracción $W_{H,i}$ para cada vértice de acuerdo a su grado de colapso determinado por su área de anillo local. Específicamente, se desea que los vértices que se hayan contraído menos se atraigan más fuertemente a sus posiciones actuales y que la contracción sea menor en la siguiente iteración.

El proceso de contracción iterativo, para cada iteración $t + 1$ con $t > 0$ es el siguiente:

1. Resolver $\begin{bmatrix} W_L^t L^t \\ W_H^t \end{bmatrix} V^{t+1} = \begin{bmatrix} 0 \\ W_H^t V^t \end{bmatrix}$ para V^{t+1}.

2. Actualizar $W_L^{t+1} = s_L W_L^t y W_{H,i}^{t+1} = W_{H,i}^0 \sqrt{\frac{A_i^0}{A_i^t}}$ donde A_i^t y A_i^0 son el actual y el original área del anillo de cada vértice, respectivamente y s_L es la constante de actualización de las restricciones de contracción.

[2]Se dice que un sistema lineal este sobre-determinado cuando éste tiene más ecuaciones que incógnitas.

3. Calcular el nuevo operador de Laplace L^{t+1} con las actuales posiciones de lo vértices V^{t+1}.

El valor inicial de los coeficientes $W^0_{L,i}$ y $W^0_{H,i}$ ponderan respectivamente la suavidad y el grado de contración de la primera iteración, junto con determinar la cantidad de detalle retenida en la subsiguiente y final malla contraída. Para manejar modelos de diferentes tamaños y resoluciones los autores utilizan el valor inicial $W^0_{H,i} = 1.0$ y $W^0_{L,i} = 10^{-3}\sqrt{A}$, donde A es el promedio del área de todas las caras del modelo. Empíricamente [2] se han obtenido *skeletons* de buena calidad, reteniendo las principales características en los modelos típicos de computación gráfica.

2.3. Revisión del *Software* Recibido

El *software* existente consiste en un proyecto del IDE *eclipse*[3] desarrollado en *Java* por Pablo Aguilar.

La clase principal es *MeshContraction3DTriangles*. Esta clase tiene 1801 líneas de código y 13 funciones. Estas funciones intentaron implementar el método de contracción basado en el operador de Laplace descrito anteriormente. Además, existen funciones dedicadas a leer y escribir archivos en formato *Wavefront* (.obj) para cargar y guardar mallas de superficie, respectivamente.

Las clases restantes definen las estructuras de datos utilizadas (*Mesh*, *MeshTriangle*, *MeshVertex* y *MeshEdge*) y resuelven operaciones matriciales (*MatrixUtils, LinearEquationSystemDoolittleSolver*).

La implementación del software recibido dista bastante de la definición de buen código mencionada en [18], y no incorpora patrones de diseño [11] que permitan su fácil extensión y mejoramiento.

Al evaluar el *software* existente no se obtuvieron buenos resultados, pues la etapa de contracción falló incluso con mallas simples como un paralelepípedo y un toroide.

[3] http://www.eclipse.org/

Capítulo 3

Análisis

En este capítulo se realiza la revisión de los problemas de implementación encontrados en el *software* recibido. Para corregir y extender este *software*, se analizan dos alternativas que abordan la esqueletonización de objetos 3D y se presenta la justificación y descripción de la solución escogida.

3.1. Problemas de Implementación

El enfoque sugerido originalmente en [2] (Capítulo 2.3) consiste en una contracción mediante la aplicación del método *laplacian smoothing*. Uno de los problemas encontrados en este enfoque es que el Laplaciano de dos objetos con características muy diferentes (por ejemplo, la curvatura) pueden ser el mismo, como se ha descrito en [9].

Otro problema se relaciona con el cálculo de la normal sugerido en [9], en particular con el cálculo de los ángulos $\alpha_{i,j}$ y $\beta_{i,j}$ que definen el arco $e_{(i,j)}$ (Figura 2.2). En las implementaciones de algoritmos geométricos la aritmética de los números reales es reemplazada por la aritmética del punto flotante, produciendo a veces salidas incorrectas [23].

3.2. Alternativas Analizadas

Existen diversos enfoques para abordar la esqueletonización de un objeto tridimensional [3]. Algunos de ellos son: algoritmos de adelgazamiento topológico, algoritmos de mapas de distancia (en dominios discretos basados en diagramas de Voronoi) y algoritmos basados en morfología matemática.

Para seleccionar el enfoque adecuado es necesario tener en cuenta el tipo de modelos que se desean evaluar con esqueletonización.
Dos de los sistemas biológicos que se analizan en *SCIAN-lab*, el retículo endoplasmático de células de cultivo y el conglomerado de las membranas plasmáticas de células de cresta neural de pez cebra (Figuras 1.1 y 1.3), se pueden clasificar como estructuras 3D tipo grafo-árbol debido a la alta ramificación que presentan. Considerando estas características, las posibles estrategias para abordar el problema se redujeron a las siguientes:

1. Nube de puntos (*Robust skeleton extraction from imperfect point clouds*) [5]

   La aplicación típica de esta estrategia corresponde al empleo de *lasers* para la reconstrucción de objetos y superficies.

   El algoritmo de esqueletonización a partir de una nube de puntos propone los siguientes pasos:

   a) Obtener una nube de puntos a partir del espacio discreto (vóxeles[1]).

   b) Crear una estructura de árbol jerárquico (*octree*[2]) a partir de la nube de puntos.

   c) Reducir el árbol a un *skeleton*.

   d) Realizar un post proceso de corrección del *skeleton* resultante.

   La extracción de complejas estructuras tipo árbol-grafo desde una nube de puntos es difícil por muchas razones, por ejemplo:

   - La variación en la densidad local de puntos causada por el alineamiento del *scanner* en un sistema de coordenadas común.
   - El submuestreo causado por efectos de oclusión.
   - Ruido y errores sistemáticos que enmascaran la estructura del objeto de interés.

   Una de las ventajas de este enfoque es que al disminuir homogéneamente la densidad de puntos, el *skeleton* final no se ve mayormente afectado por esta variación.

   En general, se emplea este algoritmo [5] cuando se desea obtener el *skeleton* de árboles y arbustos (por la naturaleza de estos objetos y por la búsqueda de resultados descriptivos a nivel global de las estructuras y distribución de ramas por sobre cuantificadores demasiado estrictos).

   Este método presenta las siguientes complicaciones:

   - Debido a que el método se basa en una subdivisión espacial, resulta ser muy sensible a rotaciones sobre el volumen de datos, por lo que no es invariante a transformaciones isométricas.
   - La profundidad de los *octrees* generados (el nivel de subdivisiones) es un parámetro del método, por lo que no se garantiza la conservación de la topología.
   - Debido a lo anterior, no se puede garantizar que el resultado sea homeotópico al objeto original.

[1] Un vóxel (representación 3D de un píxel) es una unidad cúbica de volumen que representa un valor (por ejemplo, propiedades medibles) en una grilla regular en un espacio tridimensional. La voxelización consiste en convertir objetos de su representación geométrica continua al conjunto de voxeles que más se aproxima al objeto continuo. Para más detalles, ver [15].

[2] En el contexto de las estructuras de datos, un *octree* es un árbol donde cada nodo interno tiene ocho hijos. Un *octree* corresponde a una partición tridimensional del espacio que se subdivide recursivamente en ocho celdas. Los *octrees* son la versión 3D de los *quadtrees*, estructura de datos diseñada por Raphael Finkel y J.L. Bentley [10].

2. Contracción de malla geométrica (*Skeleton extraction by mesh contraction*) [2]

   Dada una malla de superficie $G = (V, F, E)$ con vértices V, caras F y arcos E, siendo $V = [v_1, v_2, ..., v_n]$ las posiciones de los vértices, el problema consiste en extraer el *skeleton* $S = (N, U)$ de la malla, donde $N = [n_1, n_2, ..., n_m]$ y U son las posiciones de los nodos y los segmentos del *skeleton*, respectivamente.

   Este enfoque está basado en un proceso de contracción geométrico [2] que iterativamente suaviza y colapsa la malla geométrica de forma forzada. Con una contracción cuidadosamente ponderada, el proceso produce la figura de un *skeleton* delgado con uniones y ramas correspondientes a los componentes morfológicos y topológicos principales del objeto original.

   Las etapas de este algoritmo de esqueletonización son las siguientes:

   a) Contracción de la malla (*geometry contraction*): la malla es contraída hasta obtener una figura de volumen cercano a cero.

   b) Colapso de la malla contraída (*connectivity surgery*): a través de un proceso de colapso de aristas se remueven todas las caras de la malla contraída hasta obtener un *skeleton* unidimensional.

   c) Corrección del *skeleton* (*embedding refinement*): se realiza un proceso que centra y corrige las partes que se encuentren fuera de la malla original.

   Este método se escogió por las siguientes razones:

   a) La etapa de contracción geométrica no altera la conectividad de la malla original.

   b) La etapa de colapso de la malla retiene todos los túneles.

   c) El *skeleton* generado garantiza ser homotópico al objeto original.

   d) El método es invariante a las rotaciones.

Capítulo 4

Diseño

En este capítulo se analizan las decisiones de diseño tomadas en el desarrollo de este trabajo. Estas decisiones incluyen: el rediseño de las etapas del algoritmo de esqueletonización, las estructuras de datos utilizadas y el diseño de la aplicación incorporando patrones de diseño.

4.1. Diseño del Algoritmo

4.1.1. Contracción de la Malla (*geometry contraction*)

Consiste en un proceso de contracción que remueve detalles de una malla de superficie desplazando cada uno de los vértices a través de su dirección normal. El *input* de esta etapa corresponde a una malla de superficie conforme $G = (V, F, E)$ con vértices V, caras F y arcos E, siendo $V = [v_1, v_2, ..., v_n]$ las posiciones de los vértices.

Debido a los problemas de implementación mencionados en el Capítulo 3.1 que presenta el enfoque sugerido en [2] para la implementación de la etapa *geometry contraction* del algoritmo *skeleton extraction by mesh contraction* se consideraron tres alternativas. Estas alternativas de contracción están basadas en un método de contracción geométrica definido por la ecuación:

$$v_i^{t+1} = w_1 \; v_i^t + w_2 \; d_i^t \tag{4.1}$$

donde

- v_i^t es la posición del vértice v_i en la iteración t
- d_i^t es el desplazamiento del vértice v_i en la iteración t
- $w_1 > 0$ y $w_2 > 0$ son las ponderaciones a v_i^t y d_i^t, respectivamente.

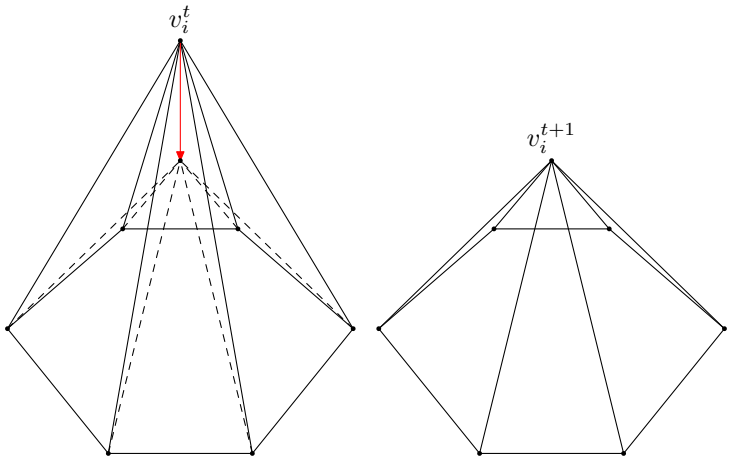


Figura 4.1: Vértice v_i antes (a) y después (b) de contraerlo en la dirección del vector d_i en la iteración t.

Este método de contracción simple busca mejorar los problemas de precisión que pueden ocurrir al implementar el enfoque sugerido en [2] cuando la calidad de los triángulos es deficiente[1]. Esto permitiría obtener *skeletons* independiente de la calidad de los triángulos de la malla de superficie. Los tres métodos de contracción geométrica que se implementaron son los siguientes:

- Contracción Geométrica I (*Geometry Contraction I*)

  En este enfoque el desplazamiento de cada vértice v_i de la malla está definido como el promedio de la posición de los vecinos de v_i :

$$v_i^{t+1} = w_1 \; v_i^t + w_2 \; d_i^t \tag{4.2}$$

$$= w_1 \; v_i^t + w_2 \left(\frac{\sum\limits_{e_{(i,k)} \in E} v_k^t}{n} \right) \tag{4.3}$$

  donde

  - v_k^t es la posición del vértice v_k en la iteración t
  - n es el número de vértices vecinos de v_i
  - $w_1 = w_2 = 0.5$.

[1]Se dice que la calidad de los triángulos de una malla es deficiente cuando éstos son obstusos y generados, es decir, cuando su arco de menor longitud es mucho menor que su circunradio [25].

Es importante señalar que la posición promedio de los vecinos de v_i no es un vector normalizado y en la mayoría de los casos no coincide con el vector normal de v_i. El único caso en que este vector apunta en la misma dirección que la normal ocurre cuando los vecinos de v_i son equidistantes a este vértice (Figura 4.2). Esto indica que este método es sensible al cambio local en el largo de los arcos de la malla.

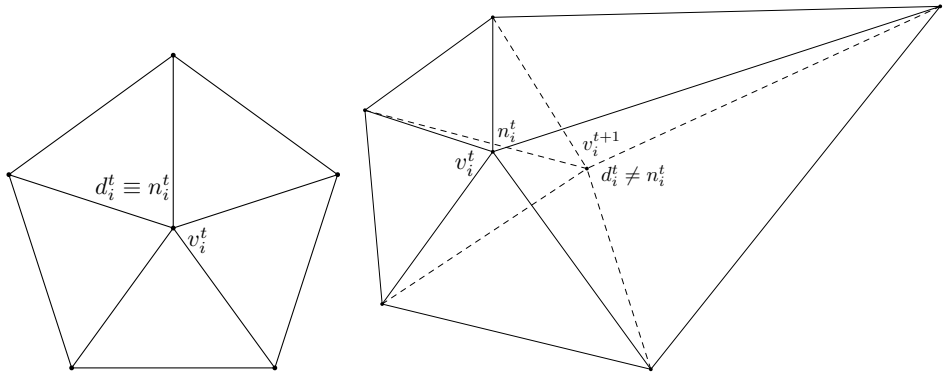


Figura 4.2: (a) Vértices vecinos de v_i equidistantes a este vértice. En este caso d_i^t apunta en la misma dirección que vector normal asociado a v_i (n_i^t). (b) Vértices vecinos de v_i que no son equidistantes a este vértice. En este caso d_i^t no coincide con n_i^t.

- Contracción Geométrica II (*Geometry Contraction II*)

  Un segundo enfoque para la etapa de contracción consiste en calcular en cada iteración el vector normal n_i^t, obteniendo la suma ponderada de las normales incidentes en v_i [19], es decir, las normales de las caras que forman parte de ese vértice. En esta suma ponderada, el peso de cada normal es proporcional al ángulo que forma la cara con v_i (Figura 4.3).

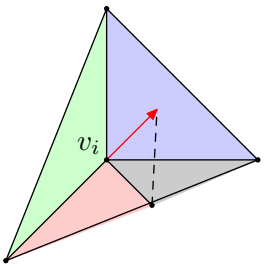


Figura 4.3: Vector normal asociado a v_i. El peso de las normales incidentes a v_i es proporcional al ángulo que forma la cara con este vértice.

El desplazamiento de v_i está dado por :

$$v_i^{t+1} = w_1\ v_i^t + w_2\ d_i^t \tag{4.4}$$
$$= w_1\ v_i^t + w_2\ n_i^t \tag{4.5}$$
$$= w_1\ v_i^t + w_2 \left(\frac{\sum_{t_{(i,j,k)} \in F} \alpha_{i,j,k}^t\ n^t(t_{(i,j,k)})}{\sum_{t_{(i,j,k)} \in F} \alpha_{i,j,k}^t} \right) \tag{4.6}$$

donde

- n_i^t es el vector normal de v_i en la iteración t
- $t_{(i,j,k)} \in F$ son los triángulos vecinos de v_i
- $n^t(t_{(i,j,k)})$ es el vector normal del triángulo $t_{(i,j,k)}$ en la iteración t
- $\alpha_{i,j,k}^t$ es el ángulo que forma $t_{(i,j,k)}$ con v_i en la iteración t
- $w_1 = 1.0$
- $w_2 = 0.001$.

A medida que la etapa de contracción avanza, los triángulos y los ángulos de la malla disminuyen su tamaño. El menor tamaño de los triángulos y los ángulos puede generar un cambio en la dirección del vector normal en cada iteración de este proceso. Es por esta razón que se elaboró una tercera alternativa para la etapa de contracción de vértices.

- Contracción Geométrica III (*Geometry Contraction III*)

  Al recalcular el vector normal durante cada iteración (como se sugiere en el método de anterior) es posible que se altere la forma de la malla original. Para intentar mantener la morfología de la malla lo más intacta posible, se propuso este método de contracción:

$$v_i^{t+1} = w_1\ v_i^t + w_2\ d_i^t \tag{4.7}$$
$$= w_1\ v_i^t + w_2\ n_i^0 \tag{4.8}$$
$$= w_1\ v_i^t + w_2 \left(\frac{\sum_{t_{(i,j,k)} \in F} \alpha_{i,j,k}^0\ n^0(t_{(i,j,k)})}{\sum_{t_{(i,j,k)} \in F} \alpha_{i,j,k}^0} \right) \tag{4.9}$$

donde

- n_i^0: es el vector normal inicial de v_i

- $n^0(t_{(i,j,k)})$: es el vector normal del triángulo $t_{(i,j,k)}$ en la malla original
- $\alpha^0_{i,j,k}$ es el ángulo que forma $t_{(i,j,k)}$ con v_i en la malla original
- $w_1 = 1.0$
- $w_2 = 0.001$

Como el proceso de contracción geométrico es iterativo, es importante definir criterios de detención generales basados en características propias de las mallas (por ejemplo, un porcentaje del volumen y/o área de la malla inicial) o un desplazamiento mínimo aceptable.

4.1.2. Colapso de la Malla Contraída (*connectivity surgery*)

Para convertir la malla contraída en un grafo 1D se realiza el colapso de la malla. Este proceso aplica una serie de colapsos de arcos para remover todas las caras[2] de la malla contraída.

El *input* de esta etapa es una malla contraída $\tilde{G} = (\tilde{V}, \tilde{F}, \tilde{E})$, con vértices $\tilde{V}$, caras $\tilde{F}$ y arcos $\tilde{E}$, siendo $\tilde{V} = [\tilde{v_1}, \tilde{v_2}, ..., \tilde{v_n}]$ las posiciones de los vértices. Esta malla tiene aproximadamente volumen cero y visualmente se asemeja a un *skeleton* 1D. Sin embargo, su conectividad aún es la de la malla original (superficie 3D).

El principal requerimiento para esta etapa es retener la geometría de malla colapsada durante el proceso, es decir, mantener suficientes nodos del *skeleton* para lograr una buena correspondencia entre éste y la malla original, considerando características como componentes conexas, ciclos y túneles. Para ello, se introduce una condición de colapso que asegura mantener el mismo número de *loops* y túneles de la malla original [2]: si existen tres vértices adyacentes (i, j, k), pero el triángulo $t_{(i,j,k)}$ no existe, entonces se prohíbe el colapso ($i \to j$).

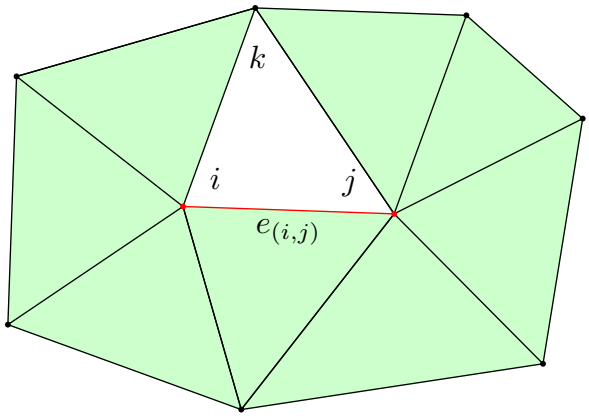


Figura 4.4: Condición de colapso de $e_{(i,j)}$. Cuando los vértices (i, j, k) son adyacentes pero el triángulo $t_{(i,j,k)}$ no existe, se prohibe el colapso ($i \to j$) [2].

El colapso de la malla es un algoritmo *greedy* [6] que en cada iteración colapsa el arco que tiene el costo mínimo entre todos los arcos de la malla. Para determinar el costo de cada arco se dispone de la función

[2]Las caras de la malla contraída tienen un área cuyo valor es aproximadamente cero.

$$F(i,j) = \text{mín} \begin{cases} F(i,j) = \text{costo de colapsar el vértice } i \text{ sobre } j \\ \\ F(j,i) = \text{costo de colapsar el vértice } j \text{ sobre } i \end{cases} \tag{4.10}$$

$F(i,j)$ se compone de dos términos: el costo de la forma (*shape cost*) y el costo de muestreo (*sampling cost*).

1. Costo de forma (*Shape cost*)

   El costo de forma está basado en el método de simplificaciones métricas de error cuadrático (*quadratic error metrics*, QEM) [12] que cuantifica la distorsión causada por el colapso de un arco (midiendo la distancia entre un vértice y sus caras asociadas) y por ende preserva la geometría de la malla.

   El error de un vértice i es la suma de todas las distancias al cuadrado a sus arcos adyacentes en su forma matricial, es decir,

   $$F_i(p) = p^T \sum_{e_{(i,k)} \in \tilde{E}} (K_{i,k}^T K_{i,k})\ p = p^T Q_i p \tag{4.11}$$

   donde p es la posición del vértice resultante del colapso de $e_{(i,j)}$ en su representación homogénea (es decir, si el colapso es $(i \to j)$, entonces p es $\tilde{v}_j$ [2]).

   Se define una matriz $K_{i,j}$ para cada arco $e_{(i,j)}$ en la malla contraída, tal que el valor $p^T(K_{i,j}^T K_{i,j})p$ es la distancia al cuadrado desde el punto p a la recta definida por el arco $e_{(i,j)}$ (los detalles de la demostración se encuentran en el Apéndice D.1).

   $$K_{i,j} = \begin{pmatrix} 0 & -a_z & a_y & -b_x \\ a_z & 0 & -a_x & -b_y \\ -a_y & a_x & 0 & -b_z \end{pmatrix} \tag{4.12}$$

   donde

   $$a = \frac{e_{(i,j)}}{\left\| e_{(i,j)} \right\|} \tag{4.13}$$

   $$b = a \times \tilde{v}_i \tag{4.14}$$

   Para mantener la geometría de malla contraída tan inalterada como sea posible, se define la función de costo $F_a(i,j)$ para determinar el próximo arco a colapsar.

   $$F_a(i,j) = F_i(\tilde{v}_j) + F_j(\tilde{v}_j) \tag{4.15}$$

   $F_a(i,j)$ es el costo de colapsar el vértice i sobre el vértice j. Al colapsar $e_{(i,j)}$, los arcos que previamente incidían sobre el vértice i ahora inciden sobre el vértice j. Además, se remueven las caras adyacentes a este arco.

Para terminar, se actualiza la matriz del error del vértice j como:

$$Q_j = Q_i + Q_j \tag{4.16}$$

es decir, como se colapsó el arco $e_{(i,j)}$, el error asociado al vértice i ahora se agrega al error asociado al vértice j.

2. Costo de muestreo (*Sampling cost*)

   Este costo penaliza a los arcos colapsados que generan arcos largos midiendo la suma de la distancia desde $\tilde{v}_i$ hasta $\tilde{v}_j$ en una arista $e_{(i,j)}$. El nuevo término de la función de costo es el siguiente:

$$F_b(i,j) = \left\|\tilde{v}_i - \tilde{v}_j\right\| \sum_{e_{(i,k)} \in \tilde{E}} \left\|\tilde{v}_i - \tilde{v}_k\right\| \tag{4.17}$$

   donde $\tilde{E}$ es el conjunto actual de arcos de la malla.

Finalmente, la función de costo total del colapso de un arco $e_{(i,j)}$ desde $\tilde{v}_i$ hasta $\tilde{v}_j$ es la suma ponderada del costo de forma y el costo de muestreo:

$$F(i,j) = w_a F_a(i,j) + w_b F_b(i,j) \tag{4.18}$$

donde $w_a = 1.0$ y $w_b = 0.1$ para los ejemplos que aparecen en [2].

Al colapsar sucesivamente los arcos, pueden generarse arcos nuevos (Figura 4.5), a la vez que las caras de la malla son eliminadas hasta que ésta se convierte en una estructura 1D compuesta sólo por segmentos conectados entre sí y en que cada nodo retiene el historial de arcos y caras contraídas de la estructura original.

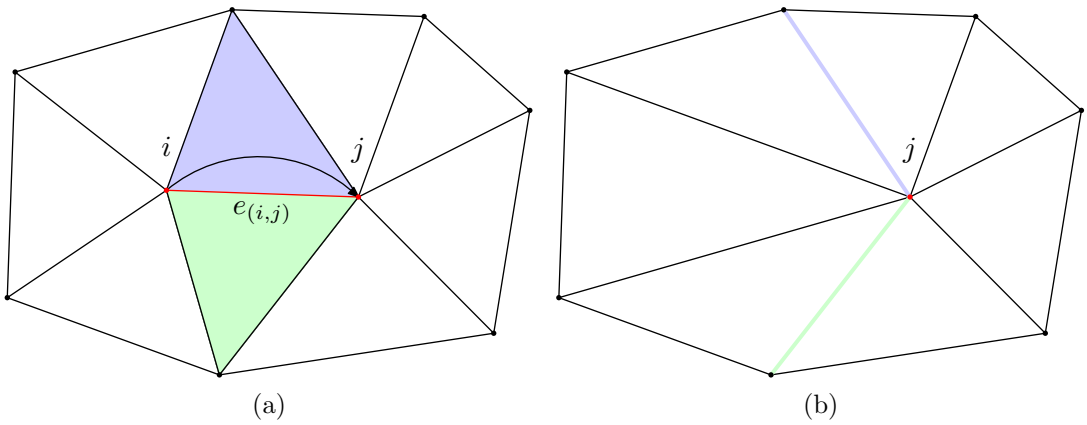


Figura 4.5: Región de la malla antes (a) y después (b) del colapso del arco $e_{(i,j)}$ de menor costo. Los triángulos incidentes a $e_{(i,j)}$ (en color) se eliminan, pero se retienen los arcos opuestos a i.

El método de colapso se detiene cuando ya no existen triángulos en la malla , es decir, cuando ya no hay arcos colapsables y por lo tanto se disponga de un *skeleton* 1D.

4.1.3. Evaluación y Corrección del *skeleton* (*embedding refinement*)

El proceso iterativo de colapso de la malla puede producir un *skeleton* que se encuentre fuera del objeto original, especialmente en las componentes que tienen grandes diferencias de curvatura. Además, zonas de baja densidad pueden generar nodos no centrados en el *skeleton* final. Para mejorar estos errores, se realiza el proceso de evaluación y corrección del *skeleton*.

Se define el *skeleton* $S = (N,\ U)$, donde $N = [n_1,\ n_2,\ ...,\ n_m]$ son las posiciones de los m nodos del *skeleton* y U son los segmentos formados por pares de nodos. El objetivo de esta etapa es mover cada nodo n_i al centro aproximado de su región local Π_{n_i}. Esta región corresponde al conjunto de los vértices de la malla original $v \in V$ que durante la etapa anterior colapsaron en n_i (Ecuación (4.19)).

$$\forall\ v \in V,\quad tq\quad v \rightarrow\ n_i \in N\ \Rightarrow\ v \in \Pi_{n_i} \tag{4.19}$$

El *boundary loop* [2] $bl(n_i, n_k)$ corresponde al subconjunto de los vértices $v \in \Pi_{n_i}$ que cumplen la siguiente propiedad: si los vértices $v_m, v_l \in V$ de la malla inicial colapsaron en los nodos $n_i, n_k \in N$, respectivamente y se encuentran conectados a través de un arco $e_{(m,l)} \in E$, entonces $v_m \in bl(n_i, n_k)$ (Ecuación (4.20)).

$$\forall\ v_m, v_l \in V\quad tq\quad v_m \in \Pi_{n_i}, v_l \in \Pi_{n_k}\quad \wedge\ e_{(m,l)} \in E\ \Rightarrow\ v_m \in bl(n_i, n_k) \tag{4.20}$$

Un ejemplo se observa en la Figura 4.6, donde:

- vértices en rojo: representan los vértices que convergen a n_i (región Π_{n_i}).
- vértices en verde: representan los vértices que convergen a n_{i-1} (región $\Pi_{n_{-1}}$).
- vértices en azul: representan los vértices que convergen a n_{i+1} (región $\Pi_{n_{i+1}}$).
- vértices rojos con borde negro unidos a los vértices en verde: representan el *boundary loop* $bl(n_i, n_{i-1})$.
- vértices rojos con borde negro unidos a los vértices en azul: representan el *boundary loop* $bl(n_i, n_{i+1})$.

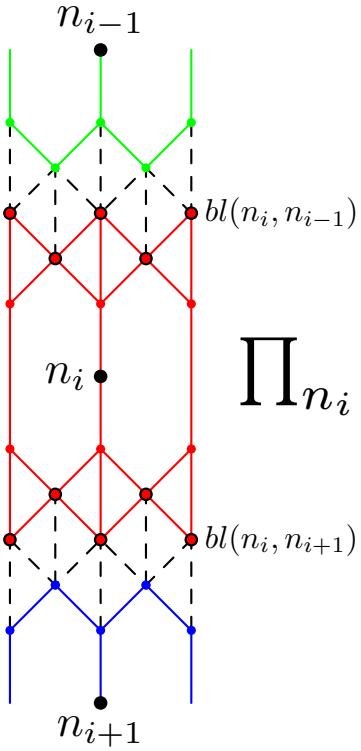


Figura 4.6: Nodo n_i y su región local Π_{n_i}. Los vértices en rojo representan los vértices que colapsaron en n_i (Π_{n_i}). Los vértices en rojo con borde negro unidos a los vértices en verde representan el boundary loop $bl(n_i, n_{i-1})$. Los vértices en rojo con borde negro unidos a los vértices en azul representan el boundary loop $bl(n_i, n_{i+1})$.

Para centrar los nodos del *skeleton*, se introduce un desplazamiento $d(bl(n_i, n_{i-1}))$:

$$d(bl(n_i, n_k)) = \frac{\sum_{v_j \in bl(n_i, n_k)} l_j(\tilde{v}_j - v_j)}{\sum_{v_j \in bl(n_i, n_k)} l_j} \tag{4.21}$$

donde $\tilde{v}_j$ y v_j corresponden a la posición final e inicial de los vértices que pertenecen al *boundary loop* $bl(n_i, n_k)$, respectivamente.

l_j corresponde a una ponderación al desplazamiento de los vértices $v_j \in bl(n_i, n_k)$. Para definir esta ponderación existen dos alternativas:

- En [2] se propone definir l_j como el largo total de dos arcos adyacentes al vértice $v_j \in bl(n_i, n_k)$ (Figura 4.7).

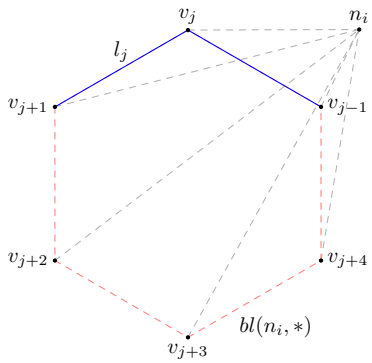


Figura 4.7: Definición de l_j dada en [2]. Se define l_j como el largo total de dos arcos adyacentes al vértice $v_j \in bl(n_i, n_k)$.

Esta definición esta hecha bajo el supuesto que la región Π_{n_i} tiene una forma similar a un cilíndro. Dado que la forma de Π_i depende fuertemente del método de contracción utilizado en la etapa *geometry contraction*, no siempre es cierta la aseveración anterior. Es por esto que se elaboró una segunda alternativa en la definición de l_j.

- La alternativa propuesta consiste en definir l_j como la distancia euclidiana entre la posición inicial del vértice v_j y la posición del nodo n_i (Figura 4.8 (b)).

$$l_j = |v_j - n_i| \tag{4.22}$$

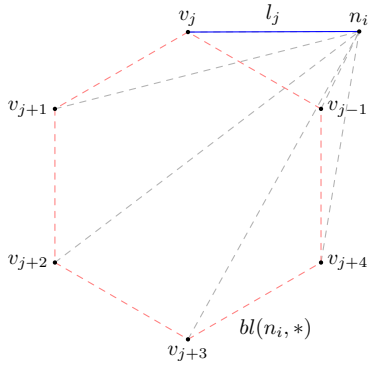


Figura 4.8: Definición de l_j propuesta en este trabajo de memoria. Se define l_j como la distancia entre v_j y n_i.

Luego, con este desplazamiento $d(bl(n_i, n_k))$ se procede a centrar los nodos del *skeleton* según la clasificación dada en [2] (Figura 4.9) :

- *terminal nodes*: nodos terminales. Corresponden a los nodos que sólo tienen un segmento incidente.
- *non junction nodes*: nodos de unión. Corresponden a los nodos que tienen dos segmentos incidentes.
- *junction nodes*: nodos de bifurcación. Corresponden a los nodos que tienen más de dos segmentos incidentes.

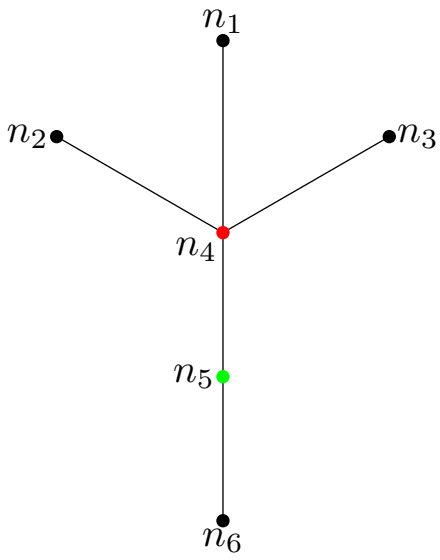


Figura 4.9: Clasificación de los nodos del *skeleton* definida en [2]. n_1, n_2, n_3 y n_6 son *terminal nodes*, n_4 es *junction node* y n_5 es *non junction node*.

Cada *non junction node* es relocalizado según la ecuación :

$$n_i = n_i - \frac{d_1 + d_2}{2} \tag{4.23}$$

donde d_1 y d_2 son los desplazamientos d mencionados anteriormente para los dos nodos vecinos de n_i.

Dado que los *junction nodes* presentan varios nodos vecinos, se extiende la ecuación anterior :

$$n_i = n_i - \frac{\sum_{u_{(i,j)} \in U} d(bl(n_i, n_j))}{n} \tag{4.24}$$

donde $u_{(i,j)} \in U$ son todos los segmentos del *skeleton* que inciden en n_i y n es número de nodos unidos a n_i mediante un segmento $u_{(i,j)}$.

Finalmente, los *terminal nodes* se centran de la siguiente forma:

$$n_i = n_i - \frac{\sum_{v_j \in \Pi_{n_i}} (\tilde{v}_j - v_j)}{n} \tag{4.25}$$

donde n es número de nodos de Π_{n_i}.

4.2. Diseño de la Aplicación

Para diseñar la aplicación se utilizaron diagramas de clases. Un diagrama de clases en el Lenguaje Unificado de Modelado (*Unified Modeling Language UML*[3]) describe la estructura

[3]http://www.uml.org/

de un sistema mostrando sus clases, atributos y las relaciones entre ellos.

Las relaciones entre las clases definidas en *UML* que se utilizaron en el diseño de la aplicación son las siguientes:

- Composición: una clase A está compuesta de una clase B cuando la clase A está compuesta de un atributo que corresponde a un objeto de la clase B. Un ejemplo se observa en la Figura 4.10.

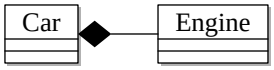


Figura 4.10: Relación de composición en un diagrama de clase. La clase *Car* está compuesta de la clase *Engine*, es decir, atributo que corresponde a un objeto de la clase *Engine*.

- Herencia: una clase A hereda de una clase B cuando la clase A corresponde a una especialización (una instancia más específica) de la clase B. Un ejemplo se observa en la Figura 4.11.

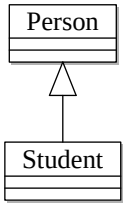


Figura 4.11: Relación de herencia en un diagrama de clase. La clase *Student* hereda de la clase *Person* porque la clase *Student* corresponde a una especialización de la clase *Person*.

- Dependencia: una clase A depende de una clase B cuando A utiliza y/o depende de la clase B en algún momento. Un ejemplo se observa en la Figura 4.12 .

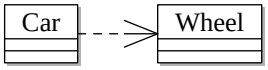


Figura 4.12: Relación de dependencia en un diagrama de clase. La clase *Car* depende de la clase *Wheel* porque depende de ella en algún momento.

4.2.1. Estructuras de Datos

Las estructuras de datos utilizadas en la implementación son: *Mesh, Triangle, Vertex, Edge, Skeleton, Node* y *Segment* (Figura 4.13)[4].

[4]El detalle de las estructuras de datos asociadas a la malla y al *skeleton* se encuentra en el Apéndice E, Figuras 9.60 y 9.61, respectivamente.

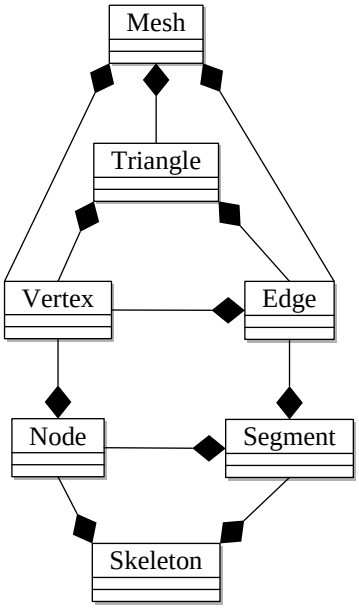


Figura 4.13: Diagrama de clases de las estructuras utilizadas.

A continuación, se presenta una descripción de las estructuras de datos para representar la malla y el *skeleton*:

- *Mesh*: está formada por tres listas: una de vértices, una de arcos y una de triángulos que la componen.
- *Vertex*: mantiene una referencia a una estructura llamada *OneRing* (Apéndice E, Figura 9.65). *OneRing* mantiene dos listas que corresponden a los arcos y a los triángulos incidentes en cada vértice. Además, guarda una lista donde se almacenan sus vértices colapsados.
- Edge: mantiene referencias a los dos vértices que lo componen, a sus dos triángulos incidentes y a sus dos vértices opuestos (Figura 4.14).

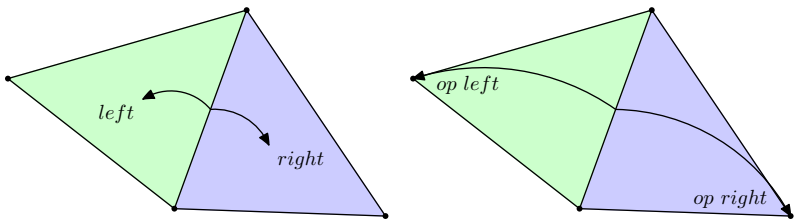
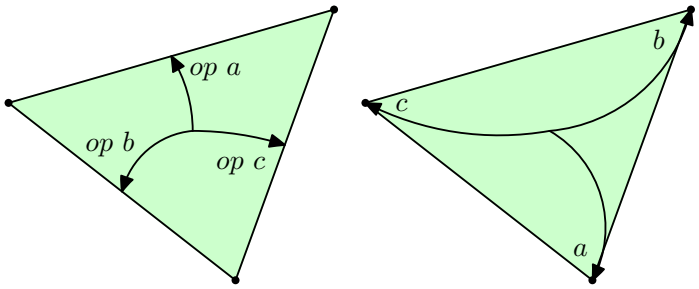


Figura 4.14: Referencias que mantiene cada arco de la malla. (a) y (b) muestran los triángulos incidentes y los vértices opuestos, respectivamente.

- *Triangle*: mantiene referencias a los tres vértices y arcos que lo componen (Figura 4.15).



(a) (b)

Figura 4.15: Referencias que mantiene cada triángulo de la malla. (a) y (b) muestran los arcos y los vértices que lo componen, respectivamente.

- *Skeleton*: esta formado por dos listas: una de nodos y otra de segmentos que lo componen.
- *Node*: esta formado por una lista de los segmentos incidentes en cada nodo. Además, mantiene una referencia al vértice que lo originó.
- *Segment*: mantiene referencias a los dos nodos que lo componen y al arco que lo originó.

4.2.2. Aplicación

Con respecto al diseño de la aplicación, el diagrama de clases elaborado se muestra en la Figura 4.17. La clase principal es *SkeletonExtractionFromMeshContraction*. Para proporcionar una interfaz unificada del algoritmo se utilizó el patrón estructural *Facade* [11] (Figura 4.16). Los roles que cumplen las clases del paquete principal *project* (Figura 4.17) de acuerdo a la definición de [11] son los siguientes:

- *SkeletonExtractionFromMeshContraction*: *Compiler*
- *ROI3DGROUPOBJETDEFINE* (clase del proyecto implementado en *SCIAN-Soft*): *Scanner, Parser*, etc.

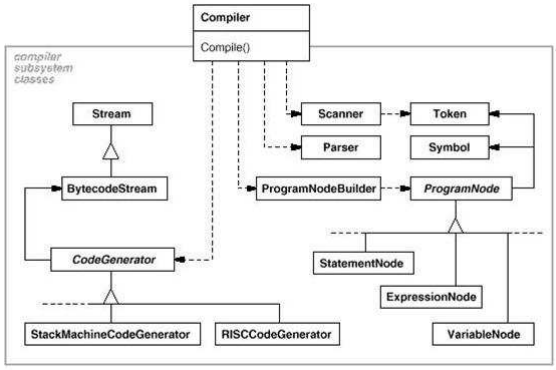


Figura 4.16: Patrón de diseño *Facade* [11].

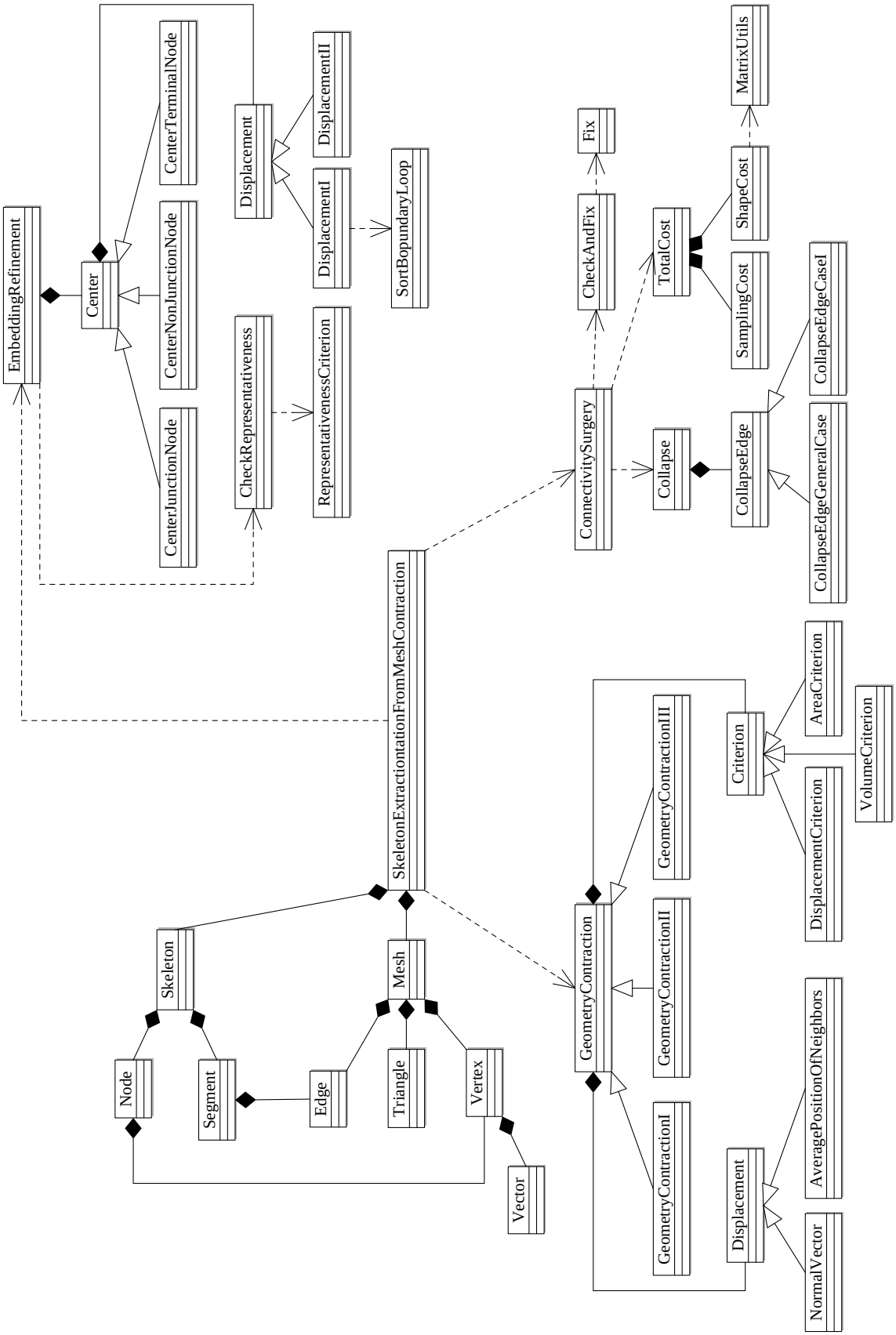


Figura 4.17: Diagrama de clases del algoritmo *skeleton extraction by mesh contraction.*

En la implementación de la etapa *geometry contraction*, para desacoplar la interacción entre el tipo de desplazamiento a utilizar y los criterios de detención se utilizó el patrón estructural *Bridge* [11] (Figura 4.18). Los roles que cumplen las clases del paquete *geometry contraction* (Figura 4.19) de acuerdo a la definición [11] son los siguientes:

- *GeometryContraction*: *Abstraction*
- *GeometryContractionI*: *RefinedAbstraction*
- *GeometryContractionII*: *RefinedAbstraction*
- *GeometryContractionIII*: *RefinedAbstraction*
- *Criterion*: *Implementor*
- *AreaCriterion*: *ConcreteImplementor*
- *DisplacementCriterion*: *ConcreteImplementor*
- *VolumeCriterion*: *ConcreteImplementor*
- *Displacement*: *Implementor*
- *NormalVector*: *ConcreteImplementor*
- *AveragePositionOfNeighbors*: *ConcreteImplementor*

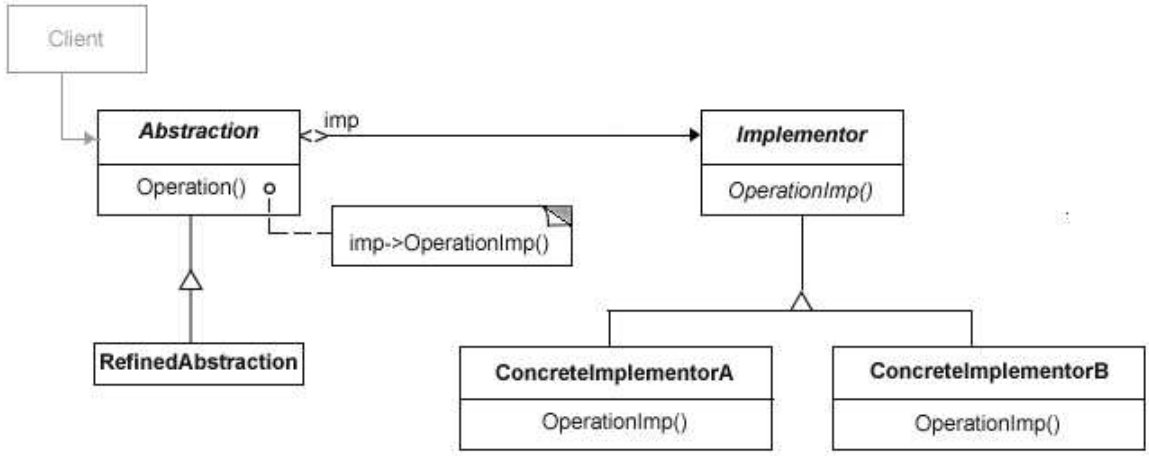


Figura 4.18: Patrón de diseño *Bridge* [11].

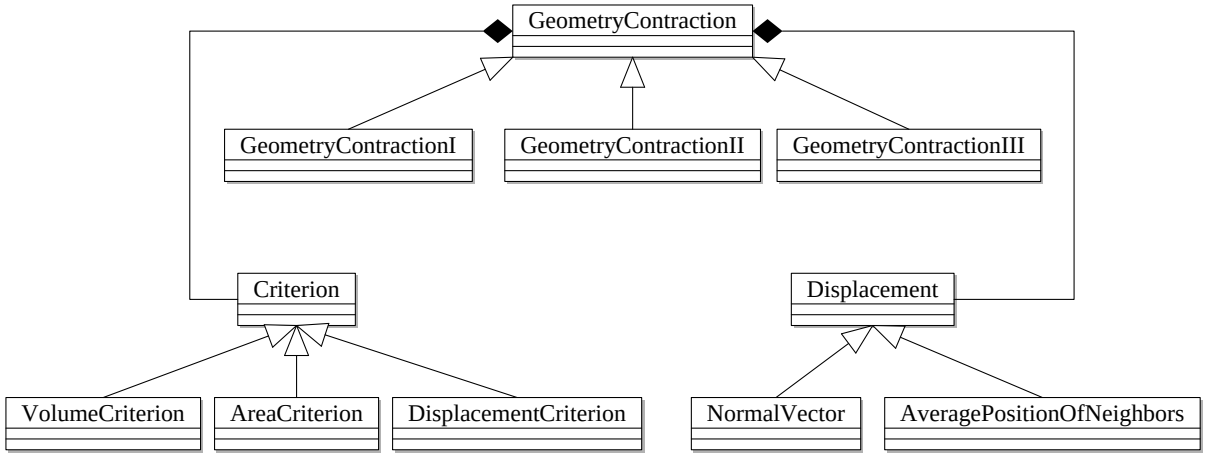


Figura 4.19: Diagrama de clases de la etapa de contracción de la malla (*geometry contraction*).

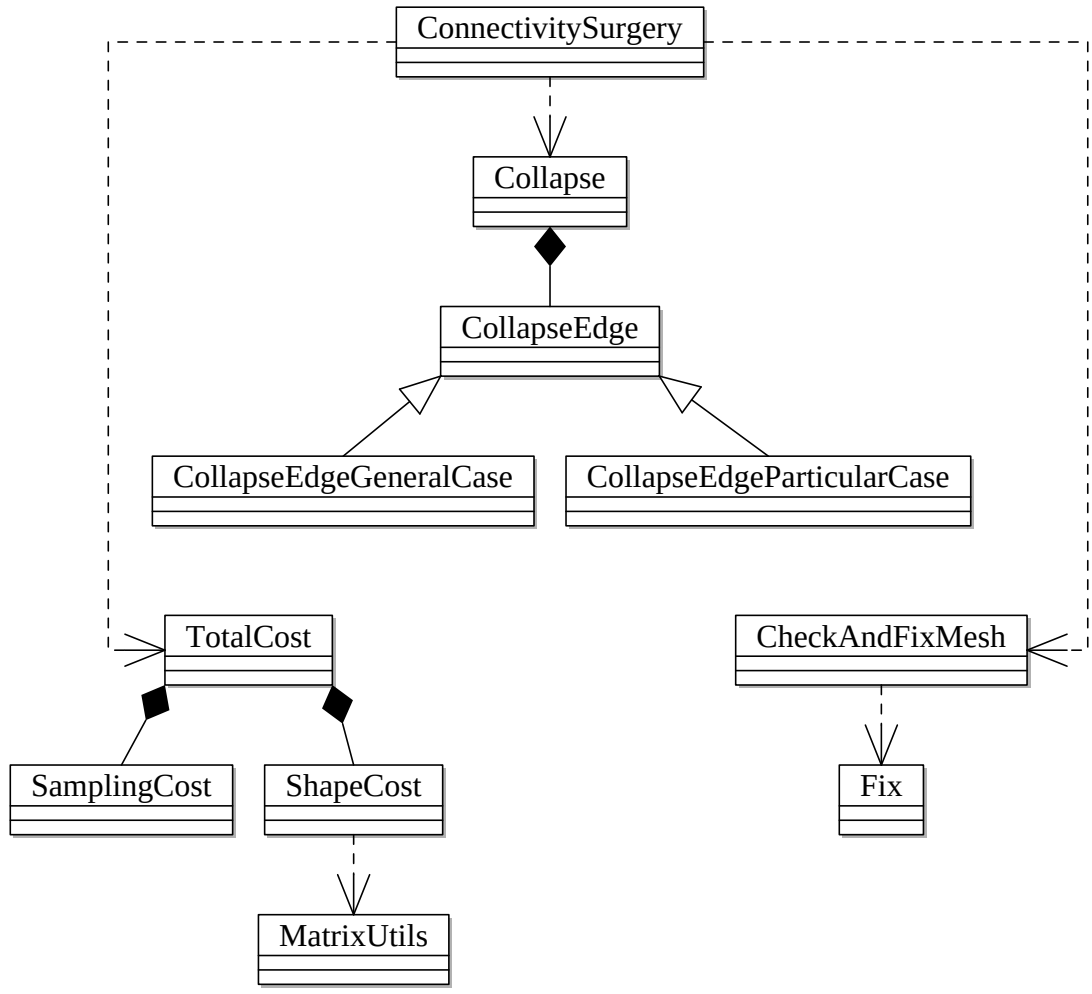


Figura 4.20: Diagrama de clases de la etapa de colapso de aristas (*connectivity surgery*).

En la implementación de etapa de centrado del *skeleton* (*embedding refinement*), para desacoplar la interacción entre el tipo de desplazamiento a utilizar y el tipo de nodo a centrar se utilizó el patrón estructural *Bridge* [11](Figura 4.21). Los roles que cumplen las clases del paquete *embedding refinement* de acuerdo a la definición [11] son los siguientes:

- *Center*: *Abstraction*
- *CenterJunctionNode*: *RefinedAbstraction*
- *CenterNonJunctionNode*: *RefinedAbstraction*
- *CenterTerminalNode*: *RefinedAbstraction*
- *Displacement*: *Implementor*
- *DisplacementI*: *ConcreteImplementor*
- *DisplacementII*: *ConcreteImplementor*

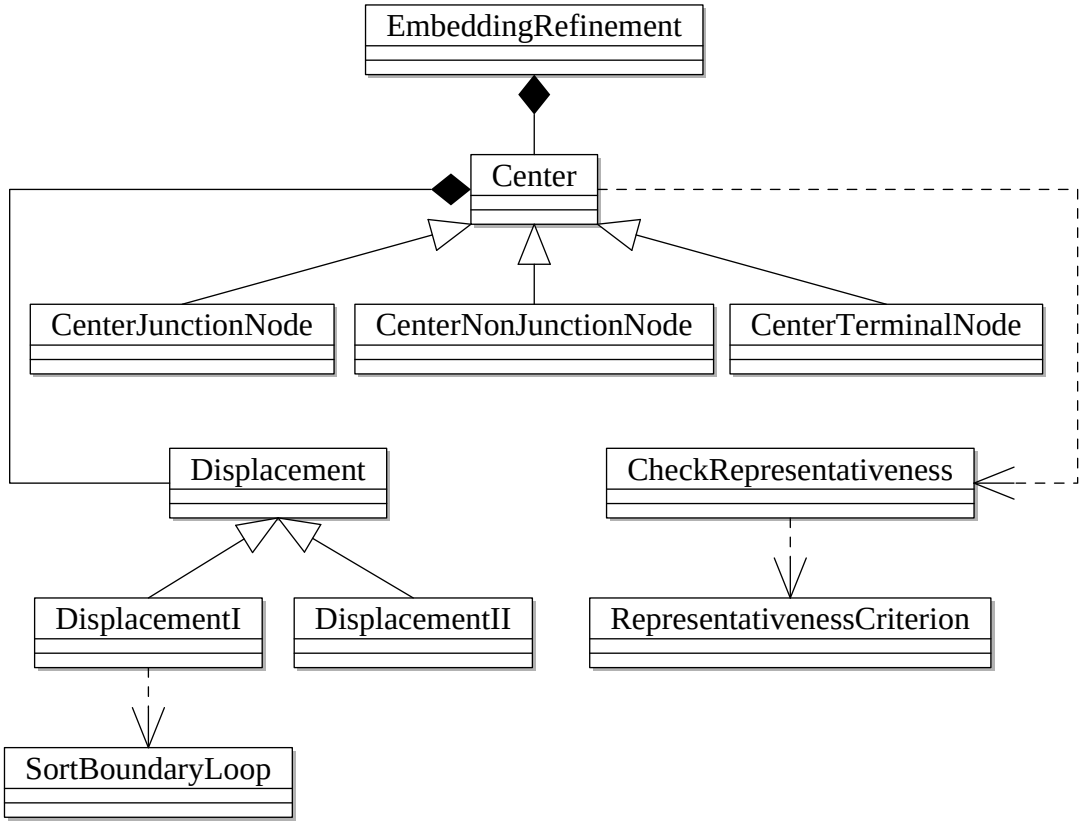


Figura 4.21: Diagrama de clases de la etapa de centrado del *skeleton* (*embedding refinement*).

El detalle de los diagramas de clases de la etapas *geometry contraction*, *connectivity surgery* y *embedding refinement* se encuentran en el Apéndice E, Figuras 9.62, 9.63, 9.64.

Capítulo 5

Implementación

En este capítulo se muestran los detalles de implementación de la solución propuesta, entre los que se encuentran: una descripción de las herramientas utilizadas y los métodos diseñados para cada una de las etapas de algoritmo de esqueletonización.

5.1. Software Utilizado

Durante la implementación del algoritmo se utilizaron las siguientes herramientas de software:

- El lenguaje de programación *Java*[1].
- *JUnit* 4.1[2] para ejecutar la batería de pruebas de validación del *software*.
- *Understand* 2.6[3] para obtener las métricas del diseño del *software*.
- *Netbeans* 7.1[4] como entorno de desarrollo de la aplicación.
- *Geomview* 1.9.4[5] para la visualización de mallas y *skeletons*.
- *Blender* 2.62[6] para la visualización de mallas y *skeletons*.
- *SCIAN-Soft* es el software para tratamiento, análisis y visualización de imágenes científicas desarrollado en el *SCIAN-lab*[7].
- *IDL* 7.0 (Interactive Data Language)[8], es el lenguaje en el cual está implementado *SCIAN-Soft*, y se utilizó para la visualización final de *skeletons* resultantes.

[1] http://www.java.com
[2] http://www.junit.org/
[3] http://www.scitools.com/
[4] http://netbeans.org/
[5] http://www.geomview.org/
[6] http://www.blender.org/
[7] Para más detalles sobre la descripción de *SCIAN-Soft*, ver el Capítulo 5.3.
[8] http://www.exelisvis.com/language/en-us/productsservices/idl.aspx

5.2. Implementación del Algoritmo

5.2.1. Etapa *Geometry Contraction*

De acuerdo a la descripción dada en la sección 2.2.1, se implementaron tres enfoques para la etapa de contracción de vértices:

Contracción Geométrica I (*Geometry Contraction I*)

Este método de contracción desplaza los vértices $v_i \in V$ hacia el promedio de la posición de vecinos de v_i.

Algorithm 1 Displacement: Average Position of Neighbors

```
1:  procedure GetAverageOfPositionNeighbors(vertex)
2:      position := 0
3:      for all edge ∈ vertex.oneRing do          ▷ oneRing: neighbors of vertex [13]
4:          neighbor ← edge.getNeighbor(vertex)
5:          if vertex.index ≡ edge.a.index then
6:              position + = neighbor
7:          end if
8:      end for
9:      if vertex.oneRing ≡ empty then  return position
10:     else return position / vertex.oneRing.size
11:     end if
12: end procedure
```

La principal ventaja de este enfoque radica en el hecho que al contraer un vértice v_i^t es muy difícil que v_i^{t+1} atraviese la sección transversal de la zona de la malla que lo contiene (Figura 5.1). Este hecho empírico se basa en que los vecinos de v_i se encuentan lo suficientemente cerca de este vértice para que este error no ocura. Esta es la razón por la cual finalmente se escogió esta alternativa como método de contracción del algoritmo *skeleton extraction by mesh contraction*[9] .

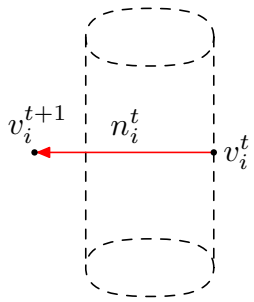


Figura 5.1: Defecto del método de contracción basado en el vector normal de cada vértice. Al contraer v_i^t en dirección de su vector normal n_i^t, v_i^{t+1} se posiciona fuera de la malla.

[9]Como se menciona más adelante, los métodos *Geometry Contraction II* y *Geometry Contraction III* no cumplen con esta caraterística.

Sin embargo, este enfoque también presenta una caraterística no deseable: las zonas de la malla que presentan mayor curvatura tienden a ubicarse fuera de la malla original a medida que ésta se contrae (Figura 5.2). Si la malla presenta muchas zonas curvas, pueden darse casos más graves donde regiones de la malla contraída se encuentren fuera de la malla original (Apéndice A, Figuras 9.3 (a), 9.11 (a) y 9.7 (a)).

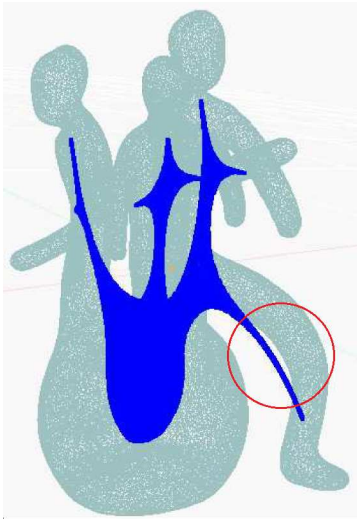


Figura 5.2: Malla de superficie contraída con regiones ubicadas fuera de la malla original. La malla de superficie fue contraída hasta alcanzar el 15 % del área inicial utilizando el método *Geometry Contraction I*.

Esta característica puede considerarse un error desde el punto de vista del algoritmo, pues en un principio es deseable que la malla contraída se encuentre completamente dentro de la malla original y además se espera que este centrada con respecto a ésta. Sin embargo, como se menciona en el Capítulo 5.2.3, este defecto no impide que se obtengan *skeletons* correctos, pues la etapa de centrado (*embedding refinement*) corrige estos errores.

Contracción Geométrica II (*Geometry Contraction II*)

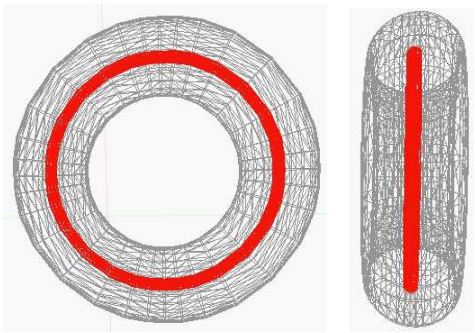
Este método se implementó con el objetivo de poder contraer los vértices $v_i \in V$ de la malla en dirección de su vector normal n_i. Como se mencionó en el Capítulo 4.2.1, el cálculo de este vector se obtuvo usando las normales de los triángulos incidentes a v_i, donde cada normal tiene una ponderación igual al ángulo que forman con este vértice (Algoritmo 2).

Algorithm 2 Displacement: Normal Vector

```
1: procedure GetNormalVector(vertex)
2:     position := 0
3:     totalAngle := 0
4:     for all triangle ∈ vertex.oneRing do
5:         normal ← computeNormal(triangle)
6:         angle ← triangle.getAngle(vertex)
7:         totalAngle += angle
8:         position += normal · angle
9:     end for
10:    position = position / totalAngle
11:    return position / ||position||
12: end procedure

13: function computeNormal(triangle)
14:    vector_1 ← (triangle.c.position − triangle.a.position)
15:    vector_2 ← (triangle.b.position − triangle.a.position)
16:    return (vector_1 × vector_2) / ||vector_1 × vector_2||
17: end function
```

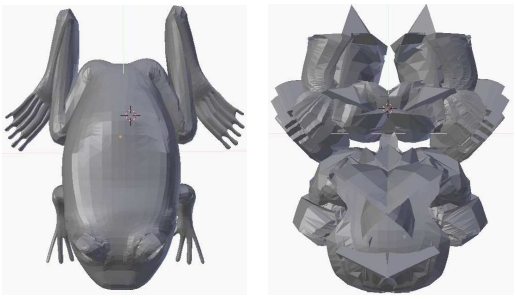
El resultado del proceso de contracción recalculando el vector normal en cada iteración presenta buenos resultados cuando las mallas de entrada son regulares (Figura 5.3).



(a) (b)

Figura 5.3: Malla *donuts* contraída utilizando el método *Geometry Contraction II*. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial.

Sin embargo, este método no está exento de problemas. Al contraer v_i en dirección de su vector normal, es frecuente que su posición final atraviese la sección transversal donde se ubica (Figura 5.1). Como consecuencia, el volumen y el área de la malla aumente progresivamente a medida que avanza el proceso de contracción (Figura 5.4).



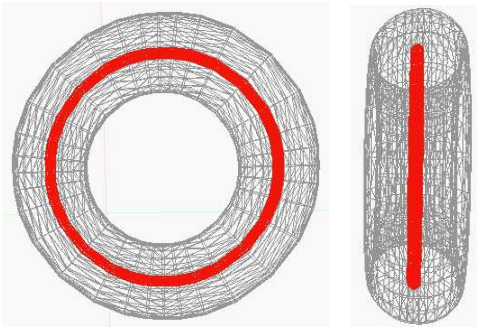
(a) (b)

Figura 5.4: Malla de superficie *frog* inicial (a) y contraída utilizando el método *Geometry Contraction II* después de 50 iteraciones (b).

Contracción Geométrica III (*Geometry Contraction III*)

El método anterior (*Geometry Contraction II*), contrae todos lo vértices de la malla calculando su vector normal en cada iteración y los desplaza en esta dirección. A medida que avance este proceso, la posición de los triángulos de la malla cambiará, provocando que las normales incidentes a v_i varien. Es por esto que se elaboró un método donde los vértices se desplazan en dirección de su vector normal inicial (Capítulo 4.2.1).

Al igual que en el método anterior, los resultados de esta estrategia de contracción son correctos cuando las mallas de entrada son regulares (Figura 5.5), pero es frecuente que los vértice v_i^t atraviesen su respectiva sección transversal al contraerlos en dirección de su vector normal.



(a) (b)

Figura 5.5: Malla *donuts* contraída utilizando el método *Geometry Contraction III*. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial.

Criterios de Detención

Como se mencionó en el Capítulo 4.2.1, dado que el proceso de contracción geométrico es iterativo, es importante definir criterios de detención generalizables a todas las mallas. Bajo esta premisa, se implementaron dos criterios de detención:

1. Criterio de detención basado en características de la malla

   Dos características propias de las mallas a considerar para el criterio de detención de la etapa de contracción son el volumen y el área; por simplicidad, se optó por la última. Para definir el umbral correcto, se estudió el decaimiento del área a partir de mallas de superficie de distinta complejidad y tamaño[10]. Como se observa en la Figura 5.6, esta función tiene un decaimiento expotencial, alcanzando rápidamente una pendiente cercana a cero.

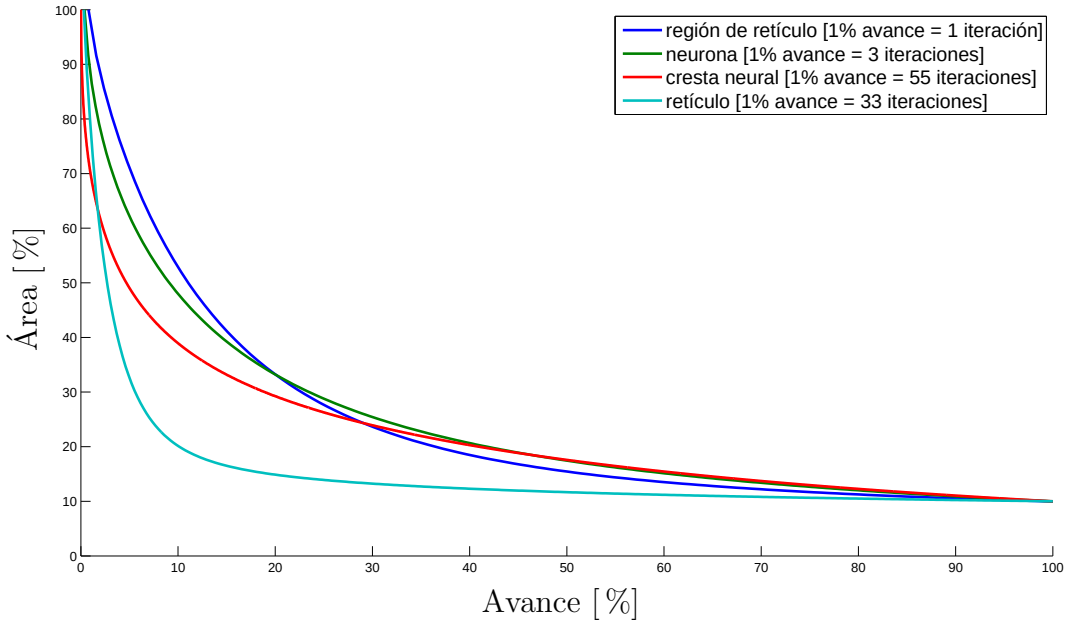
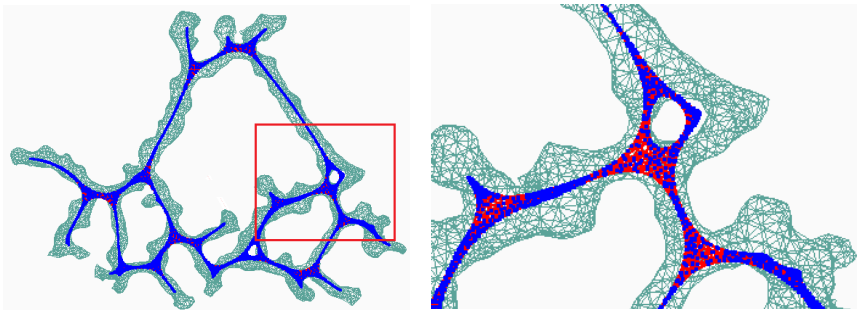


Figura 5.6: Decaimiento porcentual del área de las mallas biológicas durante la etapa de contracción (*geometry contraction*). Las mallas fueron contraídas utilizando el método escogido (*GeometryContractionI*).

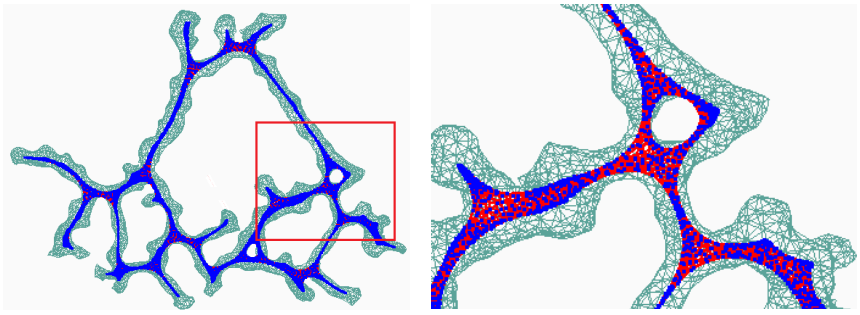
   Dado este criterio, el objetivo de la etapa de contracción es obtener mallas contraídas con el menor área posible, siempre que se ubiquen dentro de la malla original. Para verificar que las mallas contraídas se encuentren completamente dentro de la malla original, se generaron las mallas contraídas de las mallas biológicas hasta alcanzar el 10 %, 15 % y 20 % del área de superficie inicial (Figuras 5.7, 5.8, 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17 y 5.18).

[10]Se trabajaron con mallas de interés biológico y con mallas generadas artificialmente. Las estadísticas y los resultados obtenidos con las mallas de fantasía se encuentran en el Apéndice B.3.



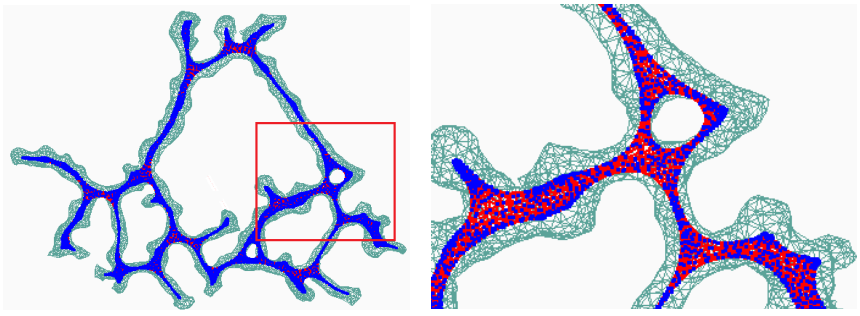
(a) (b)

Figura 5.7: (a) Malla de superficie de *retículo* contraída hasta alcanzar el 10 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).



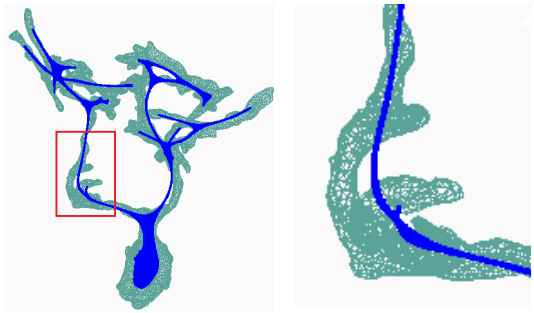
(a) (b)

Figura 5.8: (a) Malla de superficie de *retículo* contraída hasta alcanzar el 15 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).



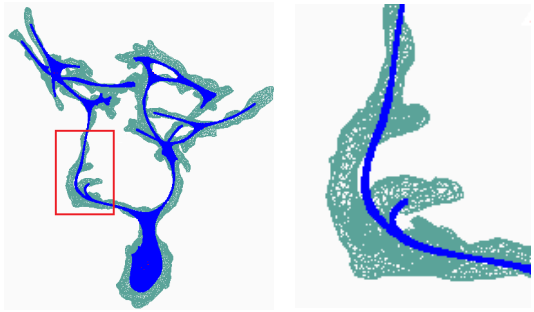
(a) (b)

Figura 5.9: (a) Malla de superficie de *retículo* contraída hasta alcanzar el 20 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).



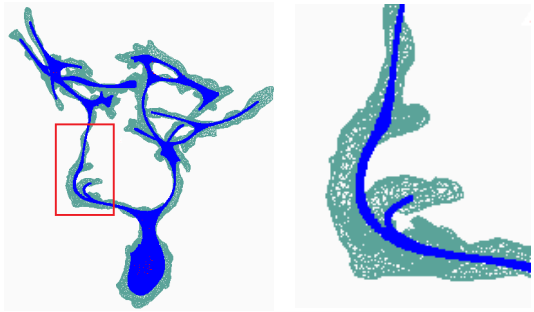
(a) (b)

Figura 5.10: (a) Malla de superficie de la *neurona* contraída hasta alcanzar el 10 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).



(a) (b)

Figura 5.11: Malla de superficie de la *neurona* contraída hasta alcanzar el 15 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).



(a) (b)

Figura 5.12: Malla de superficie de la *neurona* contraída hasta alcanzar el 20 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).

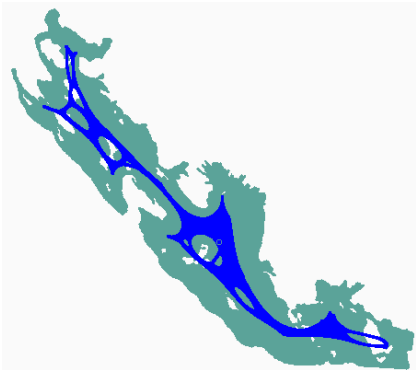


Figura 5.13: Malla de superficie de la *cresta neural* contraída hasta alcanzar el 10 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*).

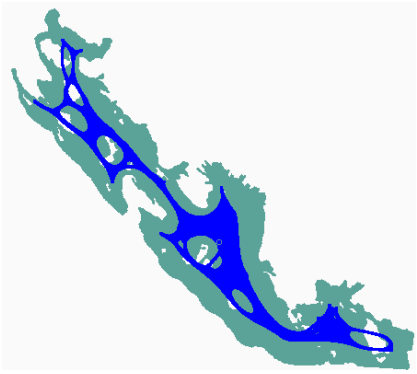


Figura 5.14: Malla de superficie de la *cresta neural* contraída hasta alcanzar el 15 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*).

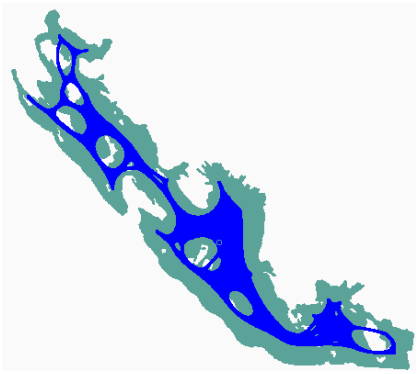
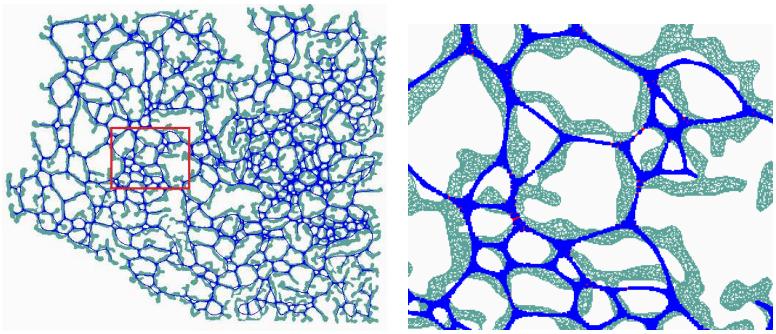
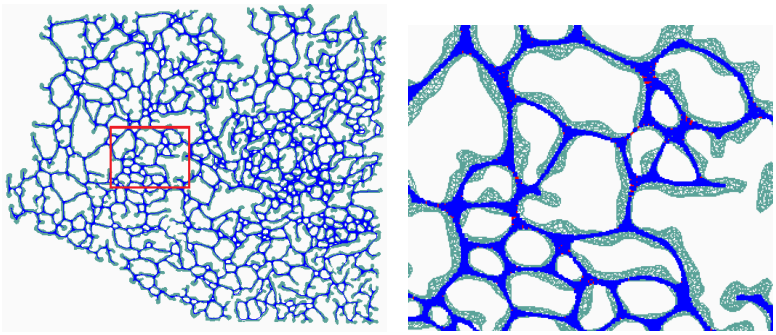


Figura 5.15: Malla de superficie de la *cresta neural* contraída hasta alcanzar el 20 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*).



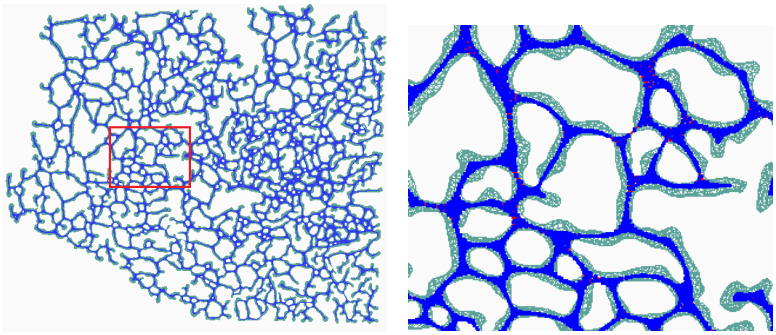
(a) (b)

Figura 5.16: (a) Malla de superficie de *retículo* contraída hasta alcanzar el 10 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).



(a) (b)

Figura 5.17: (a) Malla de superficie de *retículo* contraída hasta alcanzar el 15 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).



(a) (b)

Figura 5.18: (a) Malla de superficie de *retículo* contraída hasta alcanzar el 20 % del área de superficie inicial. Esta malla fue contraída utilizando el método escogido (*GeometryContractionI*). (b) *Zoom* de la zona enmarcada en (a).

En las mallas contraídas hasta alcanzar el 10 % del área de superficie inicial (Figuras 5.7, 5.10 , 5.13 y 5.16), se observa que existen regiones que se ubican completamente fuera de la malla original. Ejemplos de esto se encuentran en las malla de superficies de: la *región de retículo*, la *neurona* y el *retículo* (Figuras 5.7, 5.10 y 5.16, respectivamente). En el caso de la malla de superficie de la *cresta neural* (Figura 5.13) es altamente frecuente encontrar regiones ubicadas completamente fuera de la malla original.

Por otro lado, en las mallas contraídas hasta alcanzar el 20 % del área de superficie inicial (Figuras 5.9, 5.12, 5.15 y 5.18), se observa que éstas no se encuentran suficientemente contraídas a pesar que es poco frecuente encontrar regiones ubicadas fueras de la malla inicial. En el caso de las mallas de superficie de: la *región de retículo*, la *neurona* y el *retículo* (Figuras 5.9, 5.12 y 5.18, respectivamente) se observa que las zonas de la malla contraída ubicadas fuera de la malla original son casos aislados, sin embargo, el área de mallas contraídas no ha disminuido lo suficiente. En el caso de la malla de la *cresta neural* (Figura 5.15) se observa que a pesar que el área de superficie puede continuar decreciendo, la frecuencia de las zonas ubicadas fuera de la malla original es alta.

En las mallas contraídas hasta alcanzar el 15 % del área de superficie inicial (Figuras 5.8, 5.11, 5.14 y 5.17) se observa que existen regiones que se ubican fuera de las mallas iniciales (produciendo resultados topológicamente incorrectos), sin embargo, esto ocurre con menor frecuencia que en las mallas contraídas hasta alcanzar el 10 % del área inicial. En el caso de las mallas de superficie de la *región de retículo*, la *neurona* y el *retículo* (Figuras 5.8, 5.11 y 5.17, respectivamente) se obtienen resultados aceptables donde es poco frecuente encontrar zonas de la malla contraída fuera de la malla original. En el caso de la malla de superficie de la *cresta neural* (Figura 5.14) no se obtienen buenos resultados en la etapa de contracción, pues existen numerosas zonas ubicadas completamente fuera de la malla inicial.

Dado estos resultados, se fijó el umbral como el 15 % del área de superficie inicial. Es importante señalar que aunque algunas zonas de la malla contraída queden ubicadas fuera de la malla original, este defecto no impide que se obtengan *skeletons* correctos y aproximadamente centrados respecto a la malla inicial, pues la etapa de centrado (*embedding refinement*) corrige estos errores.

Algorithm 3 Area Criterion: halt criterion for Geometry Contraction

```
1: procedure IsTrue(mesh)
2:     threshold ← percentage · mesh.initialArea        ▷ threshold ≡ % area
3:     current_area ← mesh.computeArea()
4:     if current_area ≥ threshold then  return true
5:     else return false
6:     end if
7: end procedure

8: function computeArea
9:     area := 0
10:    for all triangle ∈ mesh do
11:        area + = triangle.computeArea()
12:    end for
13:    return area
14: end function

15: function computeArea
16:    a ← triangle.a
17:    b ← triangle.b
18:    c ← triangle.c
19:    s1 := (a.x (b.y − c.y)) − (b.x (a.y − c.y)) + (c.x (a.y − b.y))
20:    s2 := (a.y (b.z − c.z)) − (b.y (a.z − c.z)) + (c.y (a.z − b.z))
21:    s3 := (a.z (b.x − c.x)) − (b.z (a.x − c.x)) + (c.z (a.x − b.x))
22:    return (sqrt(s1^2 + s2^2 + s3^2)) / 2
23: end function
```

2. Criterio de detención basado en el desplazamiento de los vértices

   Durante el proceso de contracción es frecuente que el desplazamiento promedio de los vértices de la malla disminuya a medida que el proceso avanza. Para evitar que la etapa de contracción continue cuando los desplazamientos son cercanos a cero, se implementó este criterio de detención. En la Figura 5.19 se observa el decaimiento del desplazamiento promedio de las mallas de interes a lo largo de la etapa de contracción hasta alcanzar el umbral absoluto 0.001[11].

[11] El gráficos del desplazamiento de las mallas generadas artificialmente, se encuentran en el Apéndice B.4, Figura 9.53

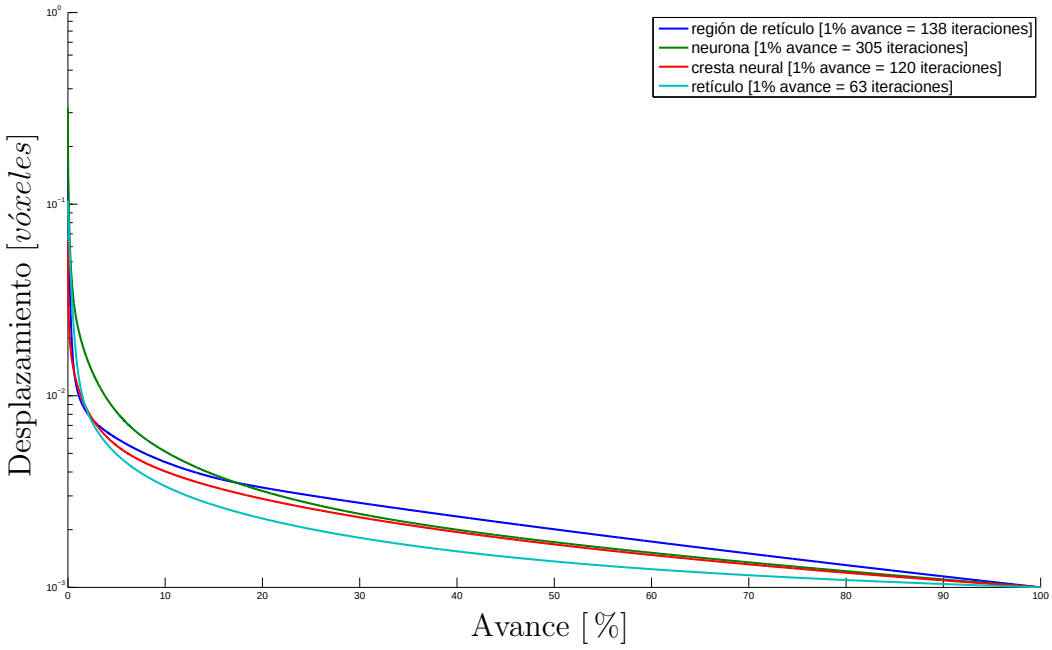


Figura 5.19: Promedio del desplazamiento de los vértices de las mallas biológicas durante la etapa de contracción (*geometry contraction*). Para contraer la malla se utilizó el método escogido (*GeometryContractionI*) hasta que el promedio de desplazamiento alcanzó el umbral 0.001.

En el gráfico de la Figura 5.19 se observa que todas las mallas de superficie alcanzan el umbral absoluto 0.001. Por otro lado, se observa que en las mallas biológicas, el proceso de contracción se detiene por el criterio anterior (alcanzar el 15 % de área de superficie inicial), pues el número de iteraciones utilizando ese criterio es menor (Cuadro 5.1).

Malla	N° de iteraciones en la etapa de contracción	
	Criterio de detención basado en % de área de superficie inicial	Criterio de detención basado en desplazamiento promedio de los vértices
región de *retículo*	100	13800
neurona	300	30500
cresta neural	5500	12000
retículo	3300	6300

Cuadro 5.1: Comparación entre el número de iteraciones en la etapa de contracción al utilizar dos criterios de detención distintos. El primer criterio está basado en alcanzar el 15 % de área de superficie inicial y el segundo está basado en alcanzar un desplazamiento promedio mínimo de los vértices de la malla igual a 0.001.

Como se señalo anteriormente, este criterio fue diseñado para evitar que las mallas continuen en la etapa de contracción cuando los desplazamientos son cercanos a cero.

Luego, el criterio de detención principal es el que basado en alcanzar el 15 % de área de superficie inicial y este criterio se considera secundario.

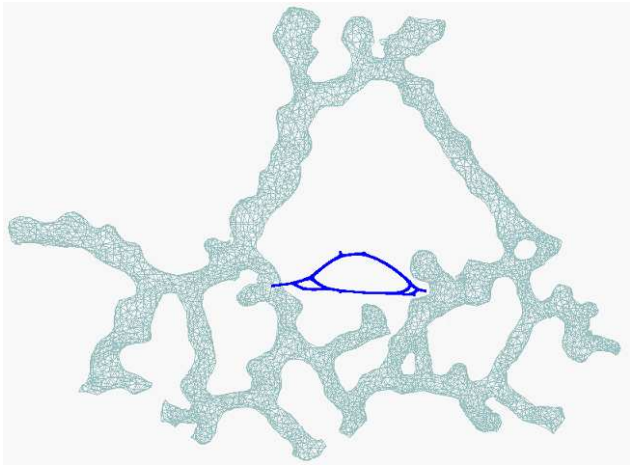


Figura 5.20: Malla de superficie de la región de *retículo* contraída utilizando como criterio de detención el promedio de desplazamiento de los vértices. Para contraer la malla se utilizó el método escogido (*GeometryContractionI*) hasta que el promedio de desplazamiento alcanzó el umbral 0.001.

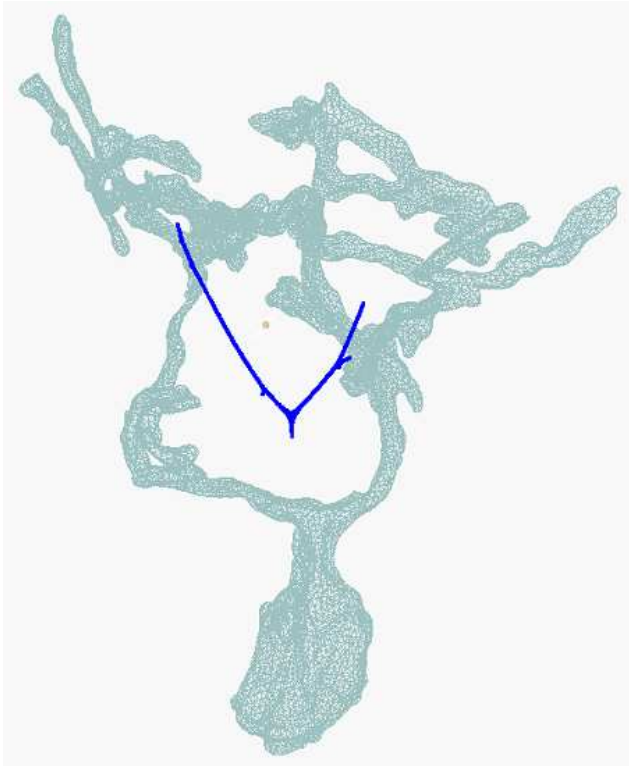


Figura 5.21: Malla de superficie del *neurona* contraída utilizando como criterio de detención el promedio de desplazamiento de los vértices. Para contraer la malla se utilizó el método escogido (*GeometryContractionI*) hasta que el promedio de desplazamiento alcanzó el umbral 0.001.

Figura 5.22: Malla de superficie de la *cresta neural* contraída utilizando como criterio de detención el promedio de desplazamiento de los vértices. Para contraer la malla se utilizó el método escogido (*GeometryContractionI*) hasta que el promedio de desplazamiento alcanzó el umbral 0.001.

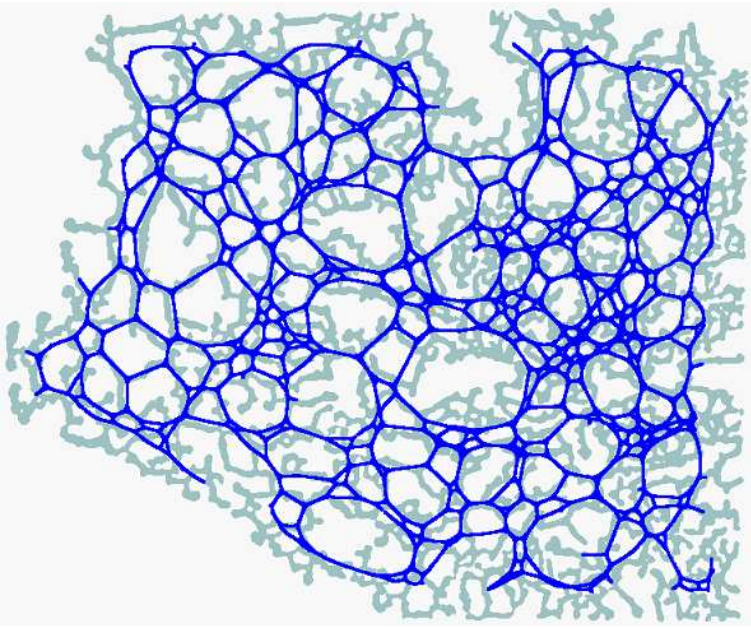


Figura 5.23: Malla de superficie del *retículo* contraída utilizando como criterio de detención el promedio de desplazamiento de los vértices. Para contraer la malla se utilizó el método escogido (*GeometryContractionI*) hasta que el promedio de desplazamiento alcanzó el umbral 0.001.

En Figuras 5.20, 5.21, 5.21, 5.22 y 5.23 se observa que las mallas biológicas contraídas hasta alcanzar un desplazamiento igual a 0.001 poseen un volumen cercano a cero, pero

se encuentran completamente fuera de la malla. Como se mencionó antes, las mallas de se detienen por el criterio anterior, pues este criterio sólo va a funcionar en el caso particular descrito antes.

Algorithm 4 Displacement Criterion: halt criterion for Geometry Contraction

```
1: procedure IsTrue(mesh)
2:     d ← mesh.getAverageDisplacement()
3:     if d ≤ threshold then return false
4:     else return true
5:     end if
6: end procedure

7: function getAverageDisplacement
8:     d := 0
9:     for all vertex ∈ mesh do
10:        d += vertex.displacement
11:    end for
12:    return d / mesh.vertices.size
13: end function
```

Algoritmo

Luego de definir estos criterios de detención para el proceso de contracción, los tres métodos implementados para esta etapa son los siguientes:

Algorithm 5 Geometry Contraction I

```
1: procedure GeometryContractionI(mesh, halt_criterion, displacement_criterion)
2:     w1 := 0.5
3:     w2 := 0.5
4:     while halt_criterion.IsTrue(mesh) do
5:         for all vertex ∈ mesh do
6:             vector ← GetAverageOfPositionNeighbors(vertex)
7:             position_1 ← vertex.position
8:             vertex.position := w1 · vertex.position + w2 · vector.position
9:             position_2 ← vertex.position
10:            vertex.displacement := |position_1 − position_2|      ▷ displacement of vertex
11:        end for
12:        if displacement_criterion.IsTrue(mesh) then
13:            break
14:        end if
15:    end while
16:    return mesh
17: end procedure
```

Algorithm 6 Geometry Contraction II

```
1: procedure GeometryContractionII(mesh, halt_criterion, displacement_criterion)
2:     w1 := 1.0
3:     w2 := 0.001
4:     while halt_criterion.IsTrue(mesh) do
5:         for all vertex ∈ mesh do
6:             normal ← GetNormalVector(vertex)
7:             position_1 ← vertex.position
8:             vertex.position := vertex.position · w1 + normal.position · w2
9:             position_2 ← vertex.position
10:            vertex.displacement := |position_1 − position_2|        ▷ displacement of vertex
11:        end for
12:        if displacement_criterion.IsTrue(mesh) then
13:            break
14:        end if
15:    end while
16:    return mesh
17: end procedure
```

Algorithm 7 Geometry Contraction III

```
1: procedure GeometryContractionIII(mesh, halt_criterion, displacement_criterion)
2:     w1 := 1.0
3:     w2 := 0.01
4:     for all vertex ∈ mesh do
5:         normal ← GetNormalVector(vertex)
6:         vertex.normalVector := normal
7:     end for
8:     while halt_criterion.IsTrue(mesh) do
9:         for all vertex ∈ mesh do
10:            normal ← vertex.normalVector
11:            position_1 ← vertex.position
12:            vertex.position := vertex.position · w1 + normal.position · w2
13:            position_2 ← vertex.position
14:            vertex.displacement := |position_1 − position_2|        ▷ displacement of vertex
15:        end for
16:        if displacement_criterion.IsTrue(mesh) then
17:            break
18:        end if
19:    end while
20:    return mesh
21: end procedure
```

Complejidad de la Etapa de Contracción

Como se ya mencionó, el método escogido es *Geometry Contraction III*. En este método, calcular el promedio de la posición de los vecinos es $\mathcal{O}(n_i)$ donde n_i es el número de vértices vecinos de cada vértice. Luego, el número de operaciones de la etapa de contracción es $\mathcal{O}(nn_i)$, donde n es el número de vértices de la malla.

5.2.2. Etapa *Connectivity Surgery*

Esta etapa consiste en el colapso de los arcos de la malla, escogiendo el arco que tiene el menor costo de acuerdo a una determinada función (Ecuación (4.10)), siempre que dicho arco cumpla con una condición de colapso que asegure mantener la forma y toplogía de la malla original. Este etapa finaliza cuando no existen triángulos en la malla.

Condición de Colapso

Luego del cálculo de la función de costo para cada arco, éstos se introducen en una cola de prioridad. Esta estructura permite ordenar los arcos de menor a mayor costo. A continuación, se extrae un arco de la cola de prioridad y se verifica si cumple la condición de colapso que garantice preservar la topología de la malla.

De acuerdo al Capítulo 4.2.2, la condición de colapso de arcos dada en [2] es la siguiente: si existen tres vértices adyacentes (i, j, k), pero el triángulo $t_{(i,j,k)}$ no existe, entonces se prohibe el colapso ($i \rightarrow j$) (Figura 4.4). En la implementación realizada, esta condición se interpretó de la siguiente forma : si alguno de los dos triángulos incidentes en el arco $e_{(i,j)}$ es nulo, entonces se prohibe el colapso ($i \rightarrow j$). El análisis de esta implementación se encuentra en el Capítulo 7.3.1.

Colapso de los Arcos

El proceso de colapso de arcos se implementó con el siguiente flujo de ejecución:

1. actualizar los arcos de los triángulos incidentes en el vértice i.
2. actualizar referencias a los nuevos arcos.
3. actualizar el vértice i en sus triángulos incidentes.
4. remover el arco $e_{(i,j)}$, el vértice i, sus triángulos incidentes y sus arcos asociados.
5. asociar a j los triángulos y arcos incidentes de i.
6. centrar el vértice resultante, es decir, ubicar j en el punto medio de $e_{(i,j)}$[12].

[12]Centrar la posición de j corresponde a un colapso de medio arco [2].

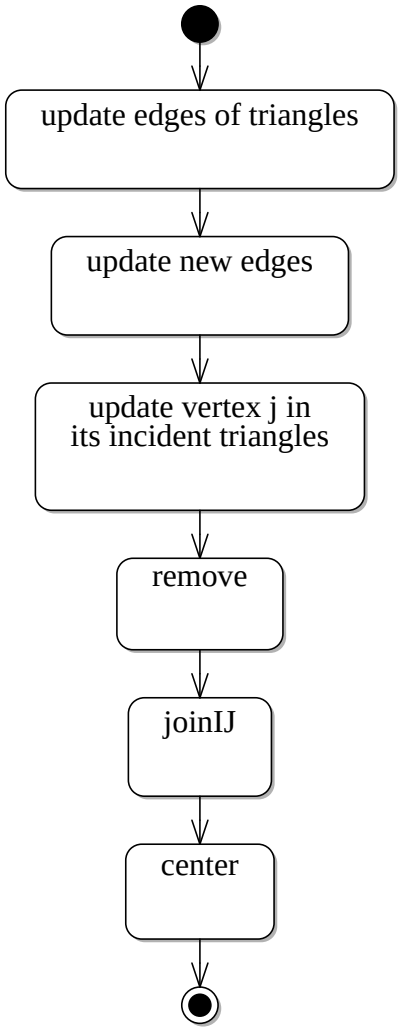
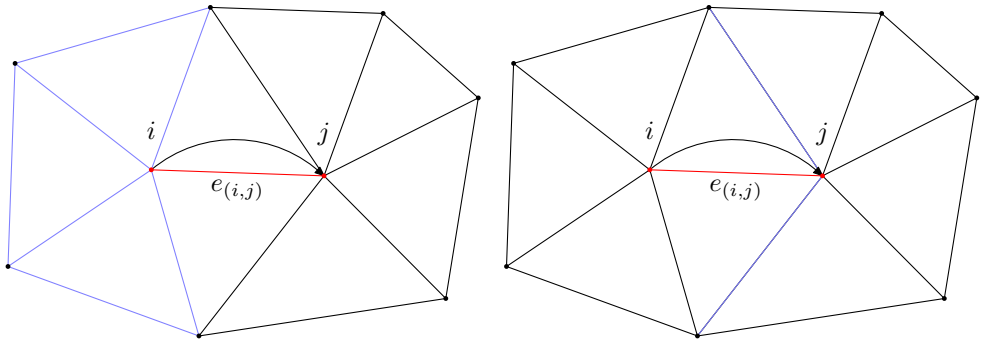
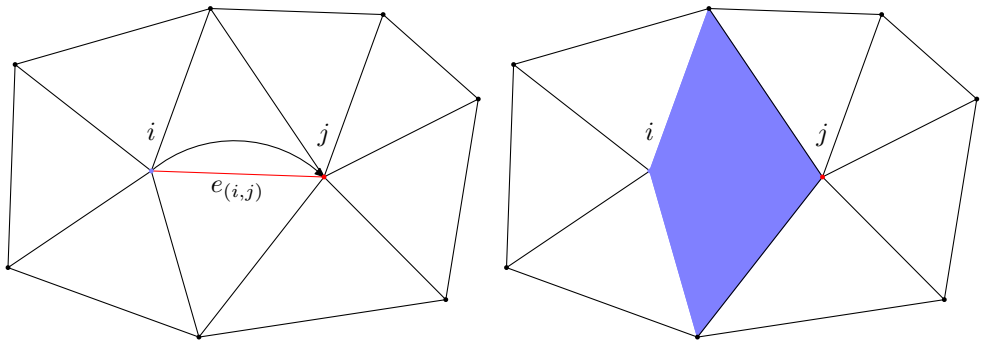


Figura 5.24: Diagrama de actividades del colapso de un arco.



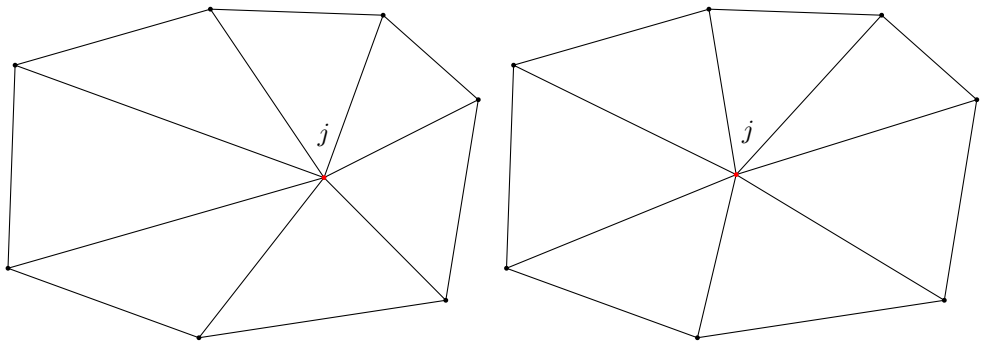
(a) etapa 1 (b) etapa 2

Figura 5.25: Etapas 1 (a) y 2 (b) del colapso del arco $e_{(i,j)}$. En (a) se actualizan los arcos de los triángulos incidentes en i y en (b) se actualizan los nuevos arcos.



(a) etapa 3 (b) etapa 4

Figura 5.26: Etapas 3 (a) y 4 (b) del colapso del arco $e_{(i,j)}$. En (a) se actualiza el vértice i en sus triángulos incidentes y en (b) se remueven $e_{(i,j)}$, i, sus triángulos incidentes y los arcos que están asociados a ellos



(a) etapa 5 (b) etapa 6

Figura 5.27: Etapas 5 (a) y 6 (b) del colapso del arco $e_{(i,j)}$. En (a) se asocian a j los triángulos y arcos incidentes de i y en (b) se centra el vértice resultante.

A medida que avanza la etapa de colapso, la implementación debe soportar distintas configuraciones de los arcos a colapsar (Apéndice A, Figuras 9.23, 9.24, 9.25 , 9.26, 9.27, 9.28, 9.29 y 9.30). Una de ellas ocurre cuando $e_{(i,j)}$ tiene más de dos triángulos incidentes (Apéndice A, Figuras 9.31 y 9.32).

Para ello, se implementaron dos tipos de colapso: el caso general (que ocurre cuando $e_{(i,j)}$ tiene dos triángulos incidentes) y un caso particular que ocurre cuando $e_{(i,j)}$ tiene más de dos triángulos incidentes. En el primer caso, se trabaja directamente con los triángulos incidentes que cada arco almacena en su estructura de datos, mientras que en el segundo caso, se trabaja con una lista que almacena los triángulos incidentes al arco correspondiente. Para crear esta lista, previamente se revisan todos los triángulos vecinos del vértice i, y si alguno de ellos contiene $e_{(i,j)}$, entonces se añade a la lista.

Previo al colapso del arco $e_{(i,j)}$, se realiza el conteo del número de triángulos incidentes. Para ello, se revisa en los triángulos vecinos de i si $e_{(i,j)}$ pertenece a ellos. Dependiendo del caso, el arco se colapsa en la clase *CollapseEdgeGeneralCase* o *CollapseEdgeParticularCase* (Figura 5.28).

En rigor, todos los arcos pueden ser colapsados utilizando la clase *CollapseEdgeParticularCase*, pero el tiempo de procesamiento es mayor cuando se revisa la lista de triángulos incidentes que cuando se accede directamente a ellos a través de variables de instancia. Además, empíricamente, son muy poco frecuentes las ocasiones donde los arcos tienen más de dos triángulos incidentes.

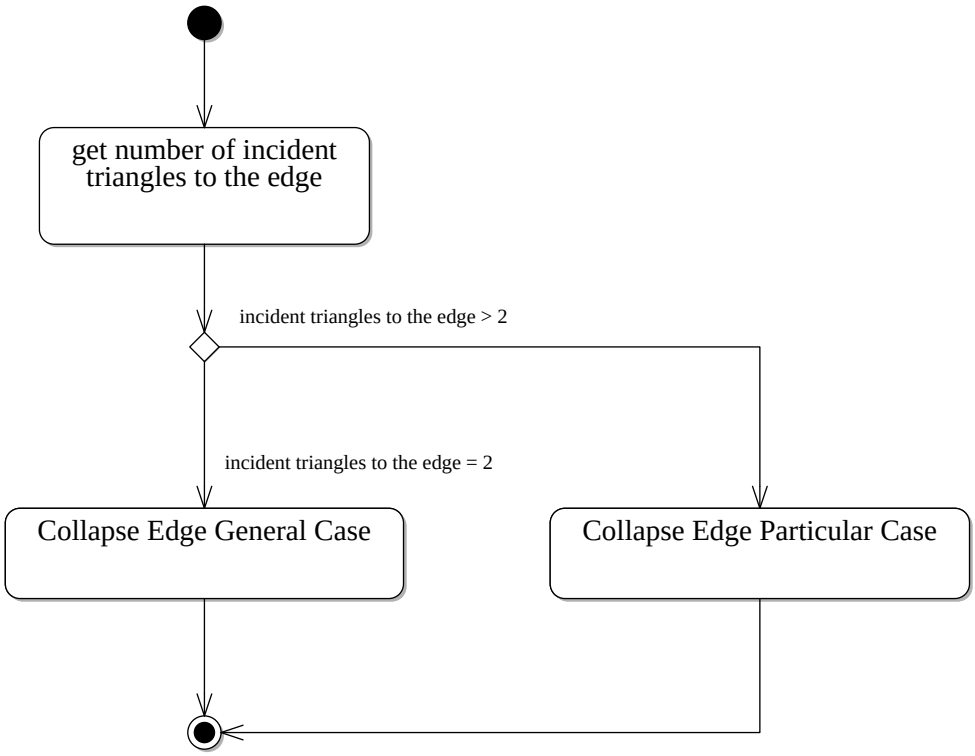


Figura 5.28: Diagrama de actividades del colapso de un arco $e_{(i,j)}$.

Con estos antecedentes, el flujo de ejecución de la primera implementación de esta etapa es el siguiente:

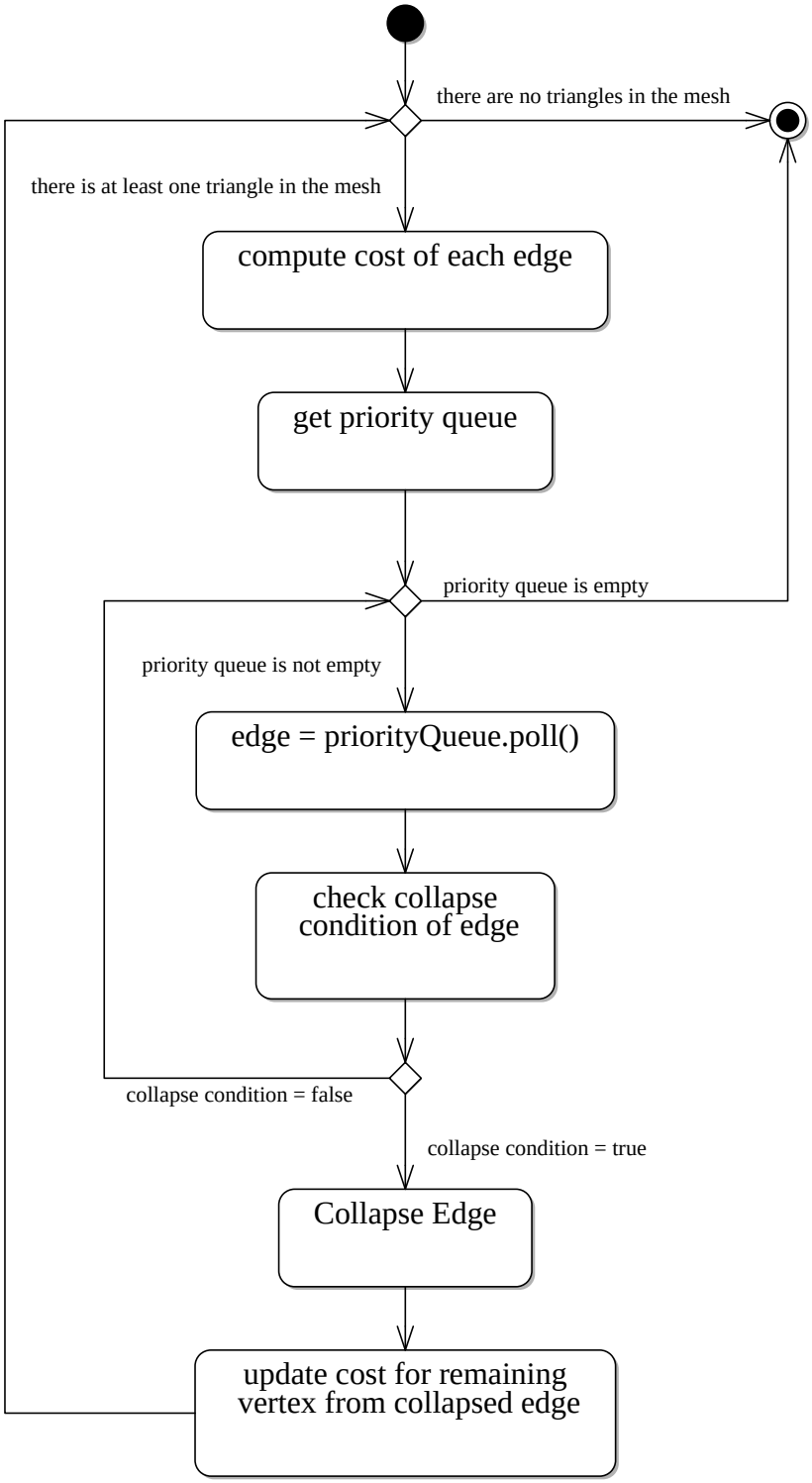


Figura 5.29: Diagrama de actividades de la primera implementación de la etapa de colapso de arcos (*connectivity surgery*).

Revisión y Corrección de la Malla

Luego del colapso ($i \rightarrow j$), se detectaron casos de triángulos y arcos duplicados en una vecindad del vértice j (Figura 5.30). Para corregir esto, se introdujo una subetapa de revisión y corrección de la malla (*check and fix mesh*).

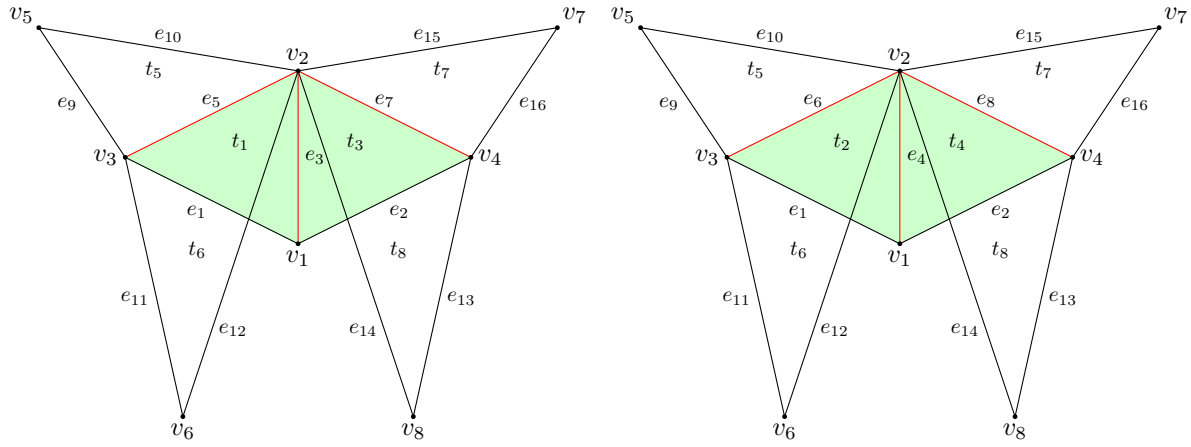


Figura 5.30: Región de malla donde hay triángulos y aristas iguales. Los pares de triángulos en verde: (t_1, t_2) y (t_3, t_4) son iguales. Lo mismo sucede con los pares de arcos en rojo: (e_3,e_4), (e_5, e_6) y (e_7, e_8).

Como se mencionó anteriormente, en esta etapa se realiza una revisión de una vecindad del vértice para evitar triángulos y arcos duplicados en la malla. La vecindad de j escogida corresponde al *three-ring* [13] de este vértice y considera a todos los arcos y triángulos que se encuentran a (máximo) tres arcos de distancia desde j. Las verificaciones realizadas son las siguientes:

1. Revisar y corregir triángulos:

   a) se revisan las referencias a los triángulos incidentes de cada arco

   b) se revisa si existen dos triángulos iguales. Si esto sucede, se elimina el triángulo excedente. Por cada par de triángulos duplicados (t_1, t_2) el triángulo excedente corresponde a t_2.

2. Revisar y corregir arcos:

   a) se revisan las referencias a los triángulos incidentes de cada arco

   b) se revisa si existen dos arcos iguales. Si esto sucede, se elimina el arco excedente. Por cada par de arcos duplicados (e_1, e_2) el arco excedente corresponde a e_2.

Luego de incorporar la revisión y corrección de la vecindad del vértice j, el flujo de ejecución de esta etapa quedó de la siguiente forma:

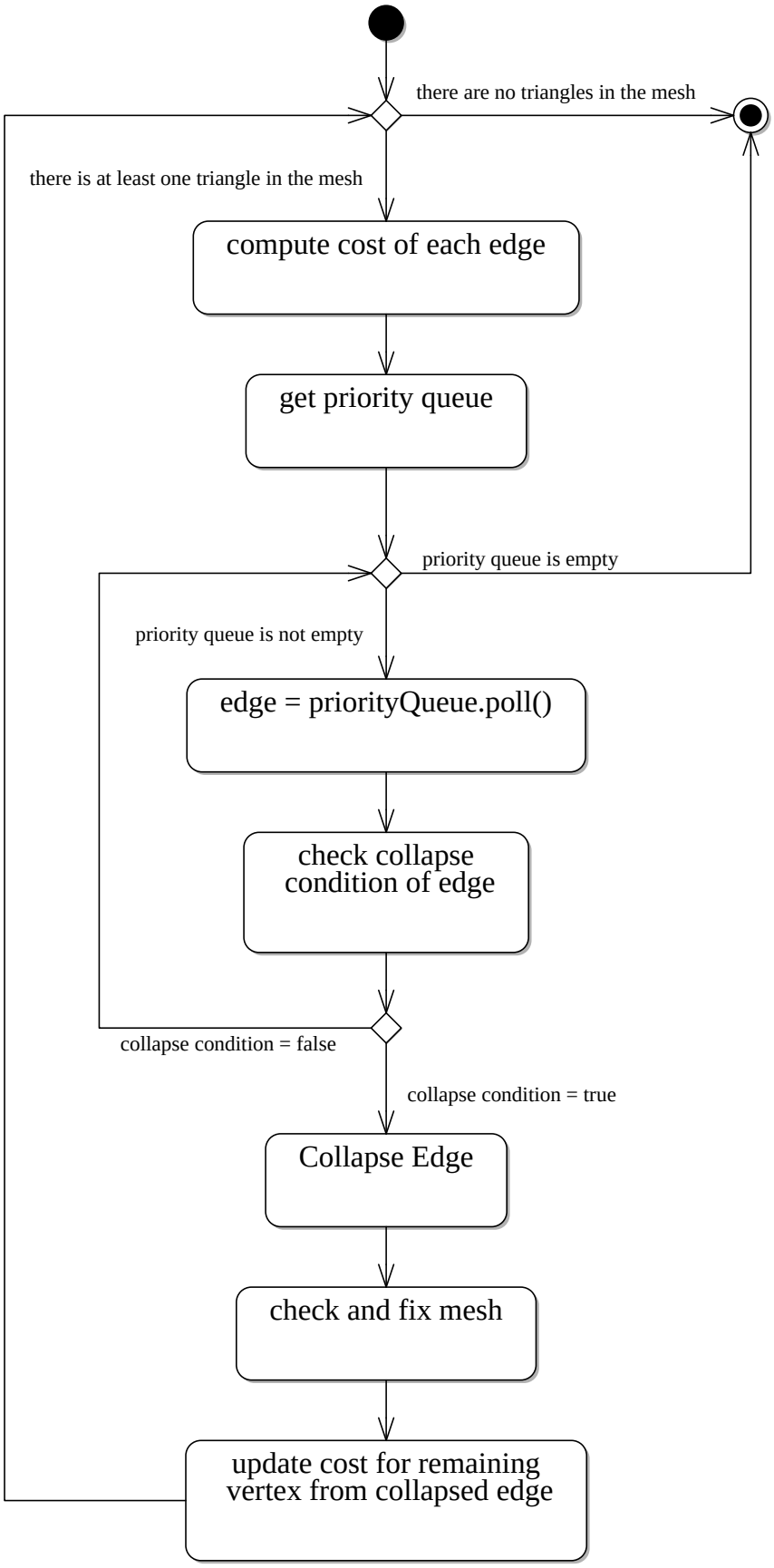


Figura 5.31: Diagrama de actividades de la segunda implementación de la etapa de colapso de arcos (*connectivity surgery*).

Eliminación de triángulos residuales

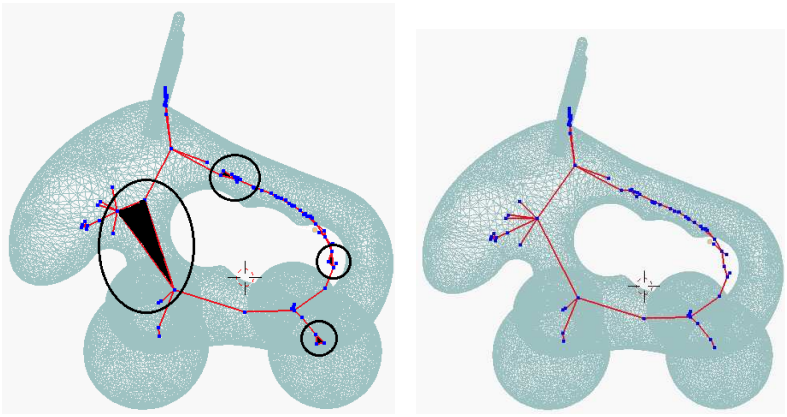
Al realizar el proceso de colapso de arcos según el diagrama de actividades de la Figura 5.31, es frecuente que la salida de esta etapa (*skeleton* no centrado) presente algunos triángulos residuales. Esto ocurre porque los arcos que forman estos triángulos sólo tienen un triángulo incidente, en consecuencia, no cumplen la condición de colapso. Cuando ninguno de los arcos del *skeleton* cumplen la condición de colapso, la cola de prioridad que los almacena queda vacía y el proceso termina (Figura 5.31).

La función que calcula el costo de los arcos mantiene una cola de prioridad que maneja los arcos a colapsar. Para eliminar estos triángulos, se agregó un método que escoge los arcos a colapsar. En el caso general (colapsar arcos utilizando la condición de colapso), se utilizan todos los arcos de la malla. En el caso de que ya no queden arcos que cumplan la condición de colapso y aún existan triángulos en la malla, se elimina esta condición y en la cola de prioridad sólo se introducen los arcos pertenecientes a estos triángulos.

Algorithm 8 GetPriorityQueueEdges

```
1: function GetPriorityQueueEdges(mesh, use_condition)
2:     edges = null
3:     if use_condition ≡ true then
4:         edges ← mesh.edges
5:     else
6:         edges = list()
7:         for all triangle ∈ mesh do
8:             for all edge ∈ triangle do
9:                 if edge ∉ edges then
10:                    edges.add(edge)
11:                end if
12:            end for
13:        end for
14:    end if
15:    return edges
16: end function
```

Al introducir los arcos de los triángulos residuales a la cola de prioridad, se procede a colapsar arcos del mismo modo que en el caso anterior, es decir, se colapsa el arco de menor costo hasta que no existan triángulos en el *skeleton* (Figura 5.32).



(a) (b)

Figura 5.32: *Skeleton* no centrado de la malla de superficie *elk* antes (a) y después (b) de eliminar los triángulos residuales.

Costo de los Arcos

El costo de cada arco está compuesto por dos términos (Ecuación (4.18)): el costo de forma (*shape cost*, Ecuación (4.11)) y el costo de muestreo (*sampling cost*, Ecuación (4.17)).

El comportamiento de ambas funciones para las mallas biológicas son los siguientes[13]:

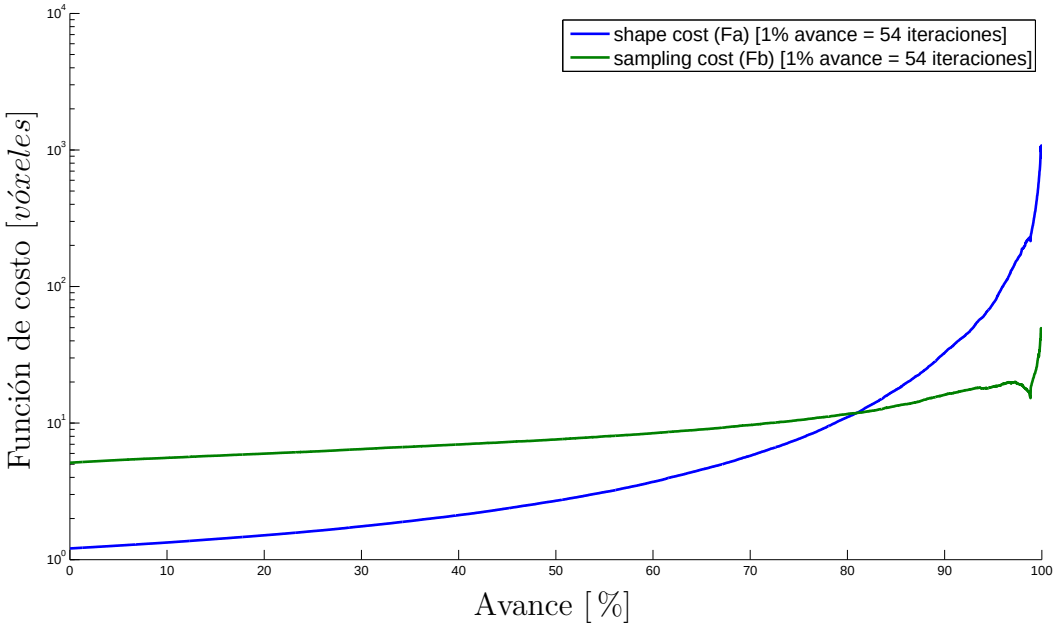


Figura 5.33: Promedio de las funciones de costo de la malla de superficie de la región de *retículo* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

[13]Los gráficos del comportamiento de los dos términos de la función de costo para las mallas generadas artificialmente se encuentran en el Apéndice B.5.

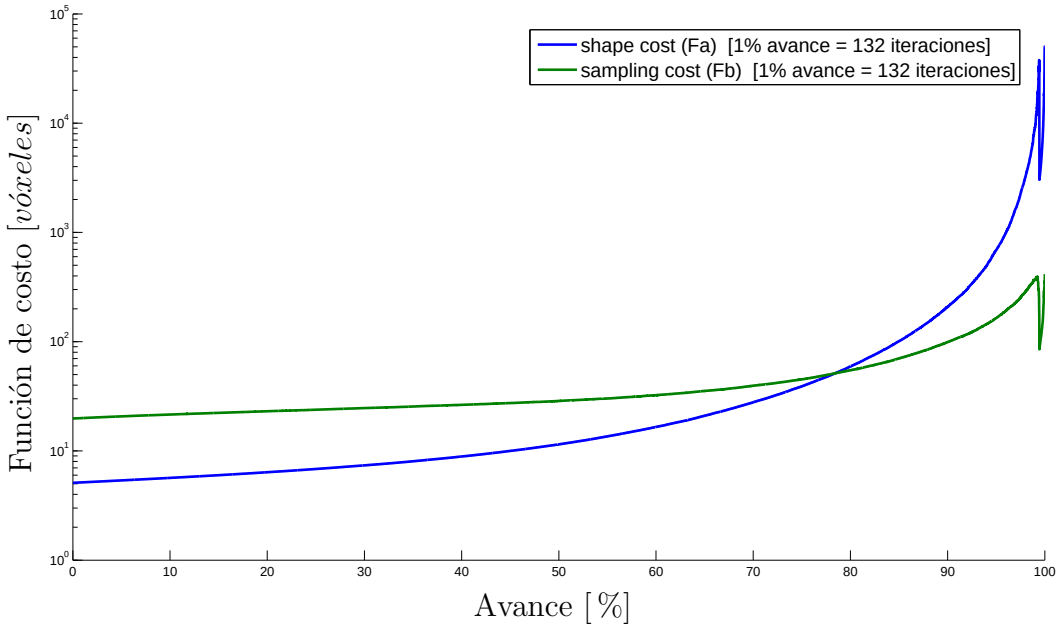


Figura 5.34: Promedio de las funciones de costo de la malla de superficie de la *neurona* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

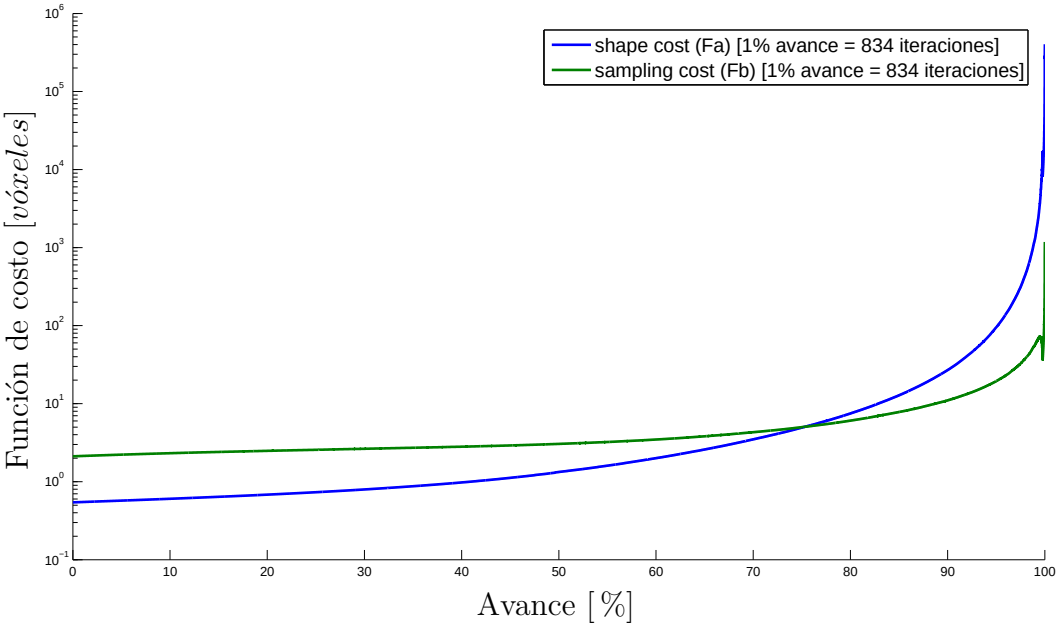


Figura 5.35: Promedio de las funciones de costo de la malla de superficie de la *cresta neural* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

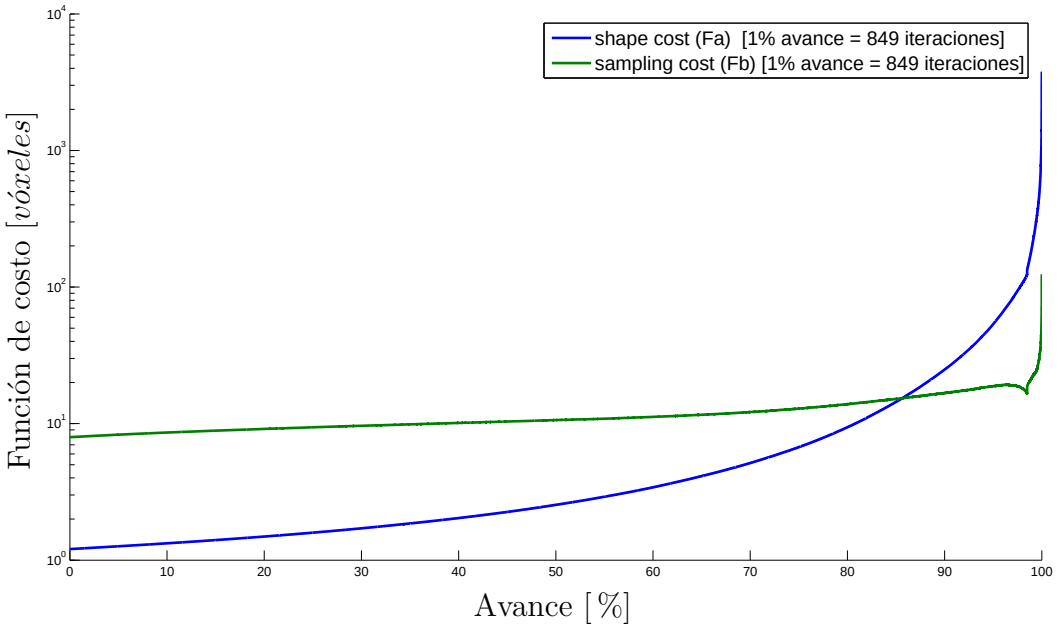
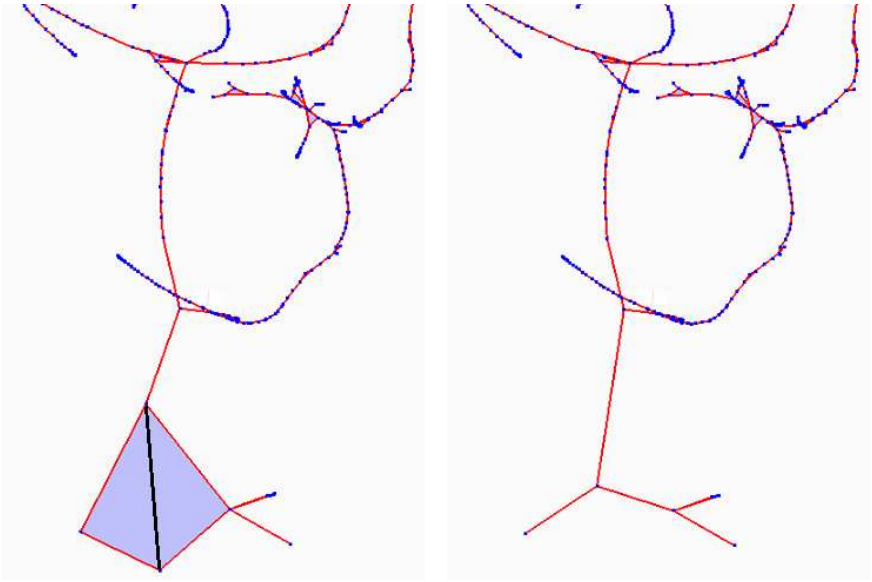


Figura 5.36: Promedio de las funciones de costo de la malla de superficie del *retículo* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

En los gráficos se observa que al principio de la etapa, ambos costos son similares (siendo el costo de muestreo (F_b) levemente superior) hasta que el costo de la forma (F_a) adquiere mayor relevancia superando ampliamente al costo del muestreo. Esto significa que al comienzo de la etapa no es importante que arco se colapsa, pues todos tienen un costo similar, en cambio, al finalizar esta etapa, sí es importante que arco se escoge para colapsar, pues el costo va a depender directamente de número de bifurcaciones que tengan los vértices que lo definen[14].

Además, en todos gráficos se observa que desde el comienzo de la etapa de colapso de arcos hasta el penúltimo tramo de ésta ambas funciones son estrictamente crecientes. Sin embargo, en el último tramo de la etapa de colapso de arcos (cuando la estructura que inicialmente era una malla de superficie ahora tiene aspecto de *skeleton*, salvo algunos triángulos que faltan por remover) se producen cambios de gran magnitud en las funciones de costo. Un ejemplo de esto se observa en el gráfico de las funciones de costo de la malla de la *neurona* (Figura 5.34). En este gráfico se aprecia que la función es estrictamente creciente, alcanzando un máximo local cuando ha transcurrido el 99.5 % de la etapa de colapso; luego la función decrece abruptamente hasta llegar a un mínimo local y finalmente continua aumentando hasta el término del proceso.

[14]Los detalles en la definición del costo de la forma se encuentran en la Ecuación (4.11).



(a) (b)

Figura 5.37: Zonas de la malla de superficie de la *neurona* cuando ocurren cambios importantes en el comportamiento de las funciones de costo durante la etapa de colapso de arcos. (a) Zona de la malla cuando las funciones de costo alcanzan un máximo local (99.5 % de procesamiento). (b) Zona de la malla inmediatamente después del colapso del arco en negro de (a) (cuando las funciones de costo decaen abruptamente).

En la Figura 5.37 se observa que el arco negro es el último arco almacenado en la cola de prioridad que cumple la condición de colapso (poseer dos triángulos incidentes). Después de colapsar este arco hay un cambio en la condición de colapso (ver sección Eliminar triángulos residuales, Capítulo 5.2.2), lo que disminuye drásticamente el costo de los arcos restantes.

Validaciones realizadas

En esta etapa de colapso de aristas se realizan operaciones delicadas y complejas en los elementos de la malla. Para asegurar la correctitud en las referencias en cada una de las fases de este algoritmo se realizaron pruebas unitarias que validan los distintos escenarios posibles. Algunas de las características que garantizan estas pruebas son las siguientes:

1. Si las mallas de entrada del algoritmo son conformes, entonces, la salida de esta etapa es un *skeleton* (posiblemente no centrado) que no contiene triángulos.

2. No existen nodos ni segmentos duplicados en el *skeleton* resultante.

3. Si $\tilde{V}_i = [\tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_k]$ son los vértices de la malla que fueron colapsados al nodo $n_i \in N$ del *skeleton*, entonces se cumple:

$$\tilde{V}_i \cap \tilde{V}_j = \emptyset \quad \forall\, n_i, n_j \ \in N \tag{5.1}$$

Algoritmo

Finalmente, la etapa de colapso de arcos queda definida según el siguiente algoritmo:

Algorithm 9 Connectivity Surgery

```
1:  procedure EXECUTE(mesh)
2:      use_condition := true
3:      while triangles ∈ mesh ≥ 0 do
4:          cost ← COMPUTECOST(mesh, use_condition)
5:          priority_queue ← cost.priorityQueue
6:          while priority_queue ≠ empty do
7:              edge ← priority_queue.poll()
8:              collapse_condition := true
9:              if use_condition then
10:                 collapse_condition := edge.left ≠ null ∧ edge.right ≠ null
11:             end if
12:             if collapse_condition then
13:                 i := edge.i
14:                 j := edge.j
15:                 COLLAPSE(edge, mesh)
16:                 CHECKANDFIX(j, mesh)
17:                 j.q := j.q + i.q                    ▷ update cost of vertex j
18:                 break
19:             end if
20:         end while
21:         if priority_queue ≡ empty then
22:             use_condition := false
23:         end if
24:     end while
25:     return mesh
26: end procedure
```

Complejidad de la Etapa de Colapso

En términos del tiempo de procesamiento, la implementación se encuentra dominada por la etapa de colapso de arcos (*connectivity surgery*).

El costo de calcular el *shape cost* es $\mathcal{O}(m_1)$, donde m_1 es el número de arcos que inciden en cada vértice. El costo de calcular el *sampling cost* también es $\mathcal{O}(m_1)$. Luego, calcular el costo de todos los arcos es $\mathcal{O}(mm_1)$, donde m es el número de arcos de la malla.

El costo de colapsar $(i \to j)$ es $\mathcal{O}(m_i)$, donde m_i es el número de arcos incidentes en el vértice i.

La operación que más pesa en esta etapa es la revisión y corrección de la vecindad j, que es $\mathcal{O}(m_1 m_3^2)$ donde m_3 es el número de nodos en la vecindad del vértice j[15].

[15]La vecindad el vértice j corresponde al *three-ring* [13] de este vértice. Esto corresponde a todos los arcos

Luego, el costo de la etapa de colapso es $\mathcal{O}(m_1 m_3^2 log(r))$, donde r es el número de triángulos de la malla.

Considerando que las mallas de entradas son conformes, entonces ellas cumplen la ecuación característica de Euler para mallas de superficies orientables [20]:

$$V - E + F = 2(1 - genus)$$

donde el *genus* o género de una malla es el número de túneles que presenta.

Luego se tiene que :

$$n = c_1\ m = c_2\ r$$

donde c_1 y c_2 son constantes, y n, m y r corresponden al número de vértices, arcos y triángulos de la malla, respectivamente.

Con esto, se tiene que el costo de la etapa de colapso de arcos es $\mathcal{O}(n_1 n_3^2 log(n))$.

5.2.3. Etapa *Embedding Refinement*

La salida de la etapa anterior corresponde a un *skeleton* que posiblemente se encuentre fuera del objeto original. Además, cada nodo del *skeleton* guarda el historial de los vértices de la malla original que colapsaron en él.

Representatividad de los nodos del *skeleton*

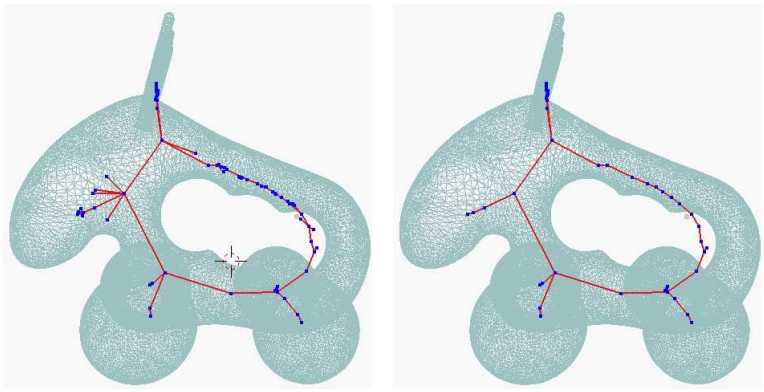
El *skeleton* generado en la etapa de colapso de arcos muchas veces contiene ruido que es inherente al algortimo [2]. Este ruido consiste en nodos terminales que no representan ninguna región de la malla original. Para eliminarlo, se implementó una etapa previa de limpieza del *skeleton* que mediante un criterio de representatividad, elimina los nodos del *skeleton* que contienen menos de 3 vértices colapsados.

Algorithm 10 Representativeness Criterion

```
1: procedure CheckRepresentativeness(skeleton)
2:     for all node ∈ skeleton do
3:         if node ≡ terminal node ∧ node.collapsed_vertices < 3 then
4:             skeleton.remove(node)
5:         end if
6:     end for
7: end procedure
```

y triángulos que se encuentran a (máximo) tres arcos de distancia desde *j*.

Este filtro genera un *skeleton* más 'limpio'y más 'cercanoá la geometría original de la malla (Figura 5.38). El resultado de esta operación se utiliza como el *input* de la etapa de centrado del *skeleton* (*embedding refinement*).



(a) (b)

Figura 5.38: *Skeleton* no centrado la malla de superficie *elk* antes (a) y después (b) de incorporar el criterio de representatividad de los nodos.

Algoritmo

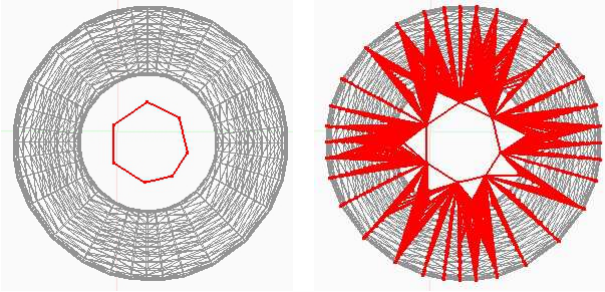
De acuerdo al Capítulo 4.2.3, los nodos del *skeleton* se centran dependiendo de su clasificación.

Algorithm 11 Embedding Refinement

```
1: procedure EmbeddingRefinement(skeleton)
2:     for all node ∈ skeleton do
3:         displacement := DisplacementII(node)
4:         if node ≡ terminal_node then
5:             CenterTerminalNode(node, displacement)
6:         else if node ≡ non_junction_node then
7:             CenterNonJunctionNode(node, displacement)
8:         else if node ≡ junction_node then
9:             CenterJunctionNode(node, displacement)
10:        end if
11:    end for
12: end procedure
```

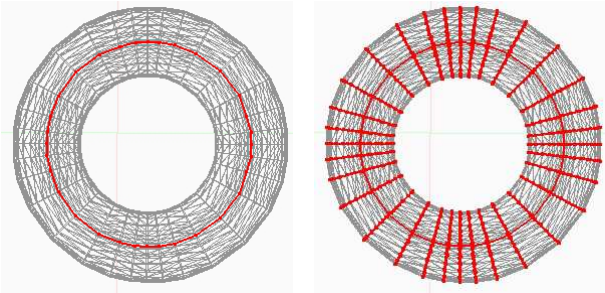
Región local y *boundary loops* de los nodos del *skeleton*

Según la Ecuación (4.21), el *boundary loop* $bl(n_i, n_k)$ se define como la frontera entre la región local Π_{n_i} con el nodo n_k, donde Π_{n_i} son todos los vértices que colapsaron en n_i. Como se ilustra en las Figuras 5.39 y 5.40, existe una correlación entre Π_{n_i} y el método de contracción que se utiliza en la etapa de contracción.



(a) (b)

Figura 5.39: (a) *Skeleton* no centrado de la malla de superficie de un toroide o *donuts*. Esta malla fue contraída utilizando como desplazamiento la posición promedio de los vecinos de cada vértice (*Geometry Contraction I*). (b) Región local Π de los nodos del *skeleton* de la figura (a).



(a) (b)

Figura 5.40: (a) *Skeleton* no centrado de la malla de superficie *donuts*. Esta malla fue contraída utilizando como desplazamiento el vector normal de cada vértice (*Geometry Contraction III*). (b) Región local Π de los nodos del *skeleton* de la figura (a).

El *boundary loop* $bl(n_i, n_k)$ se obtiene de la siguiente forma: si los vértice $v_m, v_l \in V$ de la malla inicial colapsaron en los nodos $n_i, n_k \in N$, respectivamente y se encuentran conectados a través de un arco $e_{(m,l)}$, entonces $v_m \in bl(n_i, n_k)$.

Algorithm 12 FindBoundaryLoop

```
1: function FindBoundaryLoop(n_i, n_k)
2:     loop := list()
3:     for all vertex ∈ n_i.collapsed_vertices do
4:         for all edge ∈ vertex.initialVertex.oneRing do
5:             neighbor ← edge.getNeighbor(vertex)
6:             if neighbor.skeletonNode.index ≡ n_k.index then
7:                 loop.add(vertex)
8:                 break
9:             end if
10:        end for
11:    end for
12:    return loop
13: end function
```

Desplazamiento

El desplazamiento $d(bl(n_i, n_k))$ que permite centrar los nodos del *skeleton* , se define de acuerdo a la Ecuación (4.21):

Algorithm 13 Displacement

```
1: function getD(n_i, n_k)
2:     loop ← FindBoundaryLoop(n_i, n_k)
3:     vector_d := 0
4:     l := 0
5:     l_total := 0
6:     for all vertex ∈ loop do
7:         δ ← vertex.position − vertex.inicial_vertex.position
8:         l ← getL(vertex)
9:         vector_d+ = (l · δ)
10:        l_total+ = l
11:    end for
12:    vector_d := vector_d / l_total
13:    return vector_d
14: end function
```

La diferencia entre los dos enfoques de desplazamiento, se encuentra en la definición de l (Ecuación (4.21)).

Desplazamiento I (*Displacement I*)

En este enfoque, l_j se define como el largo de dos arcos adyacentes a *vertex* en su respectivo *boundary loop*. Esto indica que el *boundary loop* debe ser una curva convexa para asi poder obtener un valor correcto de l_j. El método de ordenamiento implementado es un algoritmo *greedy* que a partir de un vértice v_i, busca el vértice v_k más cercano en el *boundary loop*.

Algorithm 14 Sort Boundary Loop

```
1: function SortBoundaryLoop(loop)
2:     sorted_loop := list()
3:     vertex ← loop.firstElement()
4:     sorted_loop.add(vertex)
5:     while sorted_loop.size < loop.size do
6:         vertex_aux1 ← sorted_loop.getLastElement()
7:         vertex_aux2 ← getClosestVertex(vertex_aux1, loop, sortedLoop)
8:         sorted_loop.add(vertex_aux2)
9:     end while
10:    return sorted_loop
11: end function

12: function getClosestVertex(vertex, loop, sorted_loop)
13:    closest_vertex := null
14:    distance := 0
15:    d_min := 10000
16:    for all vertex_aux ∈ loop do
17:        if vertex_aux.index ≠ vertex.index ∧ vertex_aux ∉ sortedLoop then
18:            distance ← getDistance(vertex_aux.initial_vertex, vertex.initial_vertex)
19:            if distance < d_min then
20:                d_min := distance
21:                closest_vertex := vertex_aux
22:            end if
23:        end if
24:    end for
25:    return closest_vertex
26: end function
```

Para encontrar el vértice más cercano, se implementó el algoritmo *Flood fill*, un método para encontrar el camino más corto entre dos vértices de un grafo [6]. Este algoritmo considera una lista de vértices a inundar que inicialmente contiene el vértice de partida (s), luego se expande a todos los vecinos de esta lista, introduciéndolos en ella si ninguno de ellos es el vértice de término (e). Esto se repite sucesivamente hasta que se encuentra el vértice e.

Algorithm 15 Flood Fill

```
1: function GetDistance(s, e)
2:     to_expand := list()
3:     to_clean := list()
4:     distance := 0
5:     to_expand.add(s)
6:     while to_expand.size > 0 ∨ e.d! = −1 do
7:         to_expand ← floodFill(to_expand, ++distance)
8:         to_clean.add(to_expand)
9:     end while
10:    result ← e.d
11:    for all vertex ∈ toClean do
12:        vertex.d := −1;
13:    end for
14:    return result
15: end function

16: function floodFill(to_expand, distance)
17:    list := list()
18:    for all vertex ∈ to_expand do
19:        for all edge ∈ vertex.oneRing do          ▷ oneRing: neighbors of vertex [13]
20:            neighbor ← edge.getNeighbor(vertex)
21:            if neighbor.d ≡ −1 then
22:                neighbor.d := distance
23:                list.add(neighbor)
24:            end if
25:        end for
26:    end for
27:    return list
28: end function
```

Desplazamiento II (*Displacement II*)

Como se mencionó previamente, en [2] proponen definir l_j como el largo total de dos arcos adyacentes al vértice $v_j \in bl(n_i, n_k)$. Esta definición esta hecha bajo el supuesto que la región Π_i tiene una forma cilindríca. La forma de la región Π_{n_i} tiene una fuerte correlación con el método escogido en la etapa de contracción. Utilizando la posición promedio de los vecinos, la forma de Π_{n_i} no es regular y frecuentemente se intersecta con las regiones de otros nodos. Es por esto que se implementó una segunda alternativa en la definición de l_j. Esta definición de l_j corresponde a la Ecuación (4.22): la distancia entre la posición inicial del vértice y el nodo n_i del *skeleton* al que pertenece.

Complejidad de la Etapa de Centrado

Como se señalo anteriormente, el desplazamiento escogido para la etapa de centrado (*embedding refinement*) es (*Displacement II*). Luego, el número de operaciones de esta etapa es

$\mathcal{O}(\overline{n})$, donde $\overline{n}$ es el número de nodos del *skeleton* resultante.

Finalmente, el algoritmo implementado es $\mathcal{O}(n_1 n_3^2 log(n))$.

5.3. Visualización

5.3.1. IDL

IDL es un ambiente computacional para el análisis y la visualización de datos que integra un lenguaje estructurado orientado a arreglos con técnicas de análisis matemático.

Algunas caraterísticas de IDL son las siguientes:

- Posee operadores y funciones que trabajan sobre arreglos, encapsulando el uso de ciclos iterativos.
- Incorpora funcionalidades y herramientas para gráficos multidimensionales, despliegue de imágenes y animaciones.

El código fuente, asi como la versión ejecutable de los programas escritos en IDL se almacenan en distintos tipos de archivos. Una aplicación en IDL contiene los siguientes archivos:

- Proyecto (.prj): es un archivo en formato propietario que contiene información sobre la ubicación y organización de los demás archivos del proyecto, además de un conjunto de opciones para la compilación, generación de ejecutables y ejecución de la aplicación.
- Código fuente (.pro): contiene la codificación de los procedimientos o funciones de la aplicación en formato ASCII. También pueden emplearse para definir clases y tipos de datos.
- Almacenamiento (.sav): archivos binarios que permiten almacenar programas, procedimientos compilados y guardar información sobre las variables y recursos manejados por la aplicación, para ser restaurados en distintas sesiones de trabajo.

5.3.2. *SCIAN-Soft*

SCIAN-Soft es un programa para tratamiento y análisis de imágenes científicas desarrollado en el *SCIAN-lab*. Proporciona funciones para manejo de imágenes 2D/3D en varios canales y en series de tiempo; incorpora filtros para segmentación y reconstrucción de modelos 2D/3D, análisis morfotopológico y visualización interactiva.

5.3.3. Integración

Dentro del esquema del proyecto existe un archivo llamado *ROI3DGroupObjectDefine.pro*. Este archivo define una clase que genera diferentes tipos de vistas de los objetos a analizar. Uno de estos métodos es *getSurfaceModel*, que genera la malla de entrada del software actualmente desarrollado.

Para generar la malla contraída, en la clase *ROI3DGroupObject* se creó el método *getSkeletonFromMeshModel*[16] que hace una llamada a un programa *Java* empaquetado en formato *jar* que contiene la implementación realizada en este trabajo de título.

Las funciones implementadas en el método *getSkeletonFromMeshModel* para ejecutar las etapas del algoritmos son las siguientes:

- *Mesh geometryContraction*(*double*[] *x*, *double*[] *y*, *double*[] *z*, *int*[] *triangles*): ejecuta la etapa de contracción de la malla. Recibe como argumento las posiciones de los vértices, y una lista con los triángulos de la malla y sus respectivos vértices. Retorna la malla contraída.
- *Mesh connectivitySurgery*(): ejecuta la etapa de colapso de arcos. Retorna el *skeleton* (no centrado) de la malla.
- *Skeleton embeddingRefinement*(): ejecuta la etapa de centrado del skeleton. Retorna el *skeleton* de la malla.

Las funciones implementadas en el programa *Java* para acceder desde IDL a los elementos del *output* de cada etapa son las siguientes:

- *double*[][] *getVerticesCoordsIDL*(): devuelve las coordenadas de los vértices de la malla (o de la malla contraída). Las coordenadas corresponden a un arreglo de $3 \times n$, donde n es el número de vértices. El formato de éstos es el siguiente:

$$\begin{aligned} coordinates[0][i] &= v_i.getPosition().getX() \\ coordinates[1][i] &= v_i.getPosition().getY() \\ coordinates[2][i] &= v_i.getPosition().getZ() \end{aligned}$$

  donde i es el índice del vértice $v_i \in V$.

- *int*[] *getEdgesForIDL*(): retorna una lista con los índices de los dos vértices de cada arco. La lista es un arreglo de tamaño $3n$, donde n es el número de arcos de la malla. El formato de éstos es el siguiente:

$$\begin{aligned} edges[i] &= 2 \\ edges[i+1] &= e_i.getA().getIndex() \\ edges[i+2] &= e_i.getB().getIndex() \end{aligned}$$

  donde i es el índice del arco $e_i \in E$ y $edges[i\ ]$ se emplea para indicar que se trabaja con arcos por compatibilidad con el formato de descripción de mallas en IDL.

[16]Para más detalles del código en IDL, ver Apéndice G, Figuras 9.66, 9.67 y 9.68.

- *int*[] *getTrianglesForIDL*(): retorna una lista con los índices de los tres vértices de cada triángulo. La lista es un arreglo de tamaño $4n$, donde n es el número de triángulos de la malla. El formato de éstos es el siguiente:

$$\begin{aligned} triangles[i] &= 3 \\ triangles[i+1] &= t_i.getA().getIndex() \\ triangles[i+2] &= t_i.getB().getIndex() \\ triangles[i+3] &= t_i.getC().getIndex() \end{aligned}$$

  donde i es el índice del triángulo $t_i \in F$.

- *double*[][] *getNodesCoordsIDL*(): devuelve las coordenadas de los nodos del *skeleton*. Las coordenadas corresponden a un arreglo de $3 \times n$, donde n es el número de nodos. El formato de éstos es el siguiente:

$$\begin{aligned} coordinates[0][i] &= n_i.getPosition().getX() \\ coordinates[1][i] &= n_i.getPosition().getY() \\ coordinates[2][i] &= n_i.getPosition().getZ() \end{aligned}$$

  donde i es el índice del vértice $n_i \in N$.

- *int*[] *getSegmentsForIDL*(): retorna una lista con los índices de los dos nodos de cada segmento. La lista es un arreglo de tamaño $3n$, donde n es el número de segmentos del *skeleton*. El formato de éstos es el siguiente:

$$\begin{aligned} segments[i] &= 2 \\ segments[i+1] &= s_i.getNode1().getIndex() \\ segments[i+2] &= s_i.getNode2().getIndex() \end{aligned}$$

  donde i es el índice del segmento $s_i \in U$.

Capítulo 6

Evaluación

En este capítulo se encuentran los resultados obtenidos en este trabajo de memoria. Además, se presenta una descripción del *hardware* utilizado y la evaluación del diseño propuesto de la aplicación.

6.1. Evaluación del Diseño

Para obtener las métricas de la implementación final se utilizó la herramienta *Understand*. En primer lugar, se muestran las métricas generales de la implementación:

Métrica	Valor
N° de clases	55
N° de archivos	55
N° de líneas	6732
N° de líneas de código	4062
N° de líneas en blanco	2146
N° de líneas de comentarios	524
Radio comentarios/código	0,13

Cuadro 6.1: Métricas generales de la implementación final.

Por otra parte, se pueden observar las métricas de programación orientada a objetos (*OOP*)[1]. A continuación, una breve explicación de las métricas usadas.

- *CountDeclMethod (CDM)*: Número de métodos locales (no heredados).
- *CountDeclMethodAll (CDMA)*: Número de métodos, incluyendo aquellos heredados.
- *MaxInheritanceTree (MIT)*: Profundidad de una clase dentro de la jerarquía de la herencia. Corresponde a la altura del árbol de la herencia.

[1] *Object oriented programming*, programación orientada a objetos [11].

- *CountClassCoupled (CCC)*: Número de clases utilizadas. Cualquier número de acoplamiento a una determinada clase de cuenta como 1 en dirección a la métrica total.
- *CountClassDerived (CCD)*: Número de inmediato subclases, es decir, el número de clases de un nivel en el árbol de la herencia de esta clase.
- *PercentLackOfCohesion (PLC)*: Evalúa el grado de disimilitud de los métodos en una clase por ejemplo, las variables de instancia o atributos.

Class	CDM	CDMA	MIT	CCC	CCD	PLC
SkeletonExtractionFromMeshContraction	12	12	1	15	0	66
GeometryContraction	2	2	1	2	3	50
GeometryContractionI	2	4	2	6	0	0
GeometryContractionII	2	4	2	6	0	0
GeometryContractionIII	2	4	2	6	0	0
Displacement	2	2	1	2	2	0
AveragePositionOfNeighbors	2	4	2	4	0	0
NormalVector	2	4	2	4	0	0
Criterion	2	2	1	1	2	50
AreaCriterion	2	4	2	1	0	0
DisplacementCriterion	2	4	2	1	0	0
ConnectivitySurgery	2	2	1	6	0	0
TotalCost	4	4	1	7	0	50
ShapeCost	8	8	1	7	0	62
SamplingCost	3	3	1	6	0	33
EdgeCost	20	20	1	1	0	87
EdgeCostComparator	1	1	1	2	0	0
Collapse	3	3	1	8	0	33
CollapseEdge	11	11	1	6	2	86
CollapseEdgeGeneralCase	5	16	2	5	0	60
CollapseEdgeParticularCase	5	16	2	5	0	25
CheckAndFix	6	6	1	7	0	54
Fix	8	8	1	4	0	0
EmbeddingRefinement	2	2	1	10	0	0
CheckRepresentativeness	2	2	1	4	0	0
RepresentativenessCriterion	2	2	1	2	0	0
Center	2	2	1	2	3	50
CenterJunctionNode	2	4	2	4	0	0
CenterNonJunctionNode	2	4	2	4	0	0
CenterTerminalNode	2	4	2	4	0	0
Displacement	4	4	1	5	2	62
DisplacementI	5	9	2	4	0	0
DisplacementII	3	7	2	3	0	0
SortBoundaryLoop	5	5	1	3	0	50
MatrixUtils	3	3	1	0	0	0
Vector	17	17	1	0	0	44

Cuadro 6.2: Métricas *OOP* de la implementación final parte I.

Class	CDM	CDMA	MIT	CCC	CCD	PLC
Mesh	14	14	1	4	0	64
Edge	27	27	1	3	0	88
Triangle	25	25	1	3	0	76
Vertex	26	26	1	3	0	90
OneRing	5	5	1	3	2	53
ThreeRing	5	10	2	3	0	40
Skeleton	7	7	1	7	0	54
Node	12	12	1	3	0	76
Segment	8	8	1	2	0	68

Cuadro 6.3: Métricas *OOP* de la implementación final parteII.

En la tablas anteriores se puede observar que en términos generales, las clases implementadas se encuentran cohesionadas, a excepción de las clases Edge, Triangle, Vertex. Esto se explica por los métodos accesores (*get*, *set*) que permiten realizar correctamente la etapa de colapso de arcos (*connectivity surgery*).

Además, existen dos clases que también se encuentran acopladas. Ellas son *EdgeCost* y *CollapseEdge*. *EdgeCost* es una clase que básicamente contiene funciones *get* y *set* para los costos del arco $e_{(i,j)}$ (*shape cost* y *sampling cost* para el colapso $(i \to j)$ y $(j \to i)$). *CollapseEdge*, es la clase abstracta de la que extienden las dos clases que implementan el colapso de un arco (*CollapseEdgeGeneralCase* y *CollapseEdgeParticularCase*). El alto acoplamiento se explica porque maneja el algoritmo de colapso de $e_{(i,j)}$ y las interacciones entre los elementos involucrados (*Mesh, Edge, Vertex* y *Triangle*). Por lo tanto, no es posible disminuir el acoplamiento de esta clase.

Finalmente, se observa que la clase principal *SkeletonExtractionFromMeshContraction*, asi como los paquetes que manejan las tres etapas del algoritmo se encuentran cohesionadas, por lo que se consideran positivos los valores obtenidos en las métricas. Esto es muy importante, pues muestra que este trabajo de título es fácilmente mantenible, lo que permite mejorarlo y/o extenderlo en futuros proyectos.

6.2. Evaluación de la Implementación

6.2.1. Hardware Utilizado

El desarrollo de este trabajo se realizó en un computador con las siguientes características:

- procesador: Intel Core i7, 950, 3.07 GHz
- memoria RAM: 24 GB
- sistema operativo: Windows 7 Professional, 64 bits

6.2.2. Resultados

Actualmente no se cuenta con métricas robustas para evaluar la correctitud del *skeleton* obtenido en función sus propiedades, en particular el centrado. Para poder evaluar de alguna forma los resultados obtenidos se realizaron dos pruebas:

1. Se compararon los *skeletons* obtenidos a partir de dos mallas de superficie de forma toroidal que cumplen la siguiente ecuación:

$$(R - \sqrt{x^2 + y^2})^2 + z^2 = r^2$$

donde:

- r: radio interior del toroide = 20 [*vóxeles*]
- R: radio exterior del toroide = 100 [*vóxeles*]
- x: posición x de los nodos del *skeleton* [*vóxeles*]
- y: posición y de los nodos del *skeleton* [*vóxeles*]
- z: posición z de los nodos del *skeleton* [*vóxeles*]

Adicionalmente a estas mallas se les introdujo ruido aditivo de una distribución uniforme $\in [0, 2r]$ ortogonal a la superficie. El ruido aditivo se encuentra definido por la ecuación:

$$r = r + rand(1) \cdot \delta \tag{6.1}$$

donde:

- r: radio interior del toroide [*vóxeles*]
- rand(1): función aleatoria uniforme $\in [0, 1]$ proporcionada por MATLAB[2]
- δ: parámetro del ruido gausiano $\in [0, r]$ [*vóxeles*]

Para cada una de las malla toroidales, se utilizaron diferentes parámetros δ en la ecuación de ruido gausiano (Cuadro 6.5):

Malla	Parámetro δ
$toroide_1$	0
$toroide_2$	$\frac{r}{2}$
$toroide_3$	r

Cuadro 6.4: Parámetro δ de la ecuación de ruido gausiano utilizado en las mallas toroidales.

A continuación, se calculó el error del *skeleton* generado comparando la posición de los nodos obtenidos con la posición del *skeleton* perfecto que cumple la ecuación:

[2]http://www.mathworks.com

$$x^2 + y^2 = R^2$$

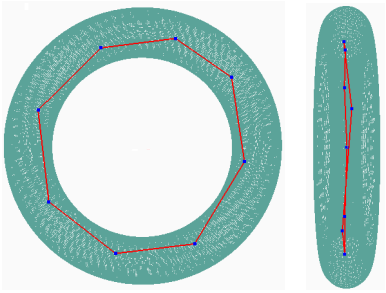
De esta forma, el error del *skeleton* corresponde al error promedio de los nodos:

$$error = \frac{\sqrt{(\sqrt{x^2+y^2} - R)^2 + z^2}}{n} [vóxeles] \tag{6.2}$$

donde n es el número de nodos del *skeleton*.

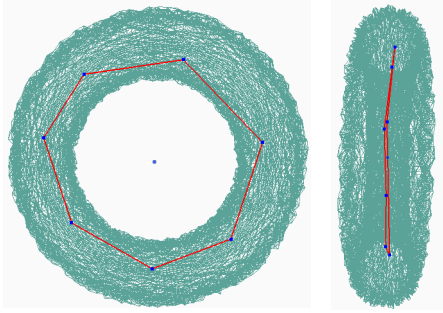
Malla	Parámetro δ	Error
$toroide_1$	0	4.137
$toroide_2$	$\frac{r}{2}$	4,463
$toroide_3$	r	4,519

Cuadro 6.5: Parámetro δ de la ecuación de ruido gausiano utilizado en las mallas toroidales.



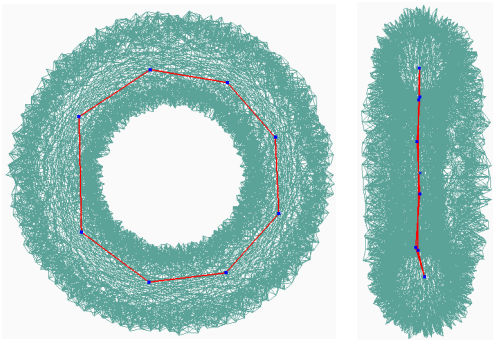
(a) (b)

Figura 6.1: Vistas del *skeleton* de la malla de superficie correspondiente al $toroide_1$ generada con ruido aditivo de una distribución uniforme de parámetro $\delta = 0$ (Ecución (6.1)).



(a) (b)

Figura 6.2: Vistas del *skeleton* de la malla de superficie correspondiente al $toroide_2$ generada con ruido aditivo de una distribución uniforme de parámetro $\delta = \frac{r}{2}$ (Ecución (6.1)).



(a) (b)

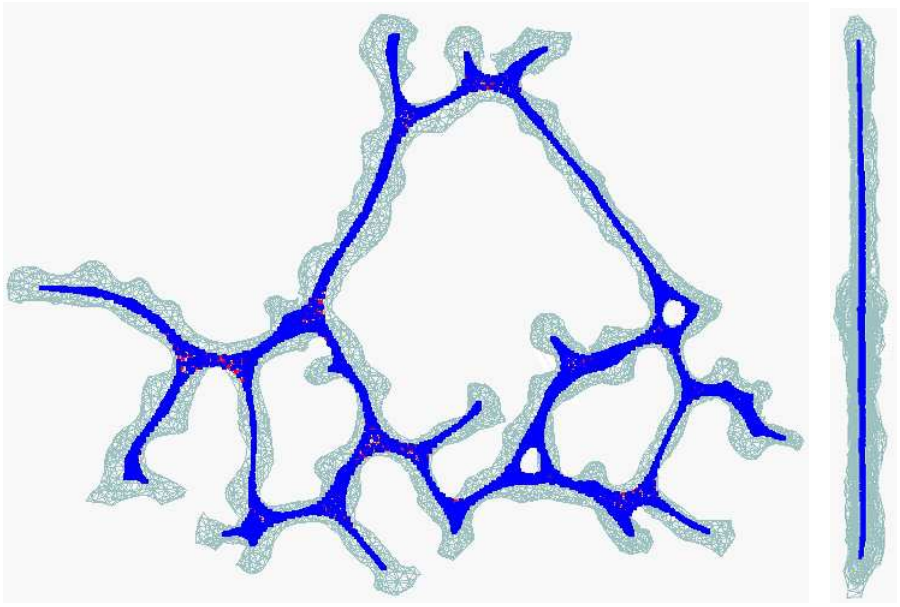
Figura 6.3: Vistas del *skeleton* de la malla de superficie correspondiente al $toroide_3$ generada con ruido aditivo de una distribución uniforme de parámetro $\delta = r$ (Ecuación (6.1)).

En el Cuadro (6.5) se observa que al aumentar el parámetro δ en la definición de ruido aditivo (Ecuación (6.2)), el error de el *skeleton* obtenidos comparados con el *skeleton* perfecto también aumenta, lo que da cuenta de la sensibilidad del método implementado ante pertubaciones locales en la superficie de la malla.

2. Se compararon visualmente los resultados obtenidos en este trabajo de memoria con la salida de un programa de la implementación original [2][3].

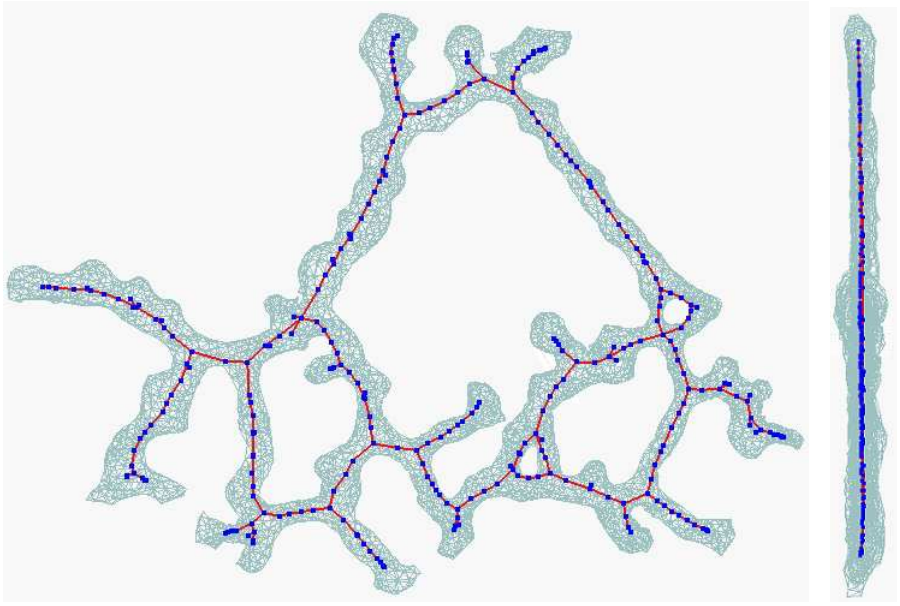
A continuación se muestran las salidas de las tres etapas del algoritmo (incluido el *skeleton* final) y el *skeleton* obtenido con una demo de la implementación de [2]:

[3]Disponible en http://visgraph.cse.ust.hk/projects/skeleton/



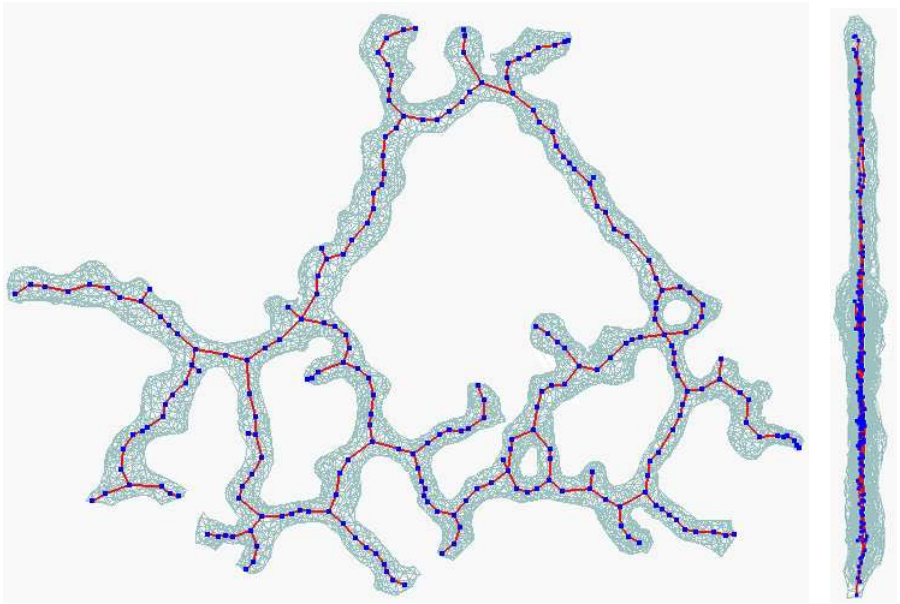
(a) (b)

Figura 6.4: Vistas de la malla de superficie de la región de *retículo* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



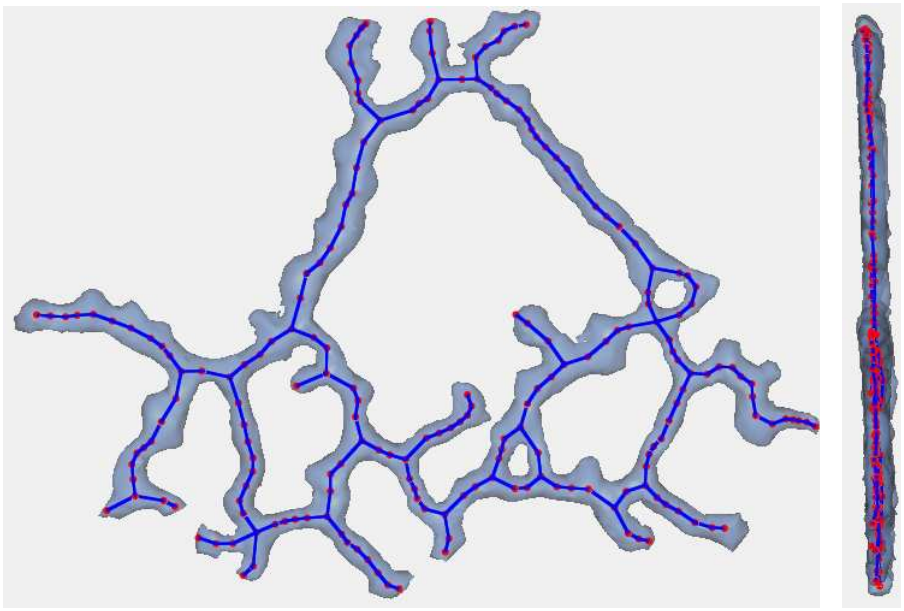
(a) (b)

Figura 6.5: Vistas del *skeleton* no centrado de la malla de superficie de la región de *retículo*.



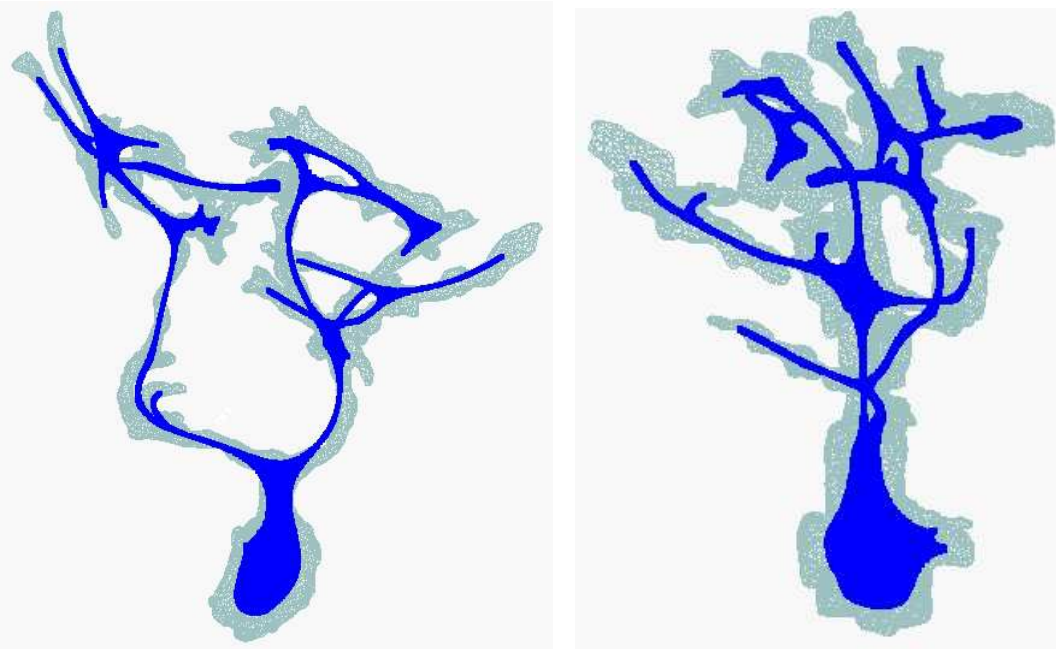
(a) (b)

Figura 6.6: Vistas del *skeleton* de la malla de superficie de la región de *retículo*.



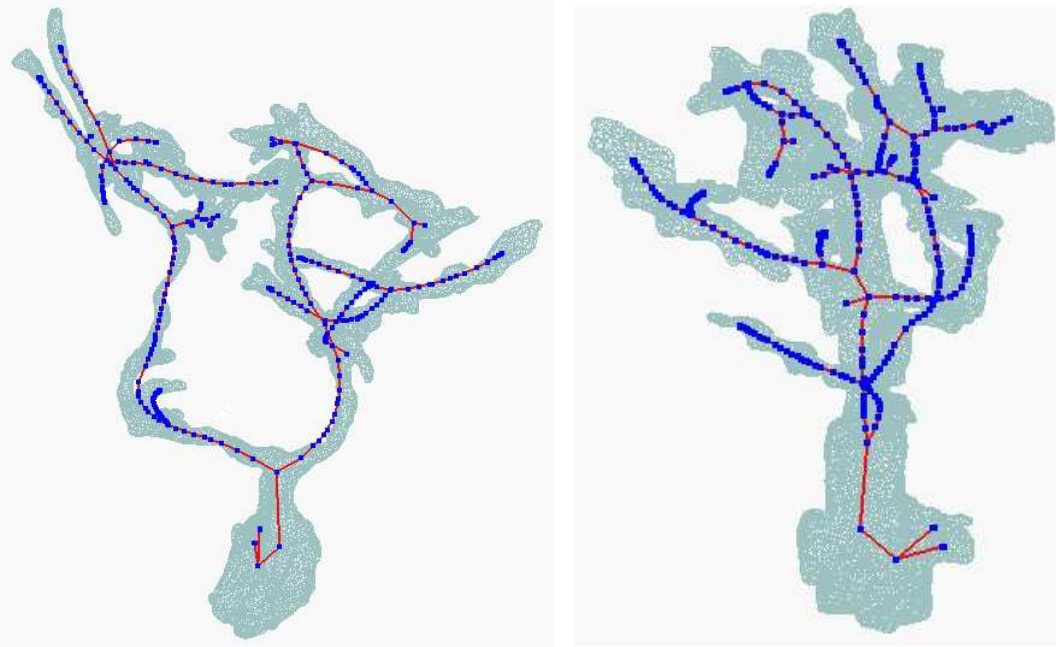
(a) (b)

Figura 6.7: Vistas del *skeleton* de la malla de superficie de la región de *retículo* generado con el demo de [2].



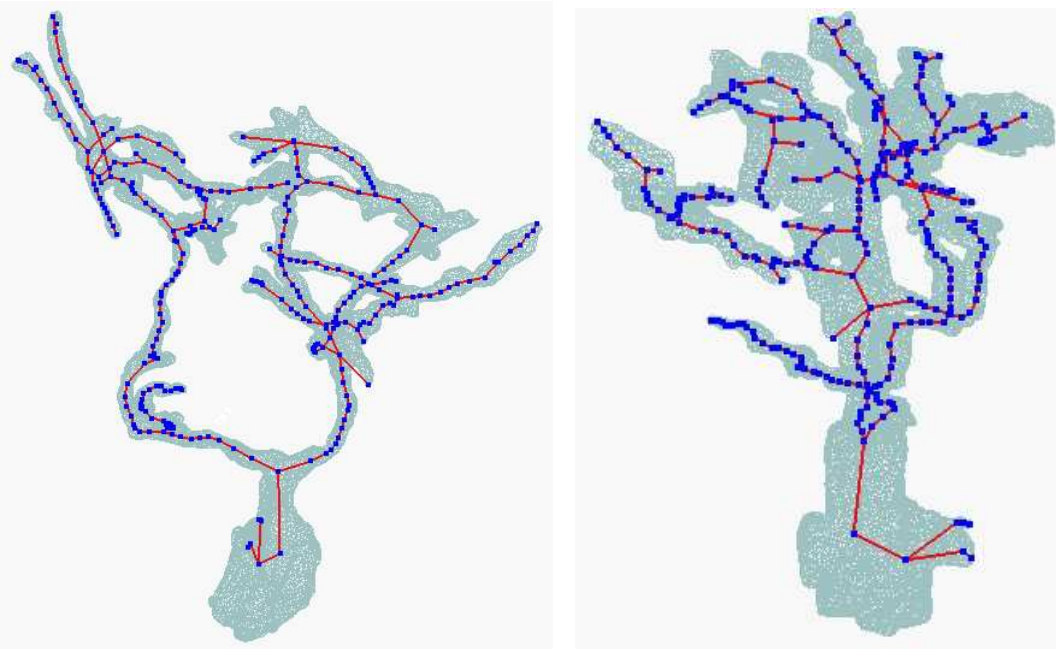
(a) (b)

Figura 6.8: Vistas de la malla de superficie de la *neurona* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



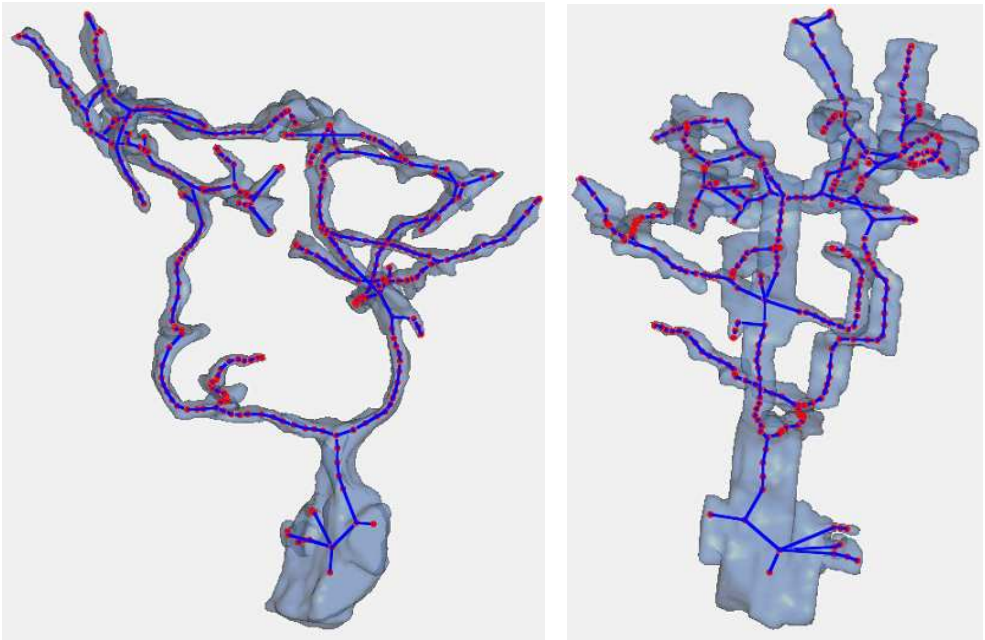
(a) (b)

Figura 6.9: Vistas del *skeleton* no centrado de la malla de superficie de la *neurona*.



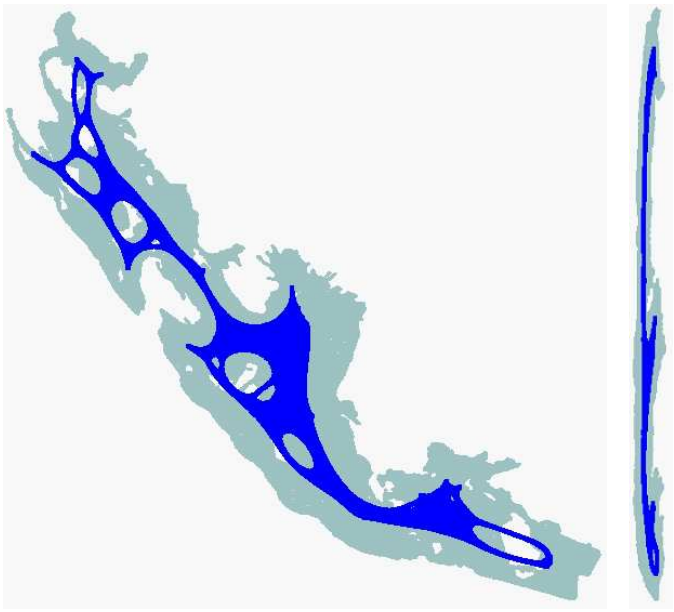
(a) (b)

Figura 6.10: Vistas del *skeleton* de la malla de superficie de la *neurona*.



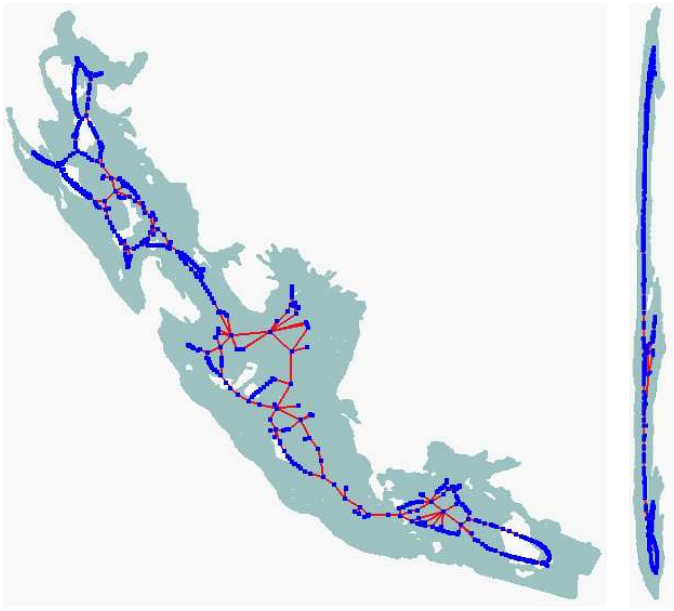
(a) (b)

Figura 6.11: Vistas del *skeleton* de la malla de superficie de la *neurona* generado con el demo de [2].



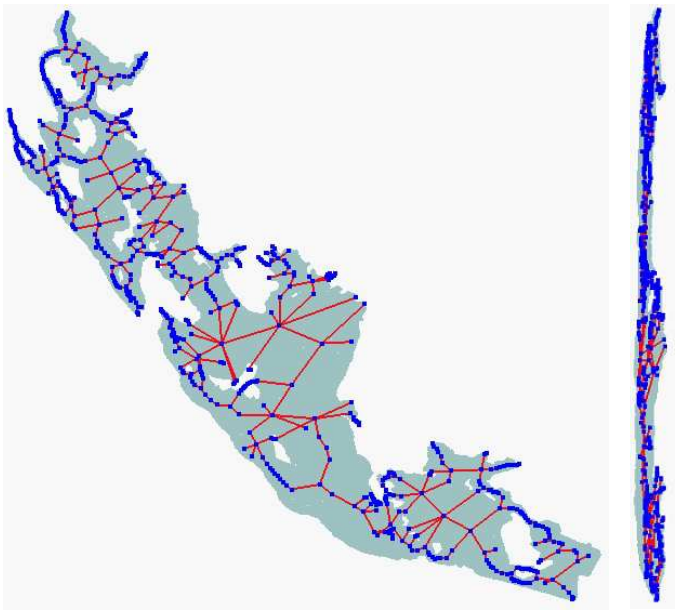
(a) (b)

Figura 6.12: Vistas de la malla de superficie de la *cresta neural* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



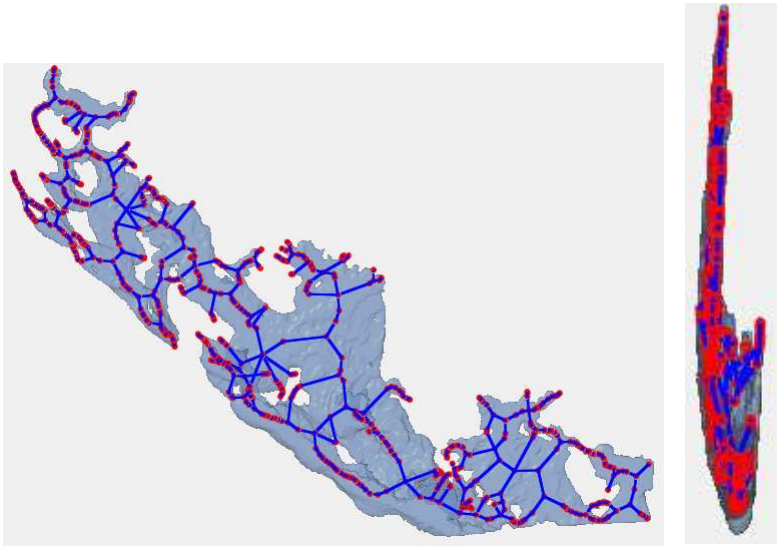
(a) (b)

Figura 6.13: Vistas del *skeleton* no centrado de la malla de superficie de la *cresta neural*.



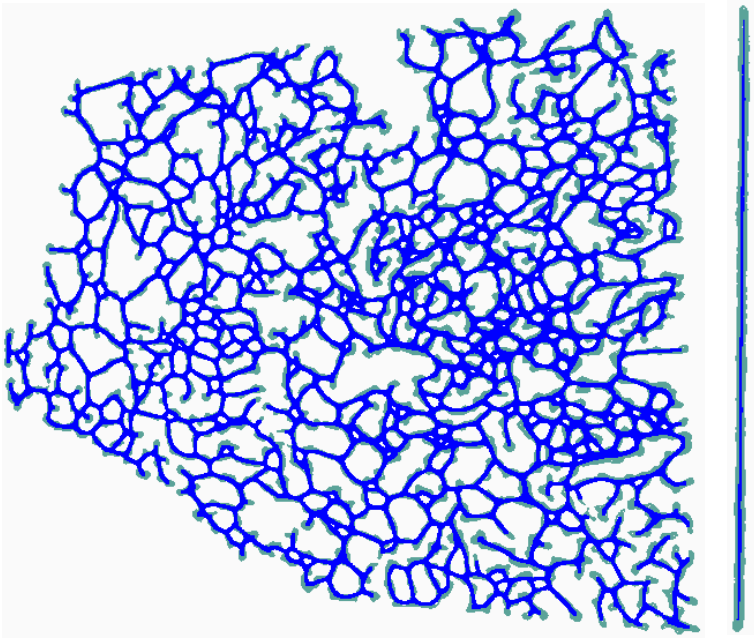
(a) (b)

Figura 6.14: Vistas del *skeleton* de la malla de superficie de la *cresta neural*.



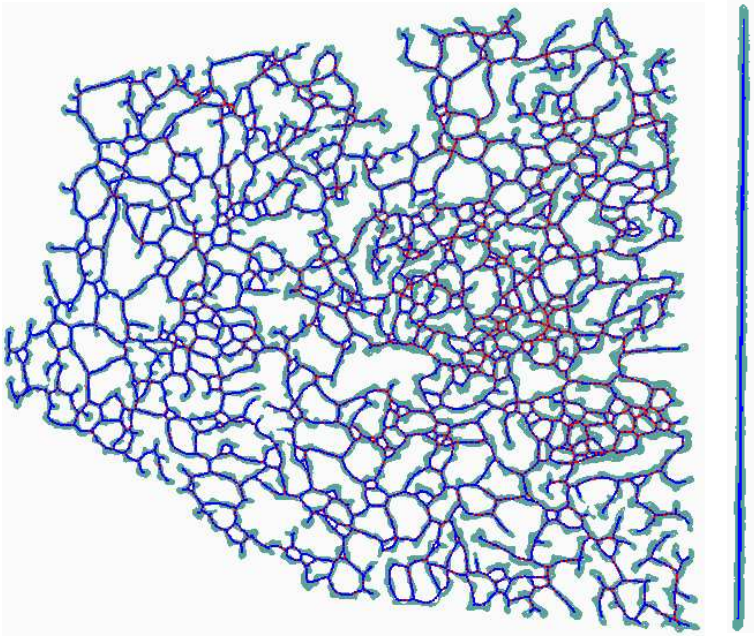
(a) (b)

Figura 6.15: Vistas del *skeleton* de la malla de superficie de la *cresta neural* generado con el demo de [2].



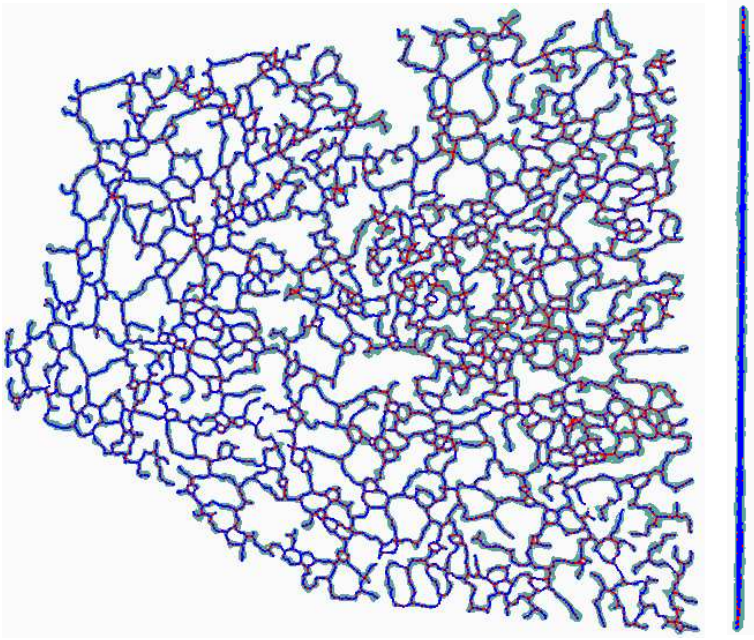
(a) (b)

Figura 6.16: Vistas de la malla de superficie de *retículo* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



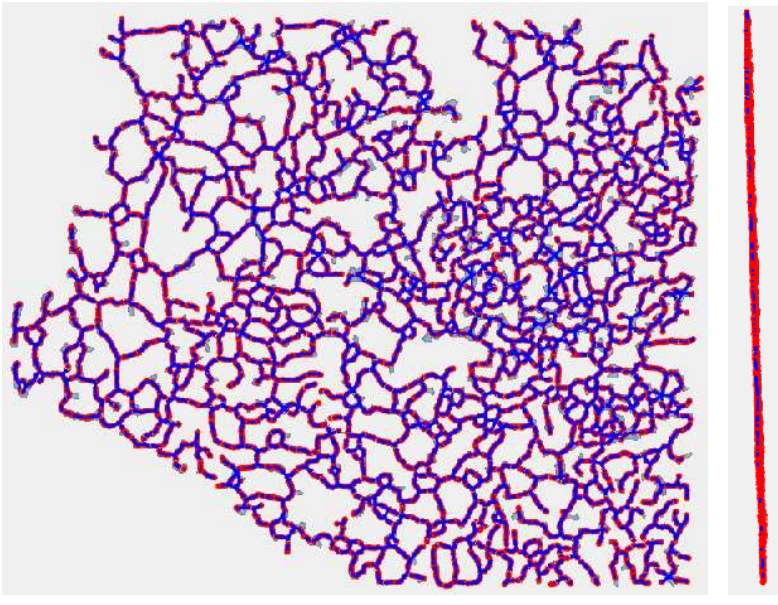
(a) (b)

Figura 6.17: Vistas del *skeleton* no centrado de la malla de superficie de *retículo*.



(a) (b)

Figura 6.18: Vistas del *skeleton* de la malla de superficie de *retículo*.



(a) (b)

Figura 6.19: Vistas del *skeleton* de la malla de superficie de *retículo* generado con el demo de [2].

En la Figura 6.4 se puede observar que las regiones de la malla contraída de la región de *retículo* no se encuentran fuera de la malla con demasiada frecuencia. Lo mismo sucede con la malla contraída de la *neurona* (Figura 6.8). En cambio, la malla contraída de la *cresta neuronal* (Figura 6.12) es de menor tamaño que la malla original y además la frecuencia de zonas que se ubican fuera de ésta.

Al comparar los *skeletons* generados con la implementación propuesta (Figuras 6.6, 6.10 y 6.14) con los obtenidos con el demo de [2] (Figuras 6.7, 6.11 y 6.15), se observa que ambos son muy parecidos. Por otro lado, los *skeletons* generados con la implementación realizada cumplen satisfactoriamente con características deseables [7] para un *skeleton*: ser homeotópico, unidimensional, aproximadamente centrado, robusto y suave.

Para comparar la densidad de nodos de los *skeletons* generados con las dos propuestas, se obtuvo el radio N° de nodos del *skeleton*/N° de vértices de la malla:

Malla	Implementación Actual	Demo
región del *retículo*	0.0419	0.0360
neurona	0.0213	0.0248
cresta neural	0.0078	0.0086
retículo	0.0796	0.0669

Cuadro 6.6: Radio N° de nodos del *skeleton*/N° de vértices de la malla

En términos generales se observa que la mayoría de los *skeletons* obtenidos con la implementación realizada tiene menos nodos que los *skeletons* obtenidos con el demo de [2], aunque ocurren excepciones (Apéndice C.1, Cuadro 9.3).

Capítulo 7

Discusión

El método original empleado como base para el desarrollo de este proyecto se enmarca dentro del área de la geometría computacional y sólo fue probado, por sus autores, en mallas artificiales típicas de esta área. En el contexto de este trabajo tanto el algoritmo original como la implementación propuesta han sido analizados y evaluados teniendo en consideración mallas artificiales y mallas reales basadas en estructuras biológicas 3D observadas desde imágenes de miscrocopía confocal de fluorescencia.

7.1. Mallas de entrada

7.1.1. Conformidad

Las mallas de entradas deben ser conformes y suaves. Conforme quiere decir que la intersección de dos triángulos disjuntos es un vértice o un arco, es decir, los elementos de la mallas se encuentran conectados y no generan vacíos. Suave quiere decir que la curvatura debe ser continua a lo largo de la malla o debe mantenerse dentro de cierto margen ya que la malla es un modelo discreto.

7.1.2. Tamaño

El tamaño de las mallas de entrada, específicamente el tamaño relativo de los triángulos afecta directamente al *skeleton* final. Por otro lado, la evaluación se realizó comparando los resultados del algoritmo propuesto con la implementación dada en [2]. Esta implementación disponible como un ejecutable no soporta archivos muy grandes. Además, su código fuente no está disponible y sólo funciona en equipos con *Windows XP* de 32 bits. La versión desarrollada en este proyecto de título soporta mallas grandes (de más de 400.000 triángulos) aunque los tiempos de ejecución son altos[1].

A continuación se presenta el tamaño de las mallas representativas de estructuras biológicas[2]:

[1] El análisis de la complejidad de cada etapa del algortimo se encuentra en el Capítulo 5.2

[2] Para más detalles sobre el tamaño de las mallas generadas artificialmente, ver el Apéndice C.1, Cuadro 9.1.

Malla	N° de vértices	N° de triángulos	N° de arcos
región de *retículo*	5720	11460	17190
neurona	13508	27028	40542
cresta neural	84513	169218	253827
retículo	92452	280398	186928

Cuadro 7.1: N° de componentes de las mallas.

En el Apéndice B.1 se encuentra la distribución de los ángulos en las mallas de superficie; y el radio entre el arco de menor longitud y el arco de mayor longitud de cada triángulo de las mallas utilizadas.

7.2. Etapa *Geometry Contraction*

7.2.1. Desplazamientos

Para lograr contraer los vértices de la malla hasta que cumplan con los criterios de detención, se implementaron dos alternativas de desplazamiento: la posición promedio de los vecinos y el vector normal.

La posición promedio de los vecinos logra contraer (y suavizar) satisfactoriamente mallas de distinta morfología. La razón de esta característica es que los vecinos de cada vértice siempre se encuentran ubicados lo suficientemente cerca de éste, impidiendo que atravise la sección transversal de su respectiva región. Sin embargo, este método no asegura que las mallas contraídas siempre se encuentren centradas con respecto a la malla original. Con frecuencia, algunas regiones de la malla contraída se ubican fuera de la malla original (Apéndice A, Figuras 9.3, 9.7, 9.11 y 9.19).

El único caso de falla de este enfoque corresponde a la malla de una esfera con alta densidad de vértices, pues cada vértice se encuentra casi en el plano con sus vecinos, luego, el desplazamiento es cercano a cero. Dado que en el *SCIAN-lab* se utilizan mallas de superficie con formas muy distintas e irregulares, esto no representa un problema.

Por otra parte, el vector normal logra contraer de forma correcta las mallas de superficie simétricas y de forma regular, como en el caso de la *donuts* (Figura 5.3). Este método falla en el resto de las mallas utilizadas pues al contraer los vértices su posición final atraviesa la sección transversal de la región donde se encuentran, ubicandose fuera de la malla original (Figura 5.1). Detectar que un vértice atraviesa su respectiva región de la malla es difícil y actualmente el laboratorio no cuenta con la implementación de un criterio que detecte este caso. Es por esta razón que el método de contracción escogido es el basado en la posición promedio de los vecinos. Aún asi, se considera una tarea desafiante implementar este criterio de detección para la mejora de enfoques basados en la dirección normal.

No se observaron diferencias relevantes entre los resultados obtenidos al utilizar los vectores normales iniciales en toda la etapa de contracción, y recalcularlos en cada iteración. En el Apéndice B.2, la Figura 9.51 muestra la variación del ángulo que forma la normal de cada

iteración con la normal inicial de la malla *donuts*. En este gráfico se observa que el ángulo de variación de la normal es prácticamente nulo, sin embargo, lo correcto es utilizar el vector normal inicial de cada vértice durante toda la etapa de contracción, pues reduce el ruido local en el cambio de dirección de la normal con la consecuente reducción de problemas de centrado del *skeleton*.

Es importante mencionar las diferencias entre los *skeletons* de la malla *donuts* obtenidos con los dos métodos de contracción. En el caso de la contracción utilizando el vector normal (Figura 5.40 (a)) se obtiene un *skeleton* con mayor densidad de nodos y visualmente más centrado que en el *skeleton* obtenido con el promedio de la posición de los vecinos (Figura 5.39 (a)). Además, la región local de cada nodo es mucho más uniforme cuando el *skeleton* se contrae con el vector normal (Figura 5.40 (b)). En este caso, la región local Π_{n_i} de un nodo interno (*non junction*) n_i no se intersecta con sus regiones vecinas $\Pi_{n_{i-1}}$ y $\Pi_{n_{i+1}}$. Además, los vértices de cada región Π_{n_i} son los que se esperan que colapsen en n_i. Esto no ocurre en el caso del método que utiliza el promedio de la posición de los vecinos (Figura 5.39 (b)), pues aquí las regiones de cada nodo se encuentran insersectadas.

Como se puede observar, ambos métodos presentan distintas ventajas y desventajas. Luego, es importante escoger el método de contracción más adecuado para los tipos de mallas con que se trabajan pues el *skeleton* final depende en gran medida de esta elección.

7.2.2. Criterios de detención

Los criterios de detención mencionados en [2] son complejos, es por esta razón que se diseñaron criterios generales a todas las mallas de superficie.

En términos generales, se observa que el funcionamiento del criterio de detención basado en el área de la malla es correcto, a pesar de su simplicidad.

En la implementación se agregó un criterio de detención basado en el volumen de las mallas de superficie. Este criterio todavía no se utiliza, pues no se encuentra implementado el método que obtiene el volumen de las mallas. Una forma de calcular este valor es aproximar cada región de la malla con un cilindro, de tal forma que el volumen de el cilindro sea lo más cercano posible al volumen de su respectiva región. Como se puede apreciar, obtener este valor es complejo y costoso pues depende de la morfología de las mallas. Un posible trabajo futuro consiste en la implementación de este método, para poder comparar el funcionamiento de los dos criterios basados en estas caraterísticas de las mallas.

Finalmente, se considera importante la introducción del criterio que detecta cuando el desplazamiento promedio de los vértices de la mallas es cercano a cero. Esto permite detener la etapa de contracción en un momento adecuado; de lo contrario, las mallas pueden continuar en el proceso de contracción cuando en realidad, permanecen en la misma posición. Luego, el volumen y el área de la malla no se reducen y no es posible cumplir la detención basada en estos criterios.

7.2.3. Posición de los nodos

En las mallas utilizadas (tanto las biológicas como las de fantasía) es frecuente que la malla contraída no se encuentre centrada con respecto a la malla original. En las mallas biológicas (Figuras 6.4, 6.8 y 6.12) esto no ocurre con demasiada frecuencia, pero en las mallas de fantasía se observa que las mallas contraídas presentan un tamaño mucho menor al de la malla original, hasta encontrarse completamente fuera de ésta utilizando el método de contracción escogido (Apéndice A, Figuras 9.3, 9.7, 9.11, 9.15 y 9.19).

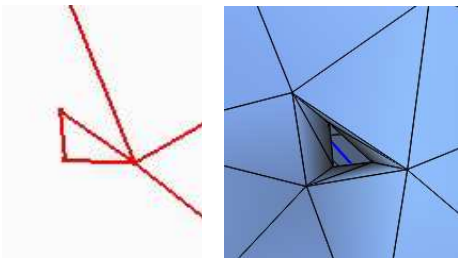
Para solucionar este defecto, una posibilidad es intentar implementar la propuesta de [2] para la etapa de contracción (Capítulo 2.3). Para evitar problemas de precisión [23] al calcular los ángulos $\alpha_{i,j}$ y $\beta_{i,j}$ que define $e_{(i,j)}$, se sugiere revisar la calidad de los triángulos de la malla y aplicar la triangulación de *Delaunay* [8] en el caso que ésta sea deficiente. Los detalles sobre la calidad de las mallas biológicas y de fantasía se encuentran en el los Apéndices B.2 y B.3.

7.3. Etapa *Connectivity Surgery*

7.3.1. Condición de colapso

Si la malla de entrada es conforme, entonces la implementación garantiza que en términos estructurales el *skeleton* resultante no posee triángulos. Un caso interesante ocurre cuando mallas conformes poseen hendiduras, pues en este escenario no se puede garantizar la ausencia de triángulos residuales en términos visuales y de conectividad.

Un ejemplo de esto se observa en la malla de superficie de la *neuropila*. Esta malla es conforme, pero presenta una hendidura con entrada de forma triangular. La Figura 7.1 (a) muestra un triángulo virtual del *skeleton* conservado por la implementación, esto es, un conjunto de arcos que tienen la conectividad de un triángulo, pero que estructuralmente no existe en la malla. En la Figura 7.1 (b) se muestra el *skeleton* generado por el demo de [2]. Se puede observar claramente que [2] no cumple con la condición de colapso que el método dice utilizar (Figura 4.4), pues de lo contrario, los arcos que forman la hendidura no se habrían colapsado.



(a) (b)

Figura 7.1: (a) Triángulo del *skeleton* de la malla de superficie de la *neuropila* generado a causa de la hendidura. (b) Hendidura de la malla de superficie de la *neuropila* y *skeleton* de esta malla producido por el demo de [2] en azul.

La condición de colapso sugerida en [2] es la siguiente: si existen tres vértices adyacentes $(i,\ j,\ k)$, pero el triángulo $t_{(i,j,k)}$ no existe, entonces se prohíbe el colapso $(i \rightarrow j)$ (Figura 4.4). Mediante el análisis de casos prácticos (Figura 7.2) se determinó que esta condición es insuficiente, por lo que fue reformulada en base al siguiente criterio: si alguno de los dos triángulos incidentes en $e_{(i,j)}$ es nulo, entonces se prohíbe el colapso $(i \rightarrow j)$.

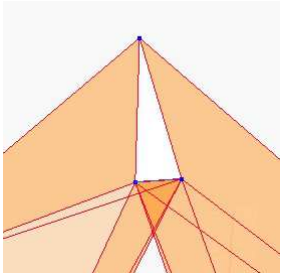


Figura 7.2: Condición de colapso sugerida en [2]. El arco en azul tiene sus respectivos triángulos incidentes, sin embargo, la condición mencionada en [2] impide colapsarlo.

En comparación con el criterio original, el criterio propuesto no permite el colapso de arcos que forman parte de un triángulo virtual.

7.3.2. Posición de los nodos del *skeleton*

Tanto en las mallas biológicas (Figuras 6.5, 6.9 y 6.13) como en las mallas generadas artificialmente (Apéndice A, Figuras 9.4, 9.8, 9.8, 9.12, 9.16, 9.20) se observa que el *skeleton* producido en esta etapa, más que estar centrado respecto de la malla original, se encuentra centrado con respecto a la malla contraída (*input* de esta etapa). Esto no se considera un problema significativo, pues la etapa de centrado del *skeleton* lo corrige satisfactoriamente.

7.3.3. Ruido y Representatividad

La etapa de colapso de arcos genera *skeletons* que presentan ruido inherente al algoritmo [2]. Esto se puede observar en el Apéndice A, Figuras 9.8, 9.12 y 9.20. Al remover el ruido utilizando el criterio de representitividad de los nodos del *skeleton*, se obtienen resultados más limpios y acordes a la morfología y topología de la malla.

La siguiente Tabla muestra la diferencia entre el número de nodos de los *skeletons* de las mallas biológicas[3] con y sin corrección de ruido:

[3]Para más detalles sobre la diferencia entre el número de nodos de los *skeletons* de las mallas generadas artificialmente con y sin ruido ver el Apéndice C.1, Cuadro 7.2

Malla	N° de nodos del *skeleton* con ruido	N° de nodos del *skeleton* final
región de *retículo*	278	240
neurona	298	288
cresta neural	708	661
retículo	7573	7364

Cuadro 7.2: Comparación entre el número de nodos de *skeleton* finales con y sin ruido.

7.4. Etapa *Embedding Refinement*

7.4.1. Centrado y Simetría

En los *skeletons* obtenidos de las mallas biológicas y en las mallas generadas artificialmente, se observa una buena correspondencia malla-*skeleton* (Figuras 6.6, 6.10 y 6.14 y Anexo A, Figuras 9.5, 9.9, 9.13, 9.17 y 9.21). Además, cumplen con características deseables, como: ser aproximadamente centrado, homeotópico, unidimensional, robusto y suave.

También se puede observar que el hecho que la malla de entrada sea simétrica no implica que el *skeleton* resultante sea simétrico. Un ejemplo de esto se observa en la Figura 9.9 que muestra un *skeleton* asimétrico. Aquí el *skeleton* de la región superior de la malla no es simétrico ni tampoco es aproximadamente centrado [7]. Una forma de corregir esto [2] es combinar un *junction node* (nodo de bifurcación) con un nodo adyacente si el nodo resultante tiene un mejor centrado que el nodo de bifurcación original. El centrado de un nodo de bifurcación n_k es medido como la desviación estándar de la distancia de la posición de los nodos $n_i \in \Pi_k$, denotada σ_k. Luego, se fusiona el nodo n_k con su vecino si $\sigma_k^{'} < \sigma_k$, donde $\sigma_k^{'}$ es el centrado del nodo resultante ($n_k^{'}$).

A pesar que el *input* de esta etapa es un *skeleton* centrado en la malla contraída, este método asegura que todos los nodos del *skeleton* se encuentran dentro de la malla inicial. No obstante, no garantiza que todos los segmentos del *skeleton* se encuentren dentro de la malla original. Esto sucede por la baja densidad de nodos que presentan algunas regiones del *skeleton*. Un ejemplo de esto se puede observar en el Apéndice A, Figura 9.22. Una forma de solucionar este problema consiste en aumentar la densidad de nodos en las zonas del *skeleton* donde los segmentos no se encuentran dentro de la malla.

7.4.2. Densidad de nodos

En los *skeletons* obtenidos de las mallas biológicas y en las mallas generadas artificialmente, se observa que las regiones de la malla que presentan mayor densidad de triángulos producen mayor densidad de nodos del *skeleton* final. La Figura 9.17 del Apéndice A, muestra el *skeleton* de la malla de superficie *frog*. Se observa que los dedos de los pies y de las manos de la malla presentan alta densidad de nodos del *skeleton*, debido a que estas regiones poseen una alta densidad de triángulos de menor tamaño.

7.5. Rendimiento

El rendimiento del algoritmo implementado ha sido analizado en función de número de operaciones requeridas por cada una de las etapas, con lo cual se tiene una estimación de los tiempos de cómputo empleados. En relación al los recursos utilizados en memoria, se analizaron el tamaño máximo del *heap*[4] y el espacio máximo utilizado.

La etapa de contracción de vértices (*geometry contraction*) es $\mathcal{O}(nn_1)$, donde n es el número de vértices de la malla y donde n_1 es el número de arcos que inciden en cada vértice.

En términos del tiempo de procesamiento, la implementación se encuentra dominada por la etapa de colapso de arcos (*connectivity surgery*), que es $\mathcal{O}(n_1 n_3^2 log(n))$, donde n_3 es el número de nodos en la vecindad del vértice j[5]. La operación que más pesa en esta etapa es la revisión y corrección de la vecindad j, que es $\mathcal{O}(n_1 n_3^2)$.

El costo de la etapa de centrado (*embedding refinement*) es $\mathcal{O}(\overline{n})$, donde $\overline{n}$ es el número de nodos del *skeleton* resultante.

Etapa	N° de operaciones
geometry contraction	$\mathcal{O}(nn_1)$
connectivity surgery	$\mathcal{O}(n_1 n_3^2 log(n))$
embedding refinement	$\mathcal{O}(\overline{n})$

Cuadro 7.3: Complejidad de las etapas del algoritmo implementado.

Para medir el tiempo de procesamiento, se utilizó el método *getCurrentThreadUserTime* de la clase *ThreadMXBean*[6] que provee la API de *Java*. Este método retorna el tiempo CPU que el *thread*[7] actual ha ejecutado en modo usuario.

A continuación, se muestra el detalle de los tiempos de procesamiento de la implementación en *Java* para las mallas biológicas[8]:

[4]En el contexto del manejo de memoria y la asignación de memoria dinámica, usualmente la memoria es asignada a partir de un *pool* de memoria no usada llamada *heap*. Para más detalles sobre el manejo de la memoria y asignación de memoria dinámica, ver [26] [16].

[5]La vecindad el vértice j corresponde al *three-ring* [13] de este vértice. Esto corresponde a todos los arcos y triángulos que se encuentran a (máximo) tres arcos de distancia desde j.

[6]http://docs.oracle.com/javase/6/docs/api/java/lang/management/ThreadMXBean.html

[7]Un *thread* o hilo de ejecución es la unidad básica de procesamiento que maneja el sistema operativo [24].

[8]Para más detalles del tiempo de procesamiento de las mallas de fantasía, ver el Apéndice C.2, Tabla 9.4.

Malla	N° de vértices	Tiempo de procesamiento (hh:mm:ss)		
		geometry contraction	*connectivity surgery*	*embedding refinement*
región de *retículo*	5720	00:00:00,10	00:00:57,20	00:00:00,03
neurona	13508	00:00:01,60	00:08:27,80	00:00:00,08
cresta neural	84513	00:06:16,60	17:26:48,60	00:00:00,16
retículo	92452	00:00:06,75	102:09:00,00	00:00:02,48

Cuadro 7.4: Tiempo de procesamiento del algoritmo implementado.

Como se puede observar, aunque el algoritmo implementado es correcto, el tiempo de ejecución es alto porque el orden de etapa de colapso de arcos (en particular, la revisión y corrección de la malla) lo es y los valores de n_1 y n_3 en la práctica son mayores que 6 y 30, respectivamente. En el caso que el tiempo de procesamiento sea un requerimiento de alta prioridad, se sugiere optimizar el rendimiento de la implementación paralelizándola mediante el uso de *threads*[9].

Con respecto a los recursos utilizados en memoria durante la ejecución del algoritmo, el tamaño máximo del *heap* y el espacio máximo utilizado se muestran en la siguiente tabla:

Malla	N° de vértices	Tamaño máximo del heap (MB)	Espacio máximo utilizado (MB)
región de *retículo*	5720	693	403
neuropila	13508	1119	846
cresta neural	84513	2212	1981
retículo	92452	3129	2577

Cuadro 7.5: Tamaño máximo del *heap* y espacio máximo utilizado durante el procesamiento de las mallas biológicas.

[9]http://docs.oracle.com/javase/6/docs/api/java/lang/Thread.html

Capítulo 8

Conclusiones

En este trabajo de memoria se implementó un algoritmo para la generación de *skeletons* 1D de objetos 3D a partir de mallas triangulares de superficie, basado en un método descrito en [2]. La implementación del algoritmo corresponde a las etapas que lo componen; estas son: contracción de la malla (*geometry contraction*), colapso de arcos (*connectivity surgery*) y corrección del centrado *skeleton* (*embedding refinement*), además de una verificación para que las mallas de entrada sean conformes.

El algoritmo debe garantizar que los *skeletons* satisfagan los requerimientos de: unidimensionalidad, homeotópico, invariante bajo transformaciones isométricas, centrado, robusto y suave para el análisis de morfo-topología en modelos geométricos de estructuras biológicas tridimensionales observadas en imágenes de microscopía de fluorescencia con resolución nanométrica.

Para la etapa de contracción se desarrollaron tres métodos que proporcionan un mecanismo robusto para el tratamiento de diversos tipos de mallas de entrada, considerando características como grandes cambios en la curvatura local de la superficie y zonas de baja curvatura con gran densidad local (alta concentración de vértices). Mediante el ánalisis del decaimiento del área de las mallas durante la etapa de contracción, se determinó que iterando hasta alcanzar un 15 % del área inicial, se obtienen resultados satisfactorios en todos los casos. Para la etapa de contracción se implementó un criterio de detención adicional que permite forzar el término de esta etapa cuando el desplazamiento promedio de la malla se encuentra bajo un umbral definido. Este criterio entrega resultados apropiados para la diversidad de mallas estudiadas en el contexto del *SCIAN-lab*.

Para la etapa de colapso de arcos se implementaron las siguientes mejoras: una definición robusta y selección de arcos colapsables, la consideración de los distintos escenarios de colapso y la revisión de las vecindades del vértice colapasado. Como resultado de esta etapa se tiene un *skeleton* preliminar unidimensional y homeotópico, pero no necesariamente centrado y que puede contener ruido inherente al algoritmo. Por ruido se entiende la existencia de nodos terminales poco representativos de la malla original.

En la etapa de centrado de nodos del *skeleton*, con las mejoras implementadas se obtienen estructuras unidimensionales aproximadamente centradas que satisfacen las condiciones buscadas con la definición de *skeletons* acorde a los requerimientos establecidos en el contexto

biológico de este proyecto.

La validación del algoritmo se llevó a cabo usando mallas de estructuras biológicas reales de retículo endoplasmático en células de cultivo, y neuropila y crestas neurales en cerebros de embriones de pez cebra, obtenidas por microscopía de fluorescencia. También se emplearon mallas de fantasía generadas artificialmente y usadas comúnmente en geometría computacional, de distinta complejidad y tamaño. Las estructuras biológicas son altamente ramificadas, mientras que las de fantasía presentan formas muy diversas pero menos ramificadas. La aplicación exitosa en mallas compuestas de 2000 a 300000 triángulos aproximadamente, da cuenta de la escalabilidad del algoritmo.

Capítulo 9

Trabajo Futuro

La visualización del *skeleton* de estructuras biológicas reales se realizó mediante la integración de los métodos desarrollados en *Java* con la plataforma *SCIAN-Soft*, utilizando el lenguaje IDL.

Aún es necesario integrar a *SCIAN-Soft* la generación y reducción de mallas de superficie conformes, las que actualmente son realizadas mediante aplicaciones externas y que se encuentran fuera del alcance de este proyecto. En este aspecto, se considera interesante revisar la calidad de los triángulos de las mallas generadas en busca de una potencial retriangularización basada en *Delaunay* [8] que permita mejorar problemas de robustez del método de contracción propuesto por [2].

En relación al rendimiento y escalabilidad del algoritmo, si bien los tiempos de procesamiento alcanzados en los casos de prueba son satisfactorios para los requerimientos de *SCIAN-lab*, es posible paralelizar y optimizar los diversos métodos.

Referencias

[1] P. Aguilar, J. Jara, O. Ramirez, K. Palma, M. Concha, N. Hitschfeld, and S. Härtel. Skeletons and their application to quantify tree-like biological structures. In *Chilean Society for Cell Biology XXIV Annual Meeting*, page 142, November 2010.

[2] O. K.-C. Au, C.-L. Tai, H.-K. Chu, D. Cohen-Or, and T.-Y. Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, Volume 27:10 pages, August 2008.

[3] X. Bai and L. J. Latecki. Discrete skeleton evolution. In *Proceedings of the 6th international conference on Energy minimization methods in computer vision and pattern recognition*, EMMCVPR'07, pages 362–374, Berlin, Heidelberg, 2007. Springer-Verlag.

[4] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Proc. Models for the Perception of Speech and Visual Form*, pages 362–380, Cambridge, MA, November 1967. MIT Press.

[5] A. Bucksch, R. Lindenbergh, and M. Menenti. Skeltre: Robust skeleton extraction from imperfect point clouds. *Vis. Comput.*, 26:1283–1300, October 2010.

[6] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[7] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, Volume 13:pages 530–548, May 2007.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[9] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[10] R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Number February. Addison-Wesley, 1995.

[12] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[13] T. Gatzke and C. M. Grimm. Estimating curvature on triangular meshes. *International Journal of Shape Modeling*, 12(1):1–28, 2006.

[14] S. Härtel, J. Jara, C. Lemus, and M. Concha. *3D Morpho-Topological Analysis of Asymmetric Neuronal Morphogenesis in Developing Zebrafish*, pages 215–220. Taylor and Francis Group, 2007.

[15] A. E. Kaufman, D. Cohen-Or, and R. Yagel. Volume graphics. *IEEE Computer*, Vol. 26(No. 7):pages 51–64, 1993.

[16] D. E. Knuth. *Fundamental Algorithms*, chapter 1.2. Addison-Wesley, 1973.

[17] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[18] R. C. Martin. *Clean Code: A handbook of agile software craftsmanship*. Prentice Hall, 2009.

[19] C. Melo. Desarrollo de una herramienta que genera malla se superficie compuestas de cuadriláteros para modelar el crecimiento de árboles. Memoria para optar al título de Ingeniero Civil en Computación, 2008.

[20] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, New York, NY, USA, 2nd edition, 1998.

[21] O. Ramírez, S. Härtel, and A. Couve. Location matters: the endoplasmic reticulum and protein trafficking in dendrites. *Biological Research*, 44(1):17–23, 2011.

[22] J. Richardson, C. Figueroa, F. Santibañez, S. Hartel, and L. C. Notch signal pathway roles in neural crest migration in zebrafish. 2011.

[23] S. Schirra. Robustness and precision issues in geometric computation. Research Report MPI-I-98-1-004, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, January 1998.

[24] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley Publishing, 8th edition, 2008.

[25] J. Suarez, A. Plaza, and G. Carey. Diagrama geométrico y subdivisión híbrida de triángulos. *Revista Internacional de Métodos Numéricos para Cálculo y Diseño en Ingeniería*, 25(1):61–78, 2009.

[26] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles. Dynamic storage allocation: A survey and critical review. pages 1–116. Springer-Verlag, 1995.

Apéndice

A . Figuras

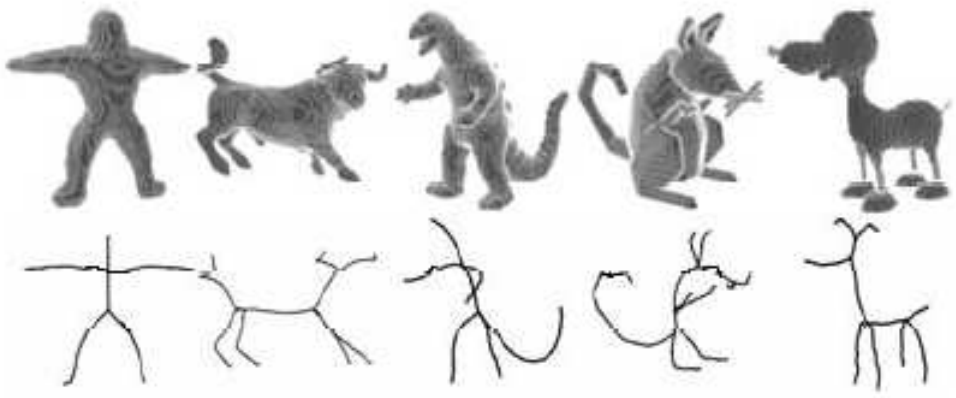
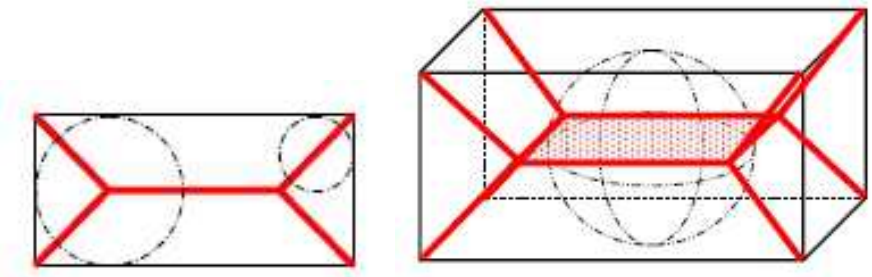
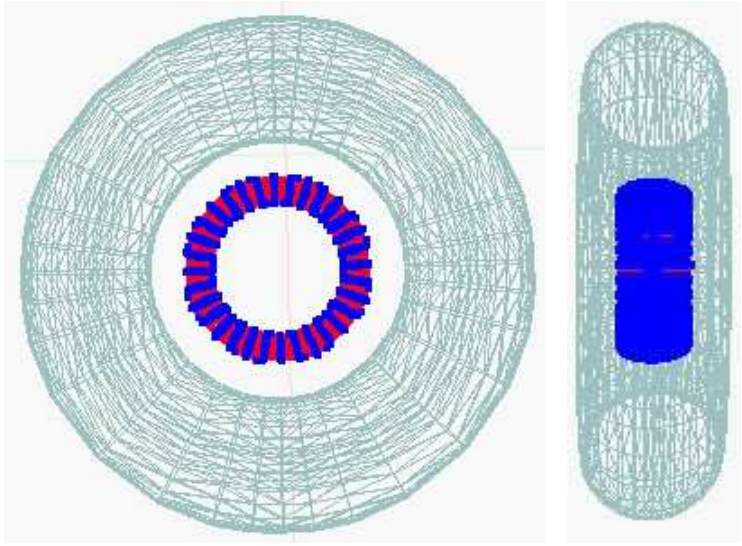


Figura 9.1: Posibles *skeletons* de diferentes objetos 3D [7].



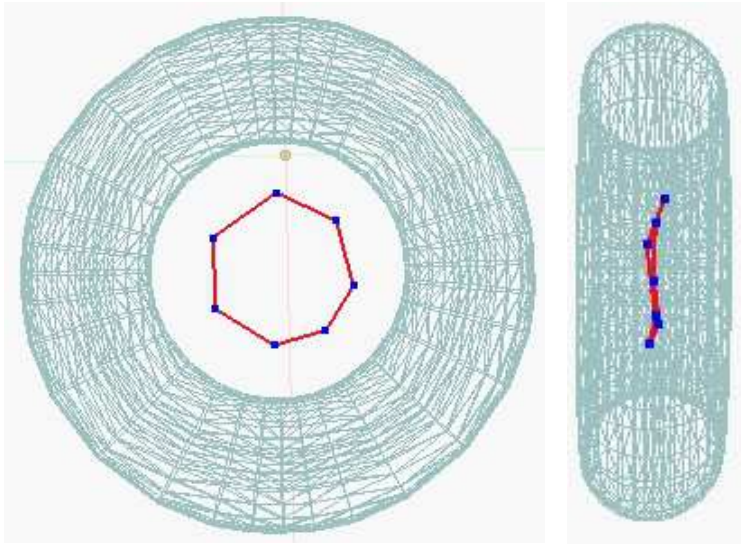
(a) (b)

Figura 9.2: (a) Eje medial y ejemplos de discos maximales inscritos en un objeto 2D [7]. (b) Superficie medial y el ejemplo de una bola maximal inscrita en un objeto 3D [7].



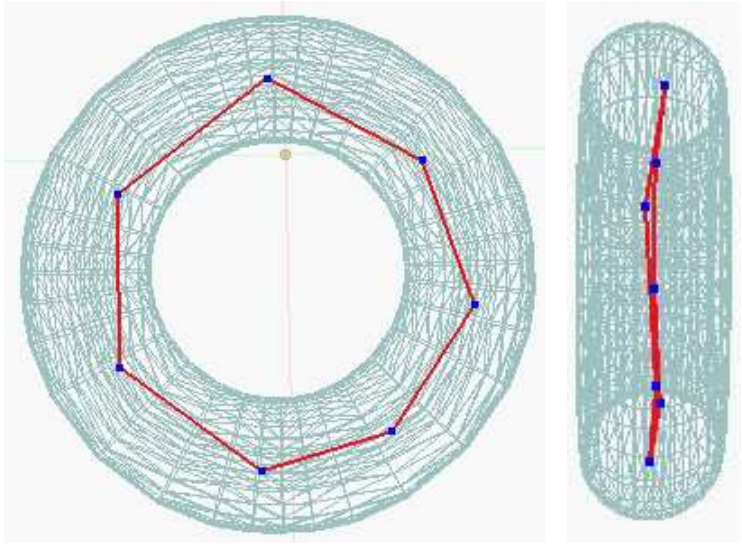
(a) (b)

Figura 9.3: Vistas de la malla de superficie *donuts* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



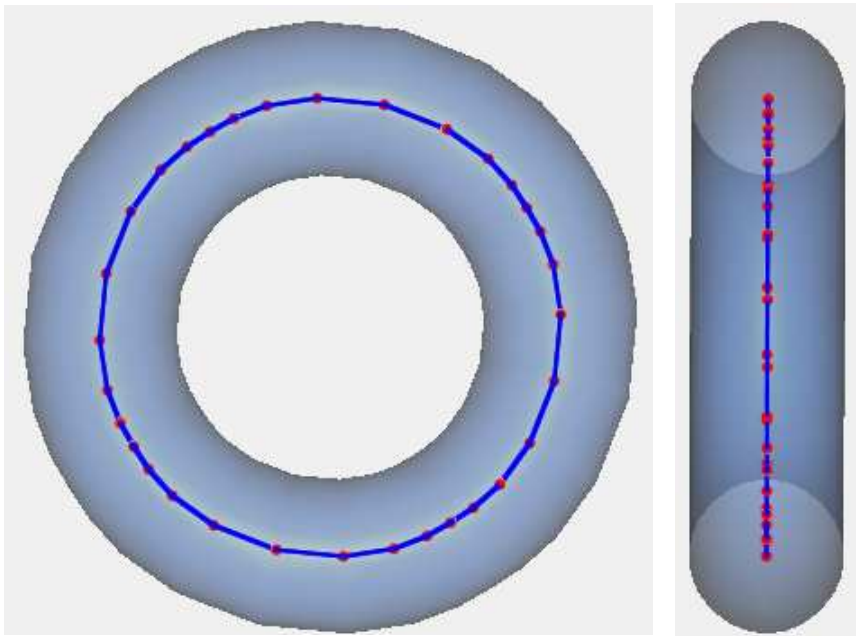
(a) (b)

Figura 9.4: Vistas del *skeleton* no centrado de la malla de superficie *donuts*.



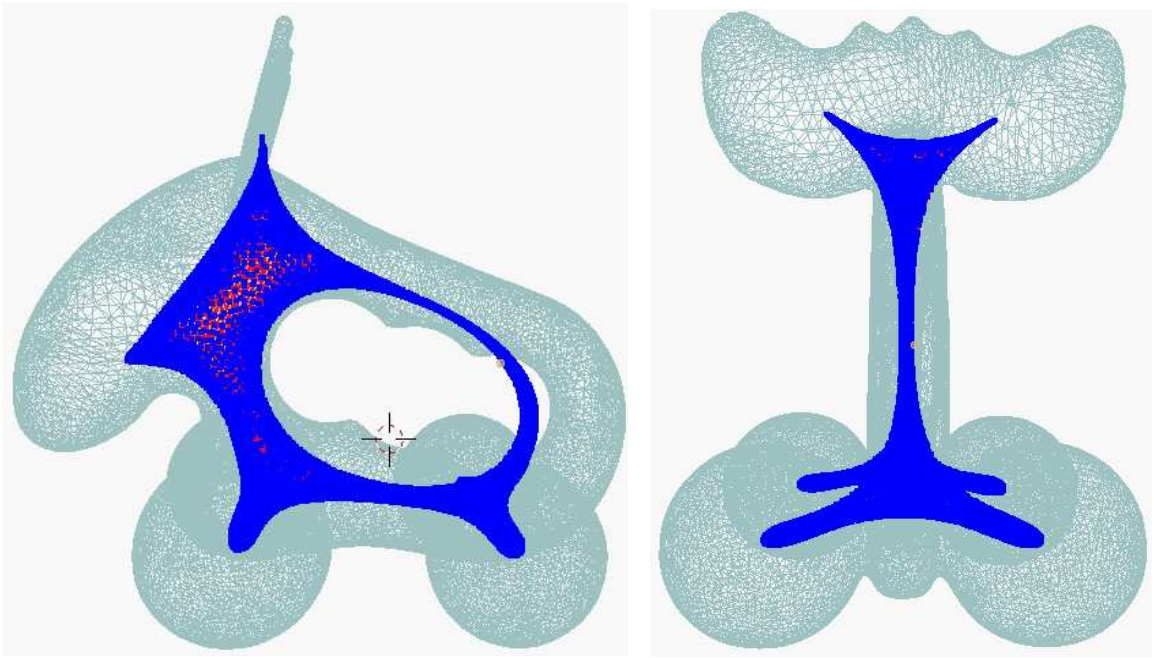
(a) (b)

Figura 9.5: Vistas del *skeleton* de la malla de superficie *donuts*.



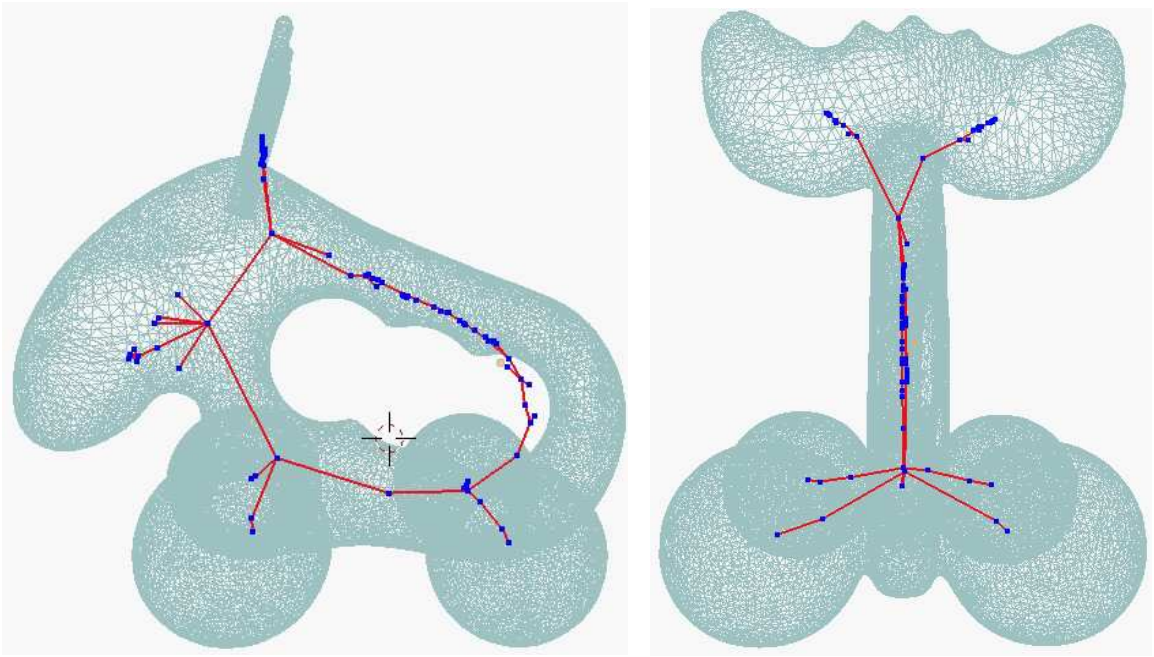
(a) (b)

Figura 9.6: Vistas del *skeleton* de la malla de superficie *donuts* generado con el demo de [2].



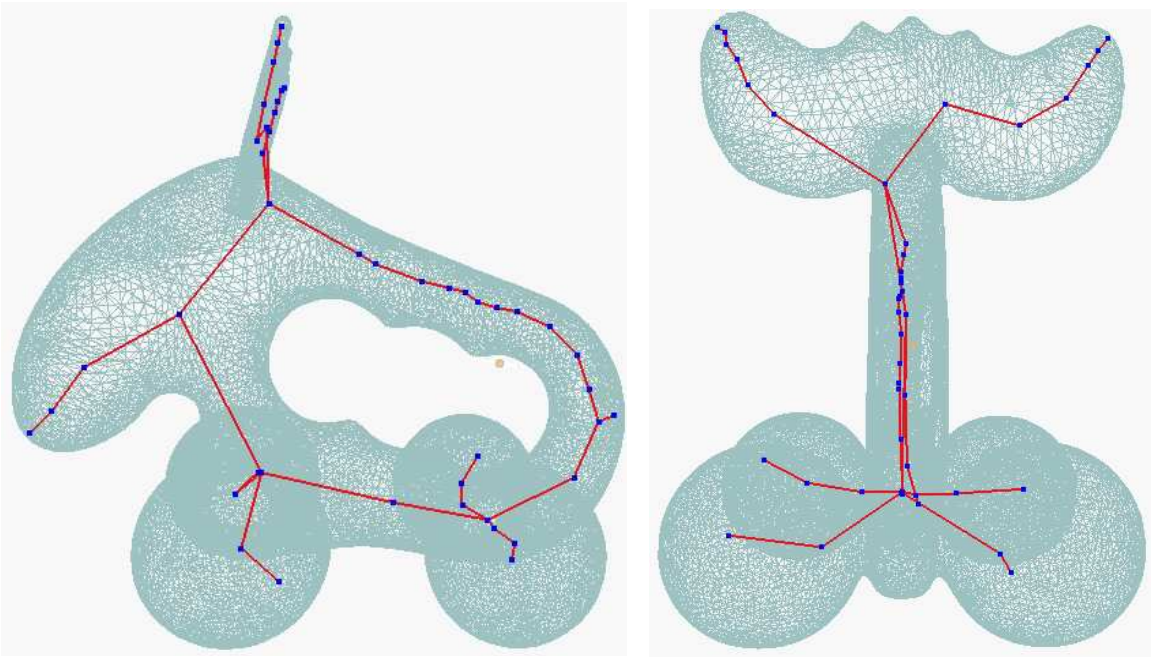
(a) (b)

Figura 9.7: Vistas de la malla de superficie *elk* contraída. La fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



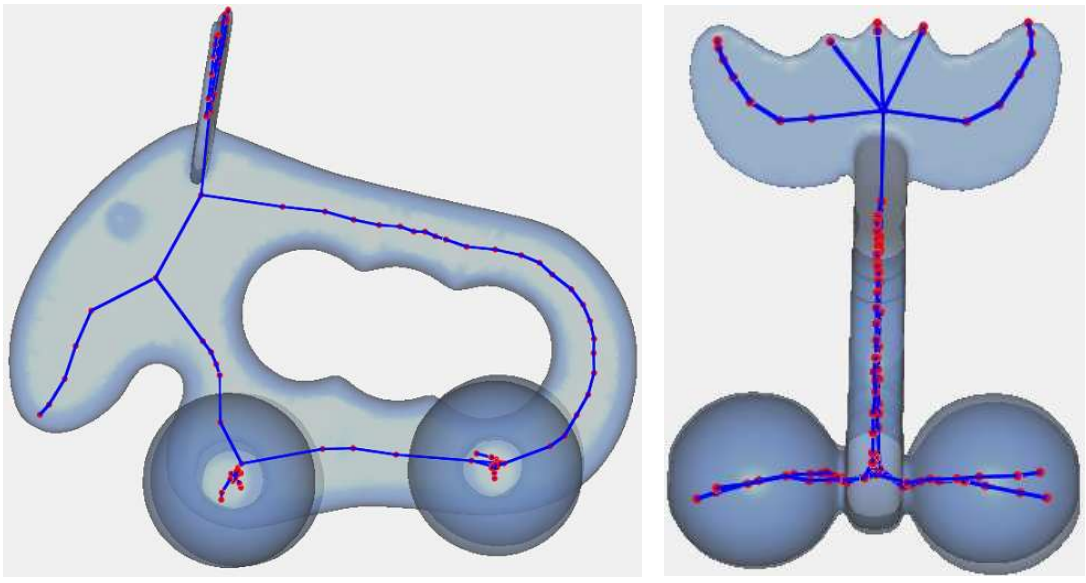
(a) (b)

Figura 9.8: Vistas del *skeleton* no centrado de la malla de superficie *elk*.



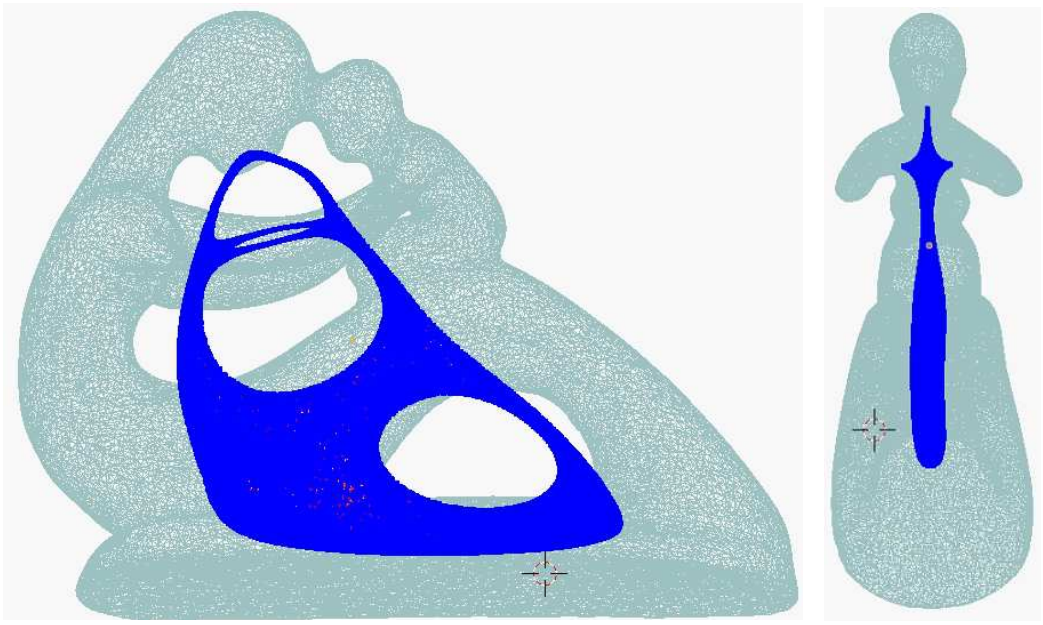
(a) (b)

Figura 9.9: Vistas del *skeleton* de la malla de superficie *elk*.



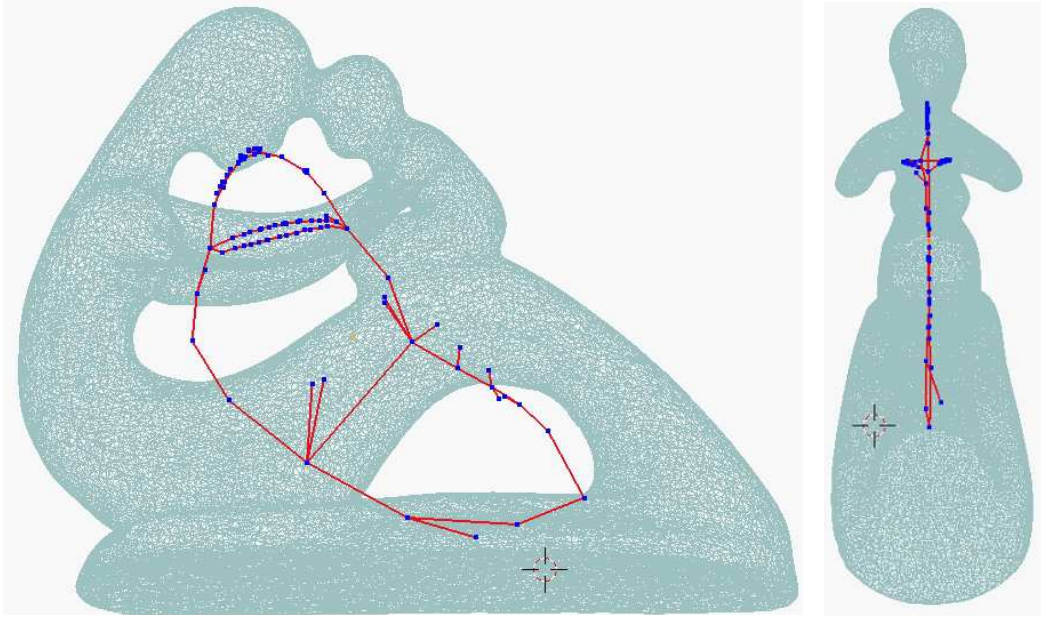
(a) (b)

Figura 9.10: Vistas del *skeleton* de la malla de superficie *elk* generado con el demo de [2].



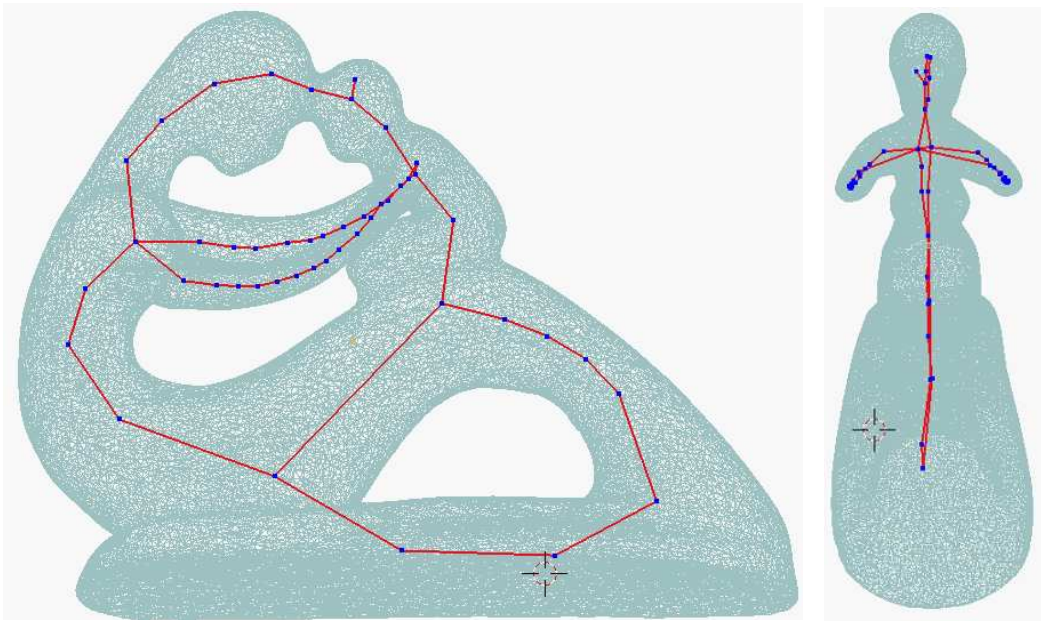
(a) (b)

Figura 9.11: Vistas de la malla de superficie *fertility* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



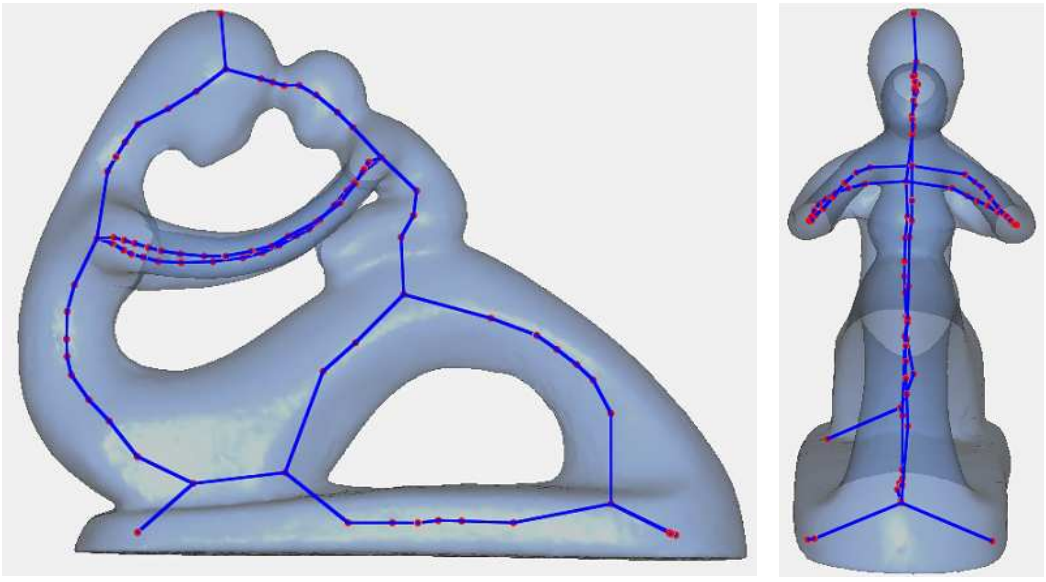
(a) (b)

Figura 9.12: Vistas del *skeleton* no centrado de la malla de superficie *fertility*.



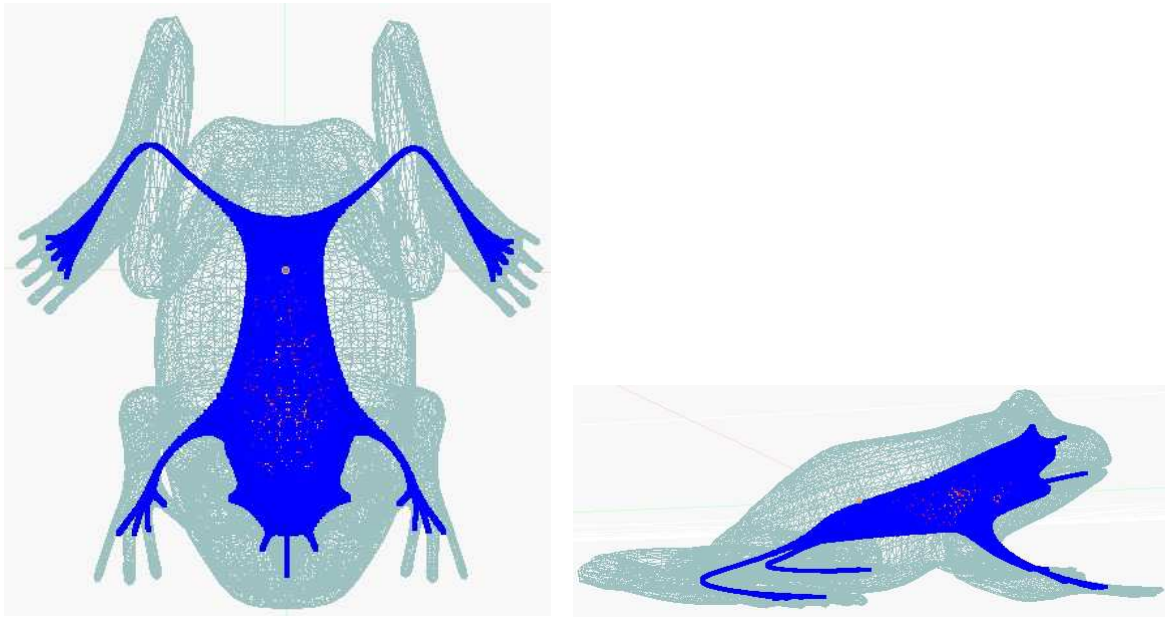
(a) (b)

Figura 9.13: Vistas del *skeleton* de la malla de superficie *fertility*.



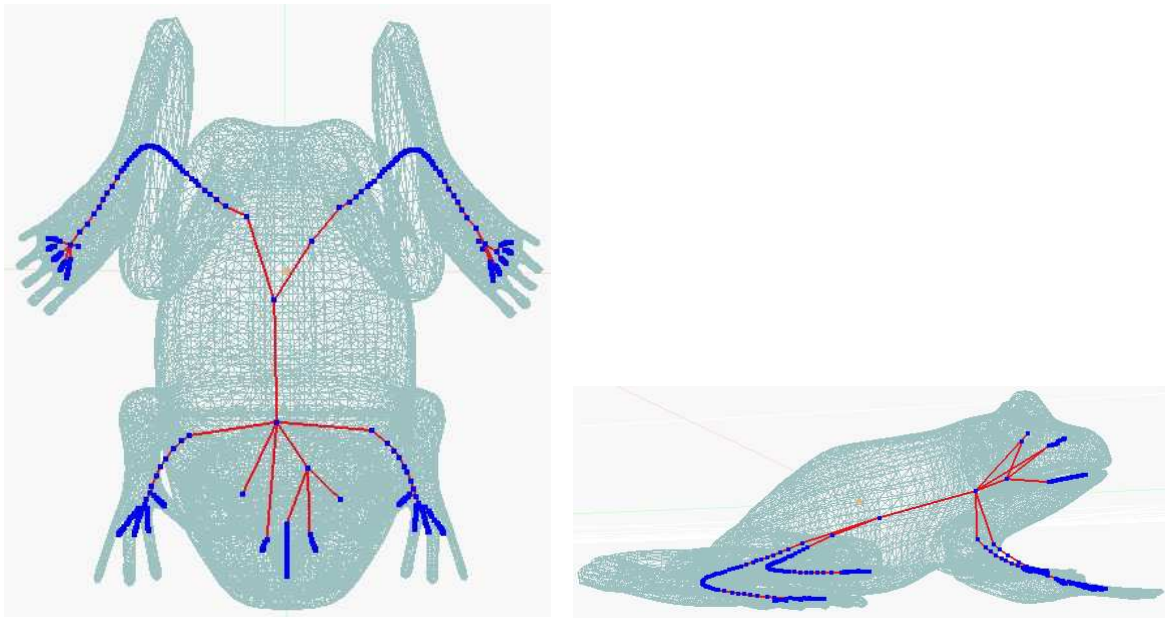
(a) (b)

Figura 9.14: Vistas del *skeleton* de la malla de superficie *fertility* generado con el demo de [2].



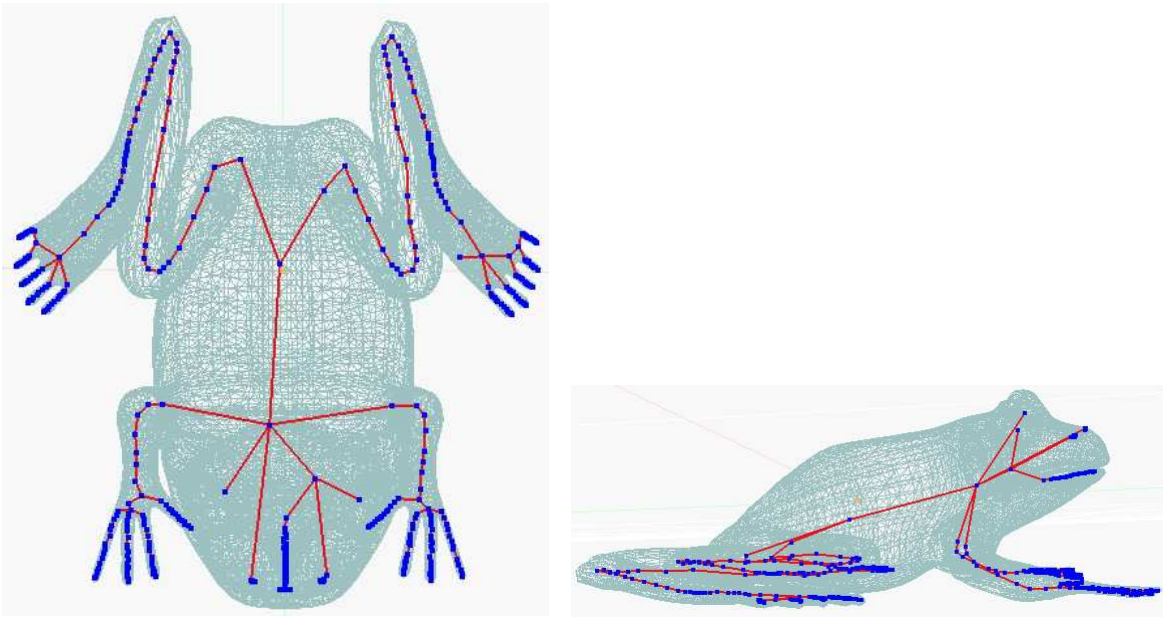
(a) (b)

Figura 9.15: Vistas de la malla de superficie *frog* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



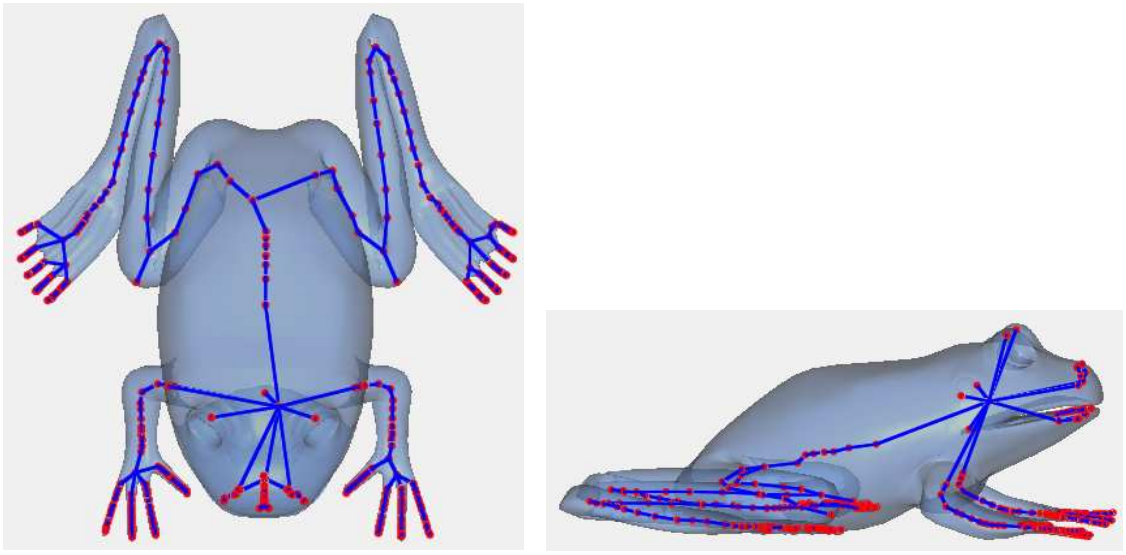
(a) (b)

Figura 9.16: Vistas del *skeleton* no centrado la malla de superficie *frog*.



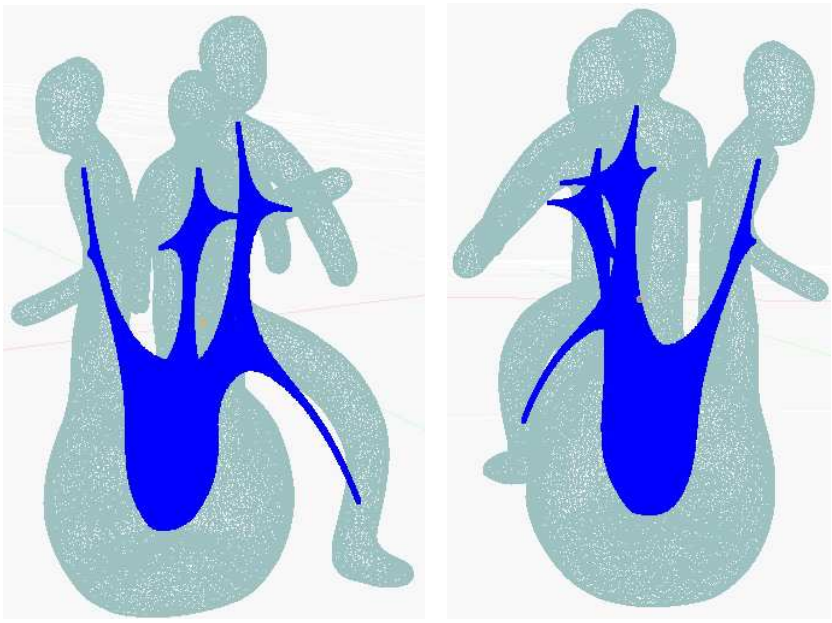
(a) (b)

Figura 9.17: Vistas del *skeleton* de la malla de superficie *frog*.



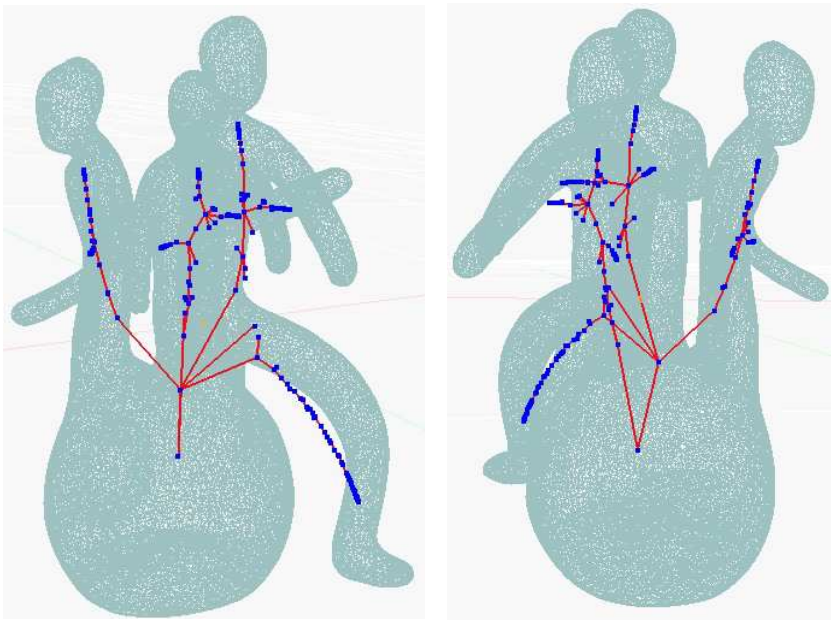
(a) (b)

Figura 9.18: Vistas del *skeleton* de la malla de superficie *frog* generado con el demo de [2].



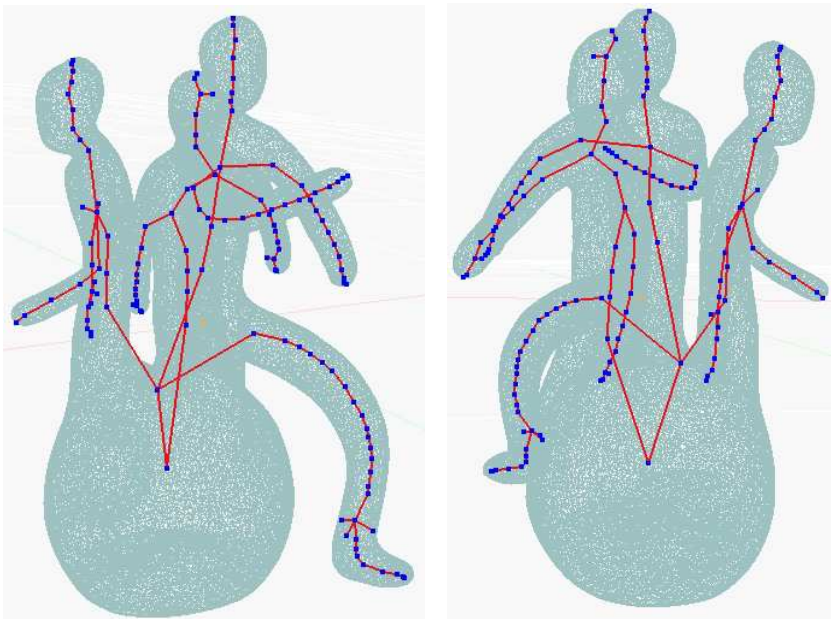
(a) (b)

Figura 9.19: Vistas de la malla de superficie *memento* contraída. La malla fue contraída hasta alcanzar el 15 % del área de superficie inicial utilizando el método *Geometry Contraction I*.



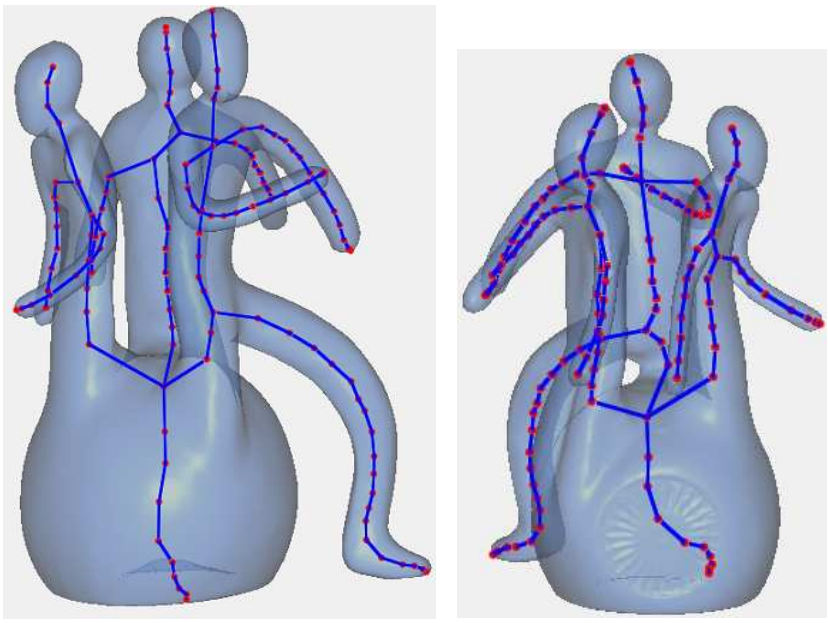
(a) (b)

Figura 9.20: Vistas del *skeleton* no centrado de la malla de superficie *memento*.



(a) (b)

Figura 9.21: Vistas del *skeleton* de la malla de superficie *memento*.



(a) (b)

Figura 9.22: Vistas del *skeleton* de la malla de superficie *memento* generado con el demo de [2].

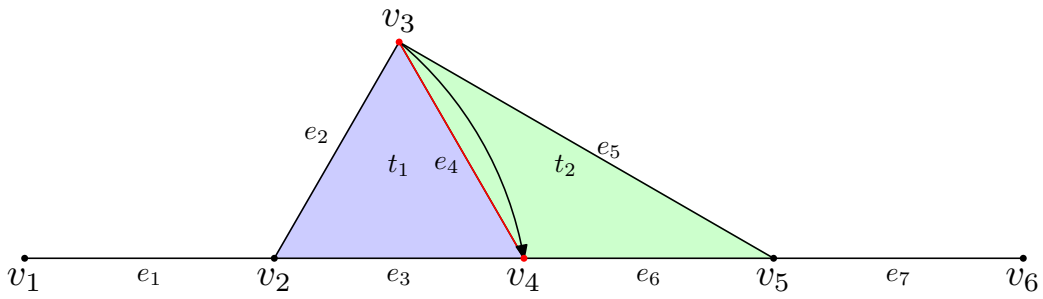


Figura 9.23: Región de malla antes del colapso del arco e_4 (en rojo). Se eliminan los triangulos t_1 (en azul) y t_2 (en verde) y los arcos e_2 y e_5.

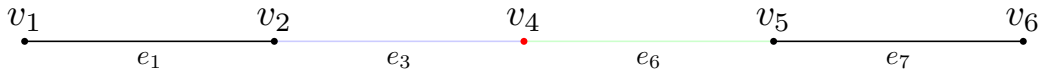


Figura 9.24: Región de la malla después del colapso del arco e_4.

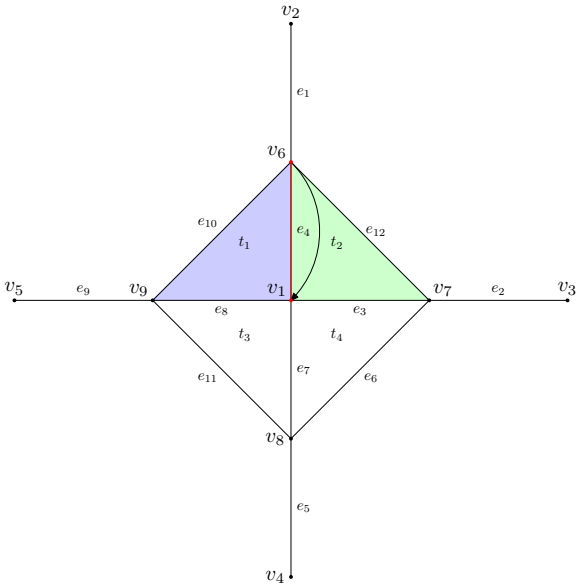


Figura 9.25:]
Región de la malla antes del colapso del arco e_4 (en rojo). Se eliminan los triangulos t_1 (en azul) y t_2 (en verde) y los arcos e_{10} y e_{12}.

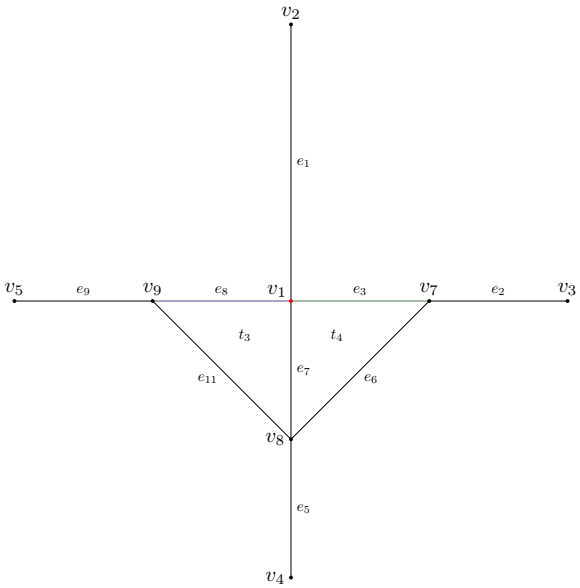


Figura 9.26: Región de la malla después del colapso del arco e_4

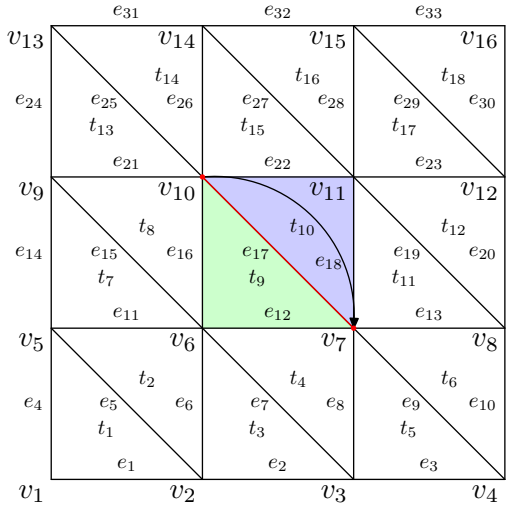


Figura 9.27: Región de la malla antes del colapso del arco e_{17} (en rojo). Se eliminan los triangulos t_9 (en verde) y t_{10} (en azul) y los arcos e_{16} y e_{22}.

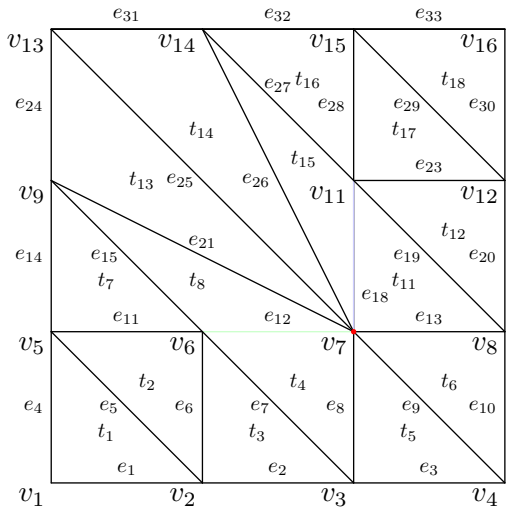


Figura 9.28: Región de la malla después del colapso del arco e_{17}

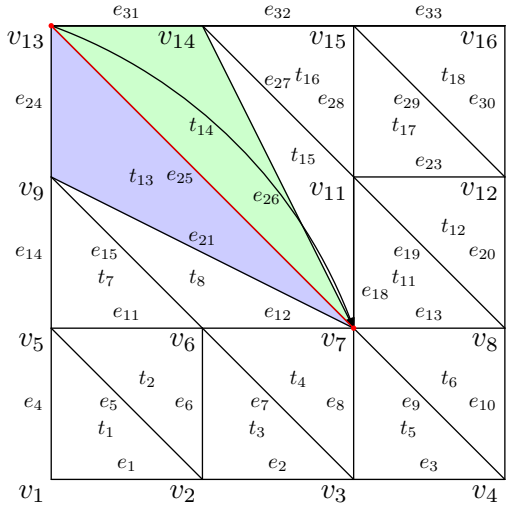


Figura 9.29: Región de la malla antes del colapso del arco e_{25} (en rojo). Se eliminan los triangulos t_{13} (en azul) y t_{14} (en verde) y los arcos e_{24} y e_{31}.

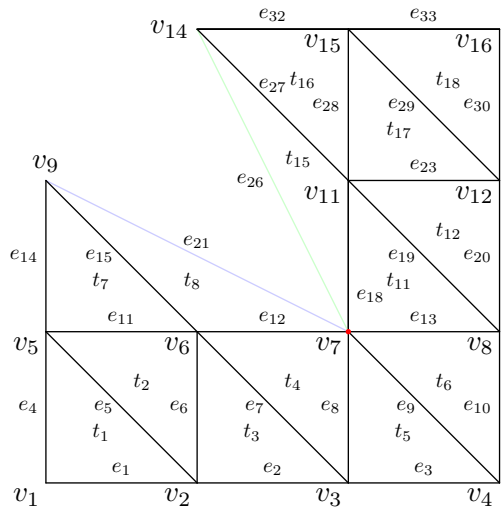


Figura 9.30: Región de la malla después del colapso del arco e_{25}

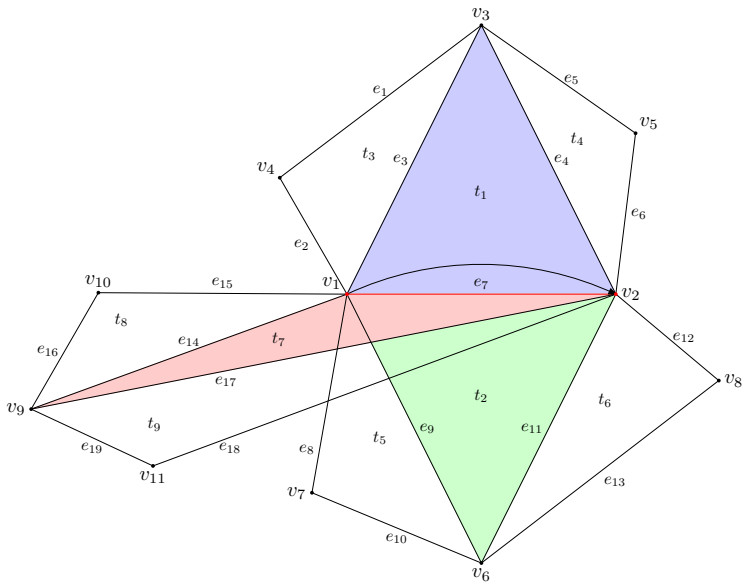


Figura 9.31: Región de malla antes del colapso de un arco con tres triángulos incidentes. Al colapsar el arco e_7 (en rojo), se eliminan los triángulos t_1 (en azul), t_2 (en verde) y t_7 (en rosado) y los arcos e_3, e_9 y e_{14}.

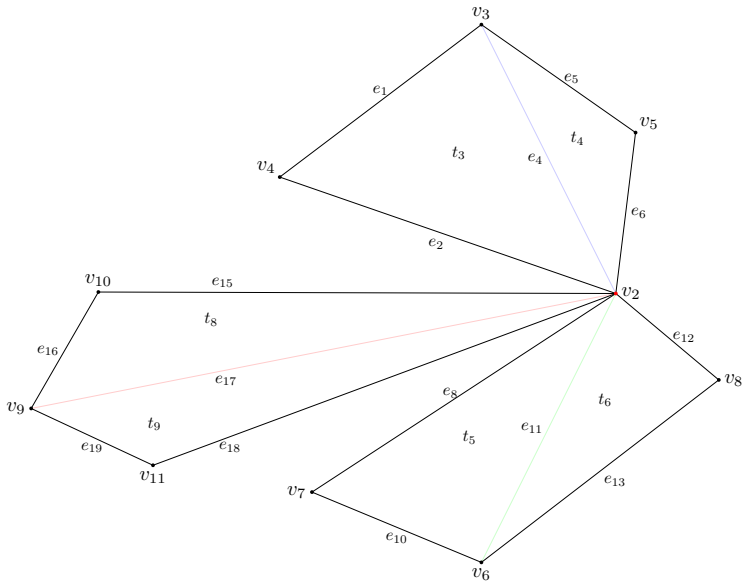


Figura 9.32: Región de malla después del colapso de un arco con tres triángulos incidentes. Al colapsar el arco e_7, se retienen los arcos e_4 (en azul), e_{11} (en verde) y e_{17} (en rosado).

B . Gráficos

B .1. Estadísticas de las Mallas de Entrada

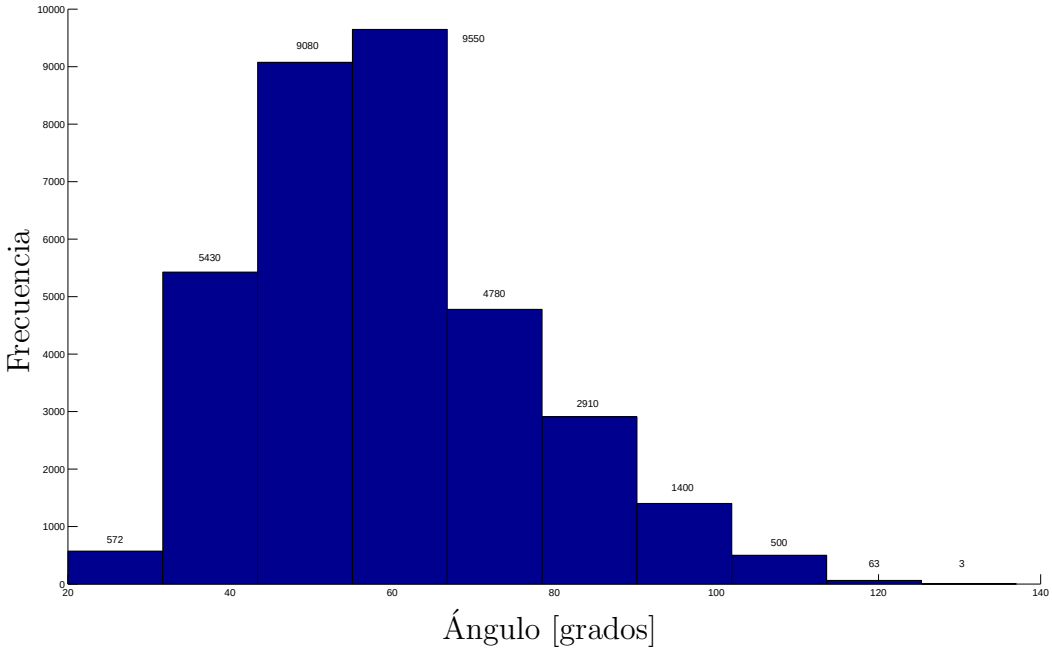


Figura 9.33: Distribución de los ángulos de la malla de superficie de la región de *retículo*.

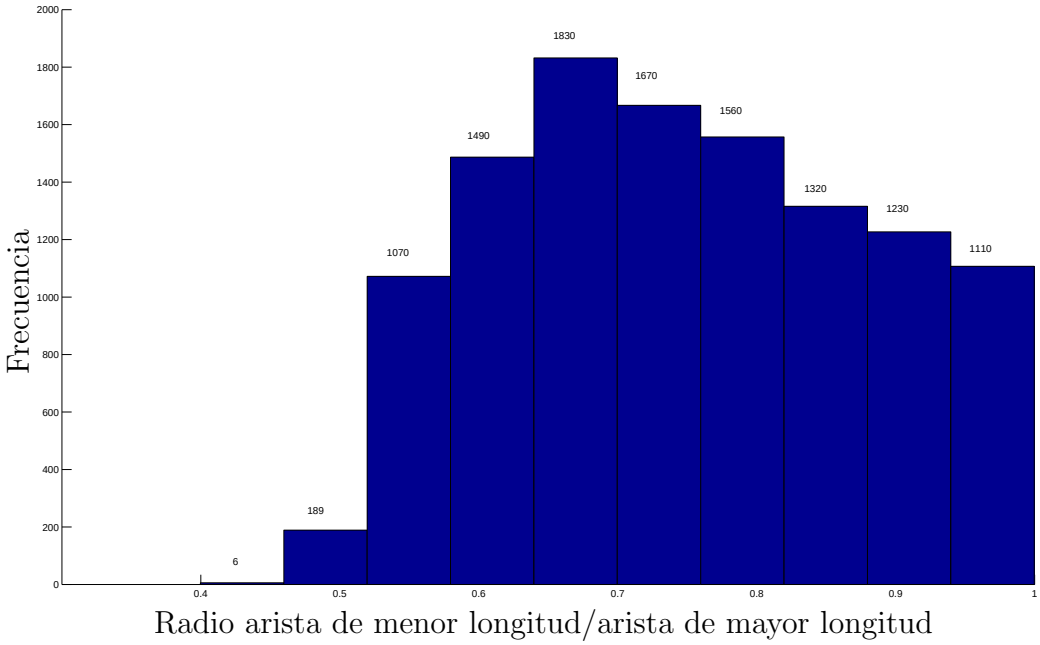


Figura 9.34: Distribución del radio aristas de menor longitud/arista de mayor longitud en los triángulos de la malla de superficie de la región de *retículo*.

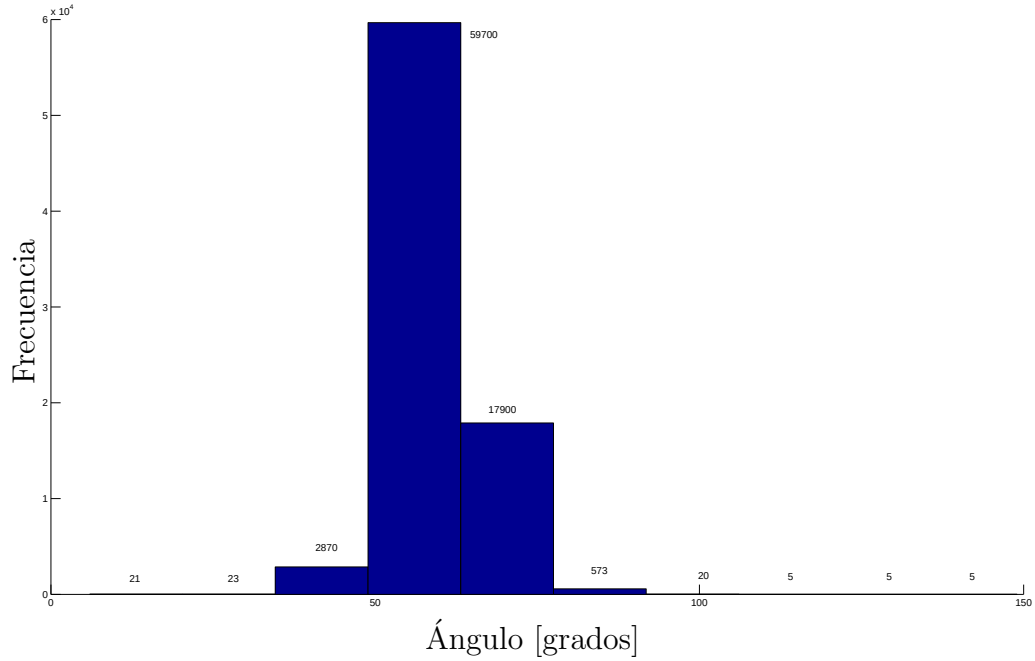


Figura 9.35: Distribución de los ángulos de la malla de superficie de la *neurona*.

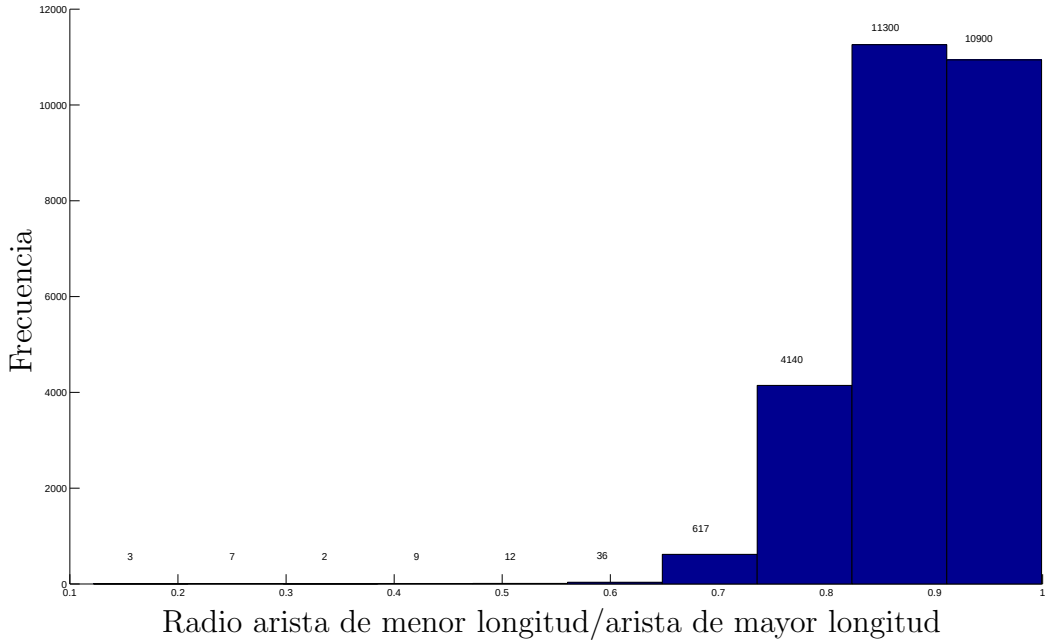


Figura 9.36: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de superficie de la *neurona*.

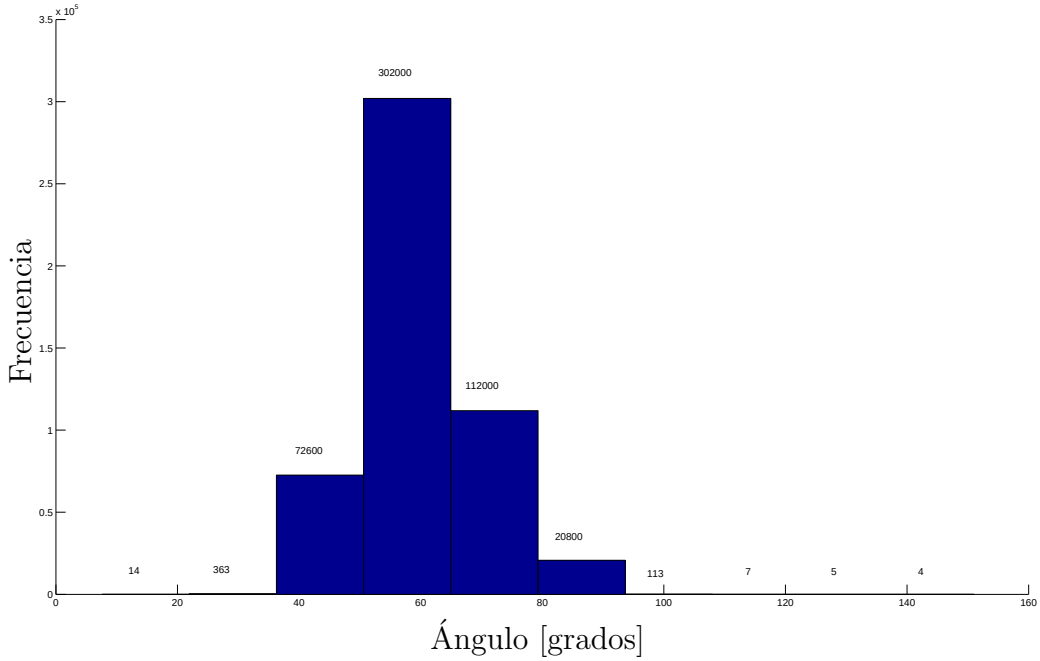


Figura 9.37: Distribución de los ángulos de la malla de superficie de la *cresta neuronal*.

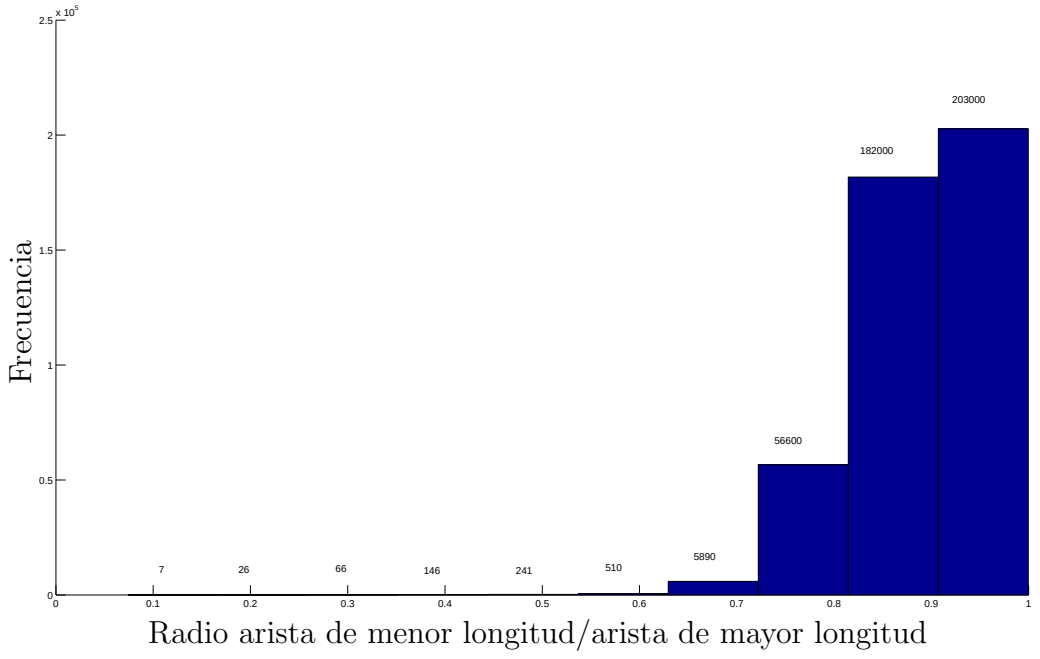


Figura 9.38: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de la *cresta neuronal*.

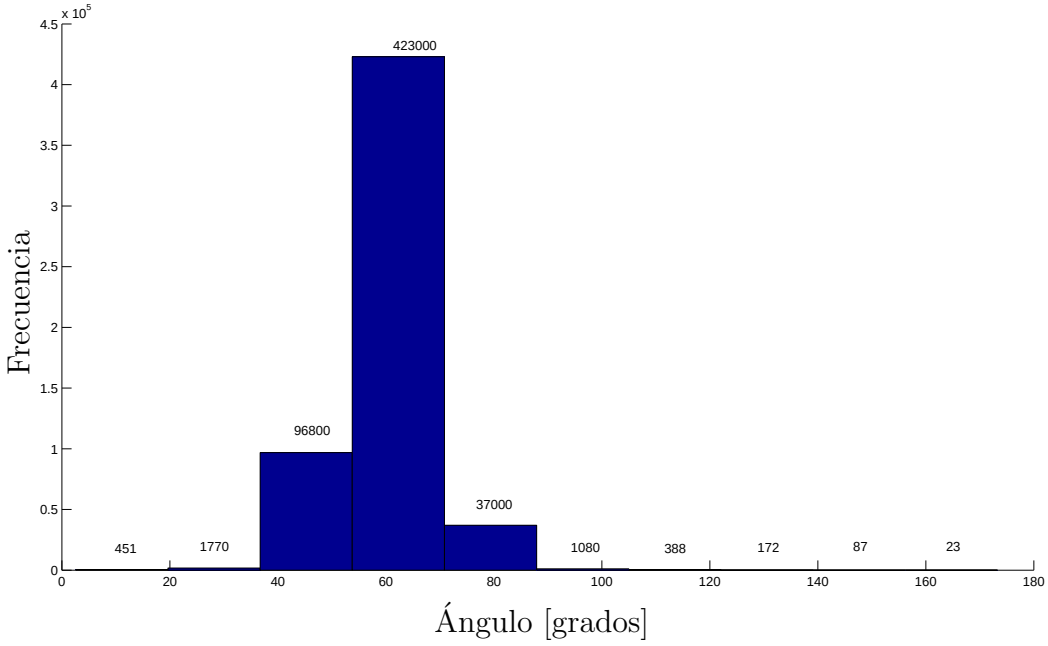


Figura 9.39: Distribución de los ángulos de la malla de superficie del *retículo*.

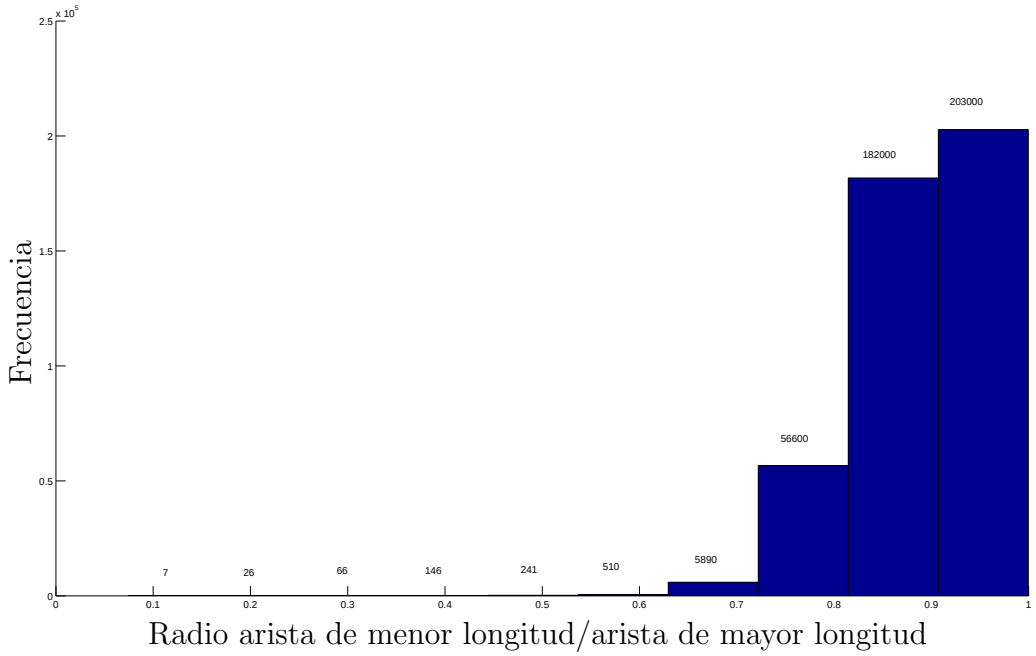


Figura 9.40: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de superficie del *retículo*.

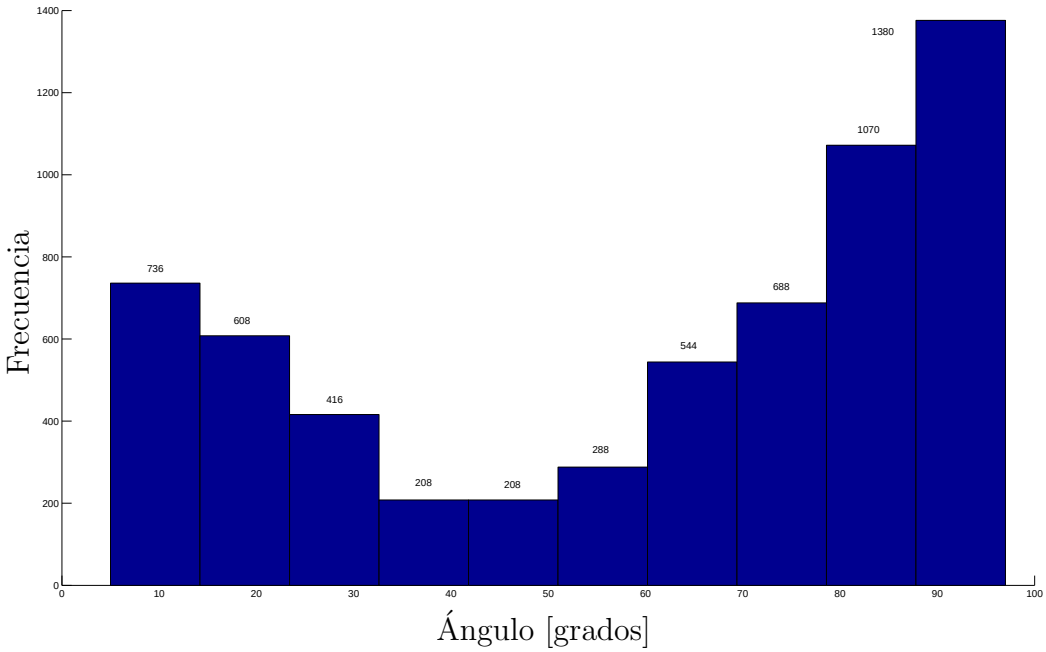


Figura 9.41: Distribución de los ángulos de la malla de superficie *donuts*.

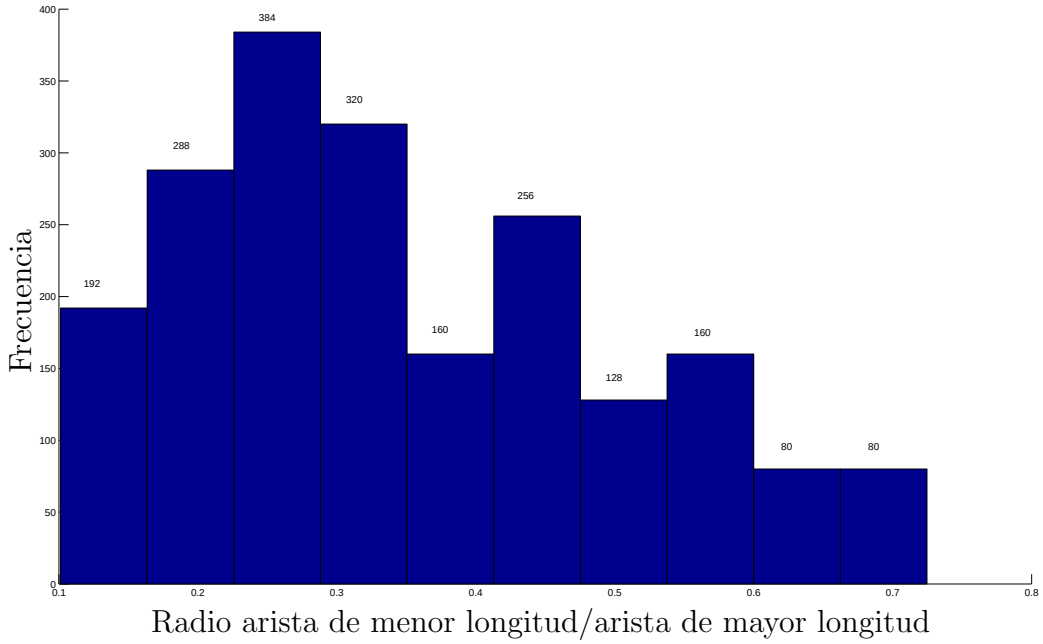


Figura 9.42: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de superficie *donuts*.

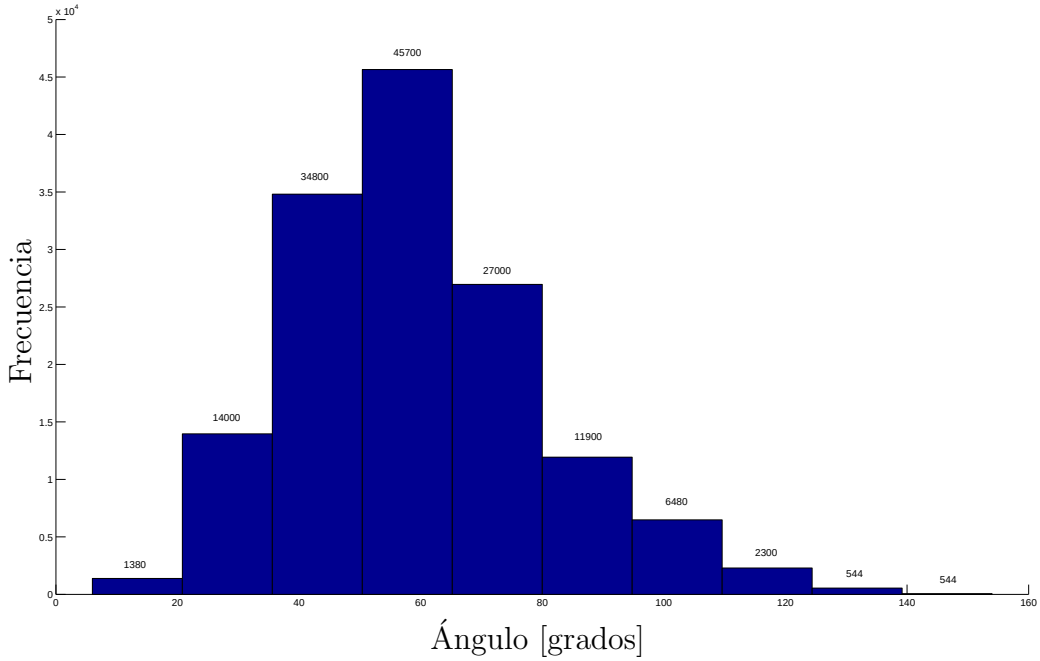


Figura 9.43: Distribución de los ángulos de la malla de superficie *elk*.

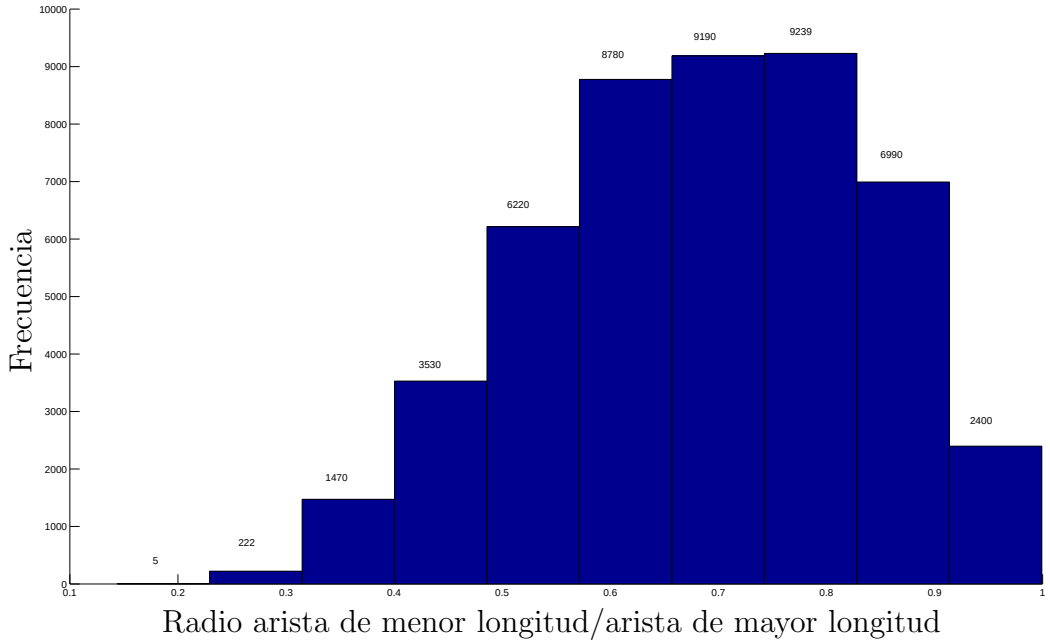


Figura 9.44: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de superficie *elk*.

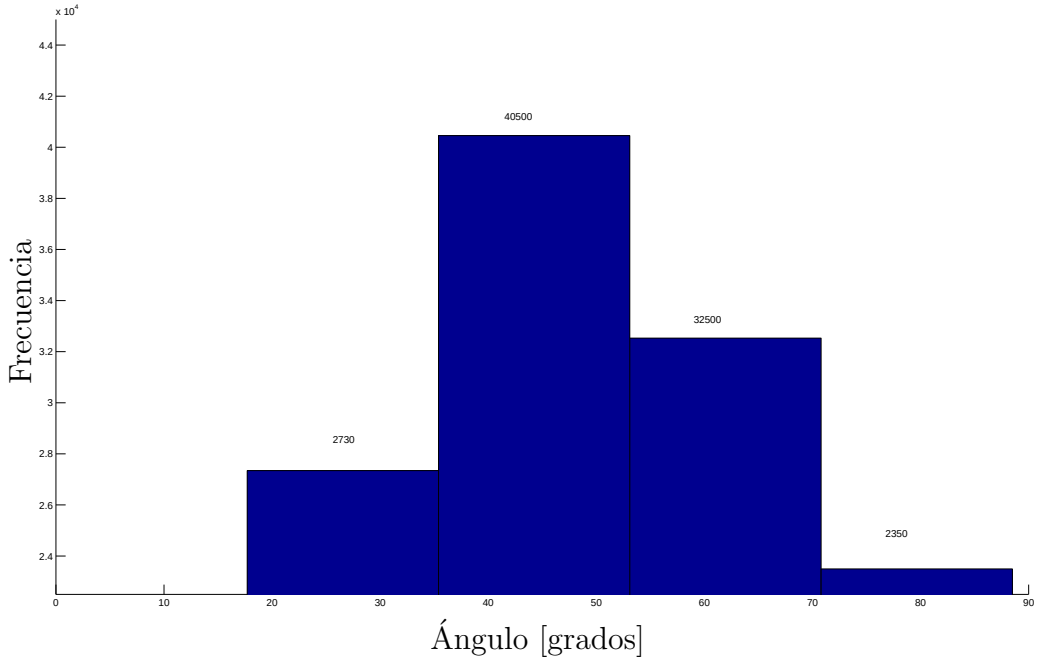


Figura 9.45: Distribución de los ángulos de la malla de superficie *fertility*.

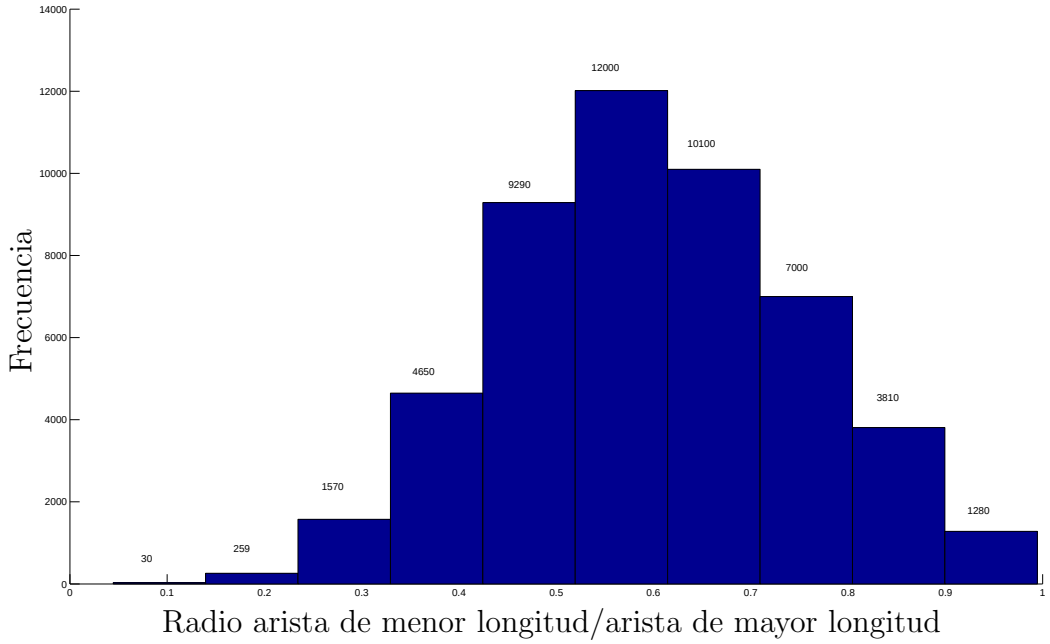


Figura 9.46: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de superficie *fertility*.

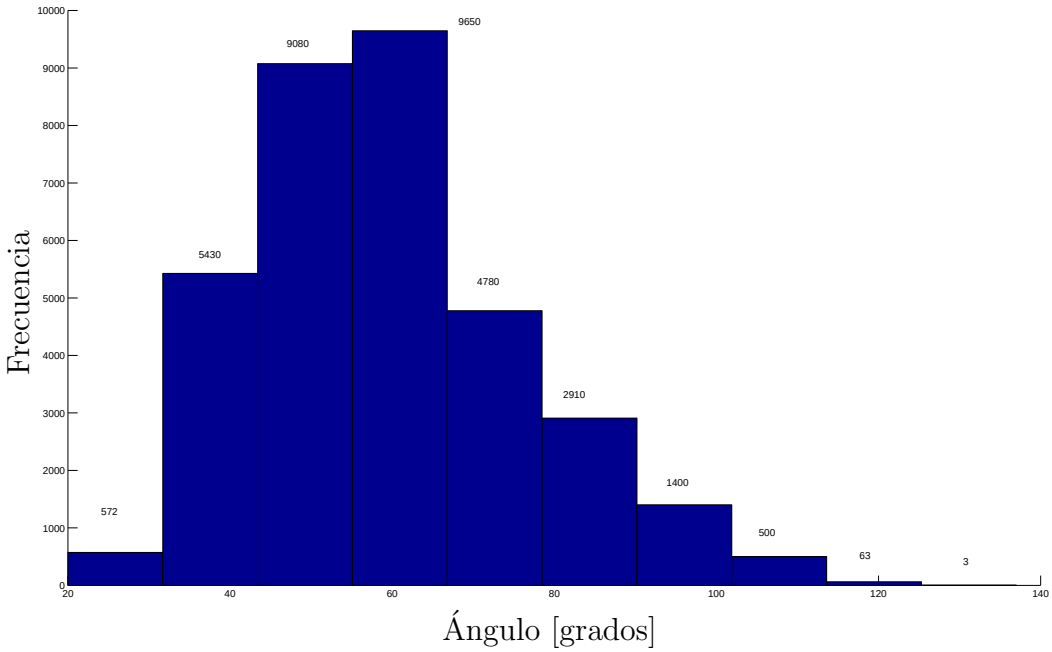


Figura 9.47: Distribución de los ángulos de la malla de superficie *frog*.

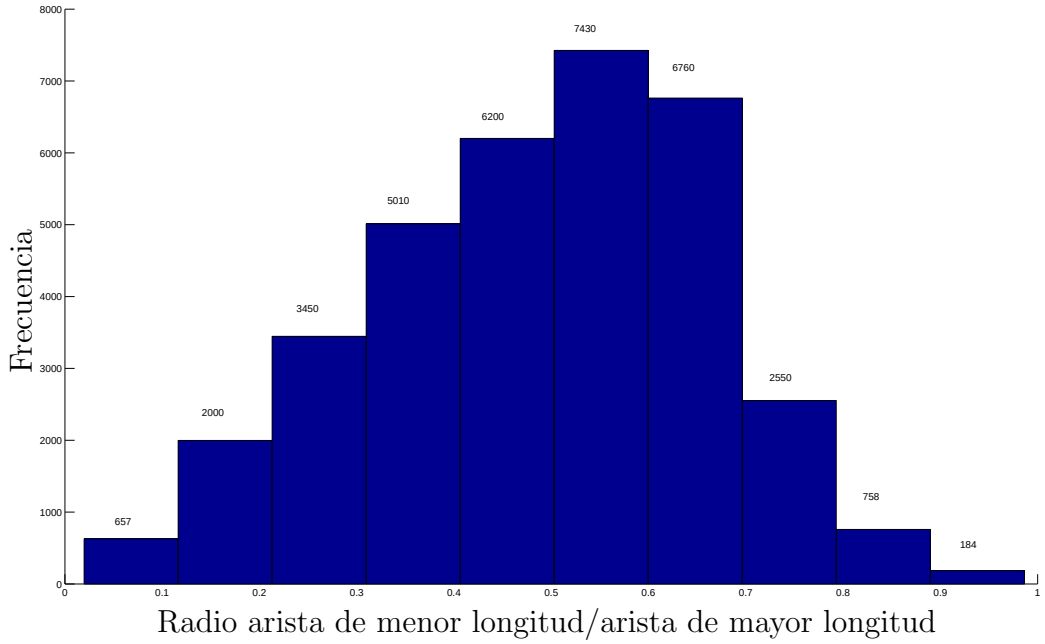


Figura 9.48: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de superficie *frog*.

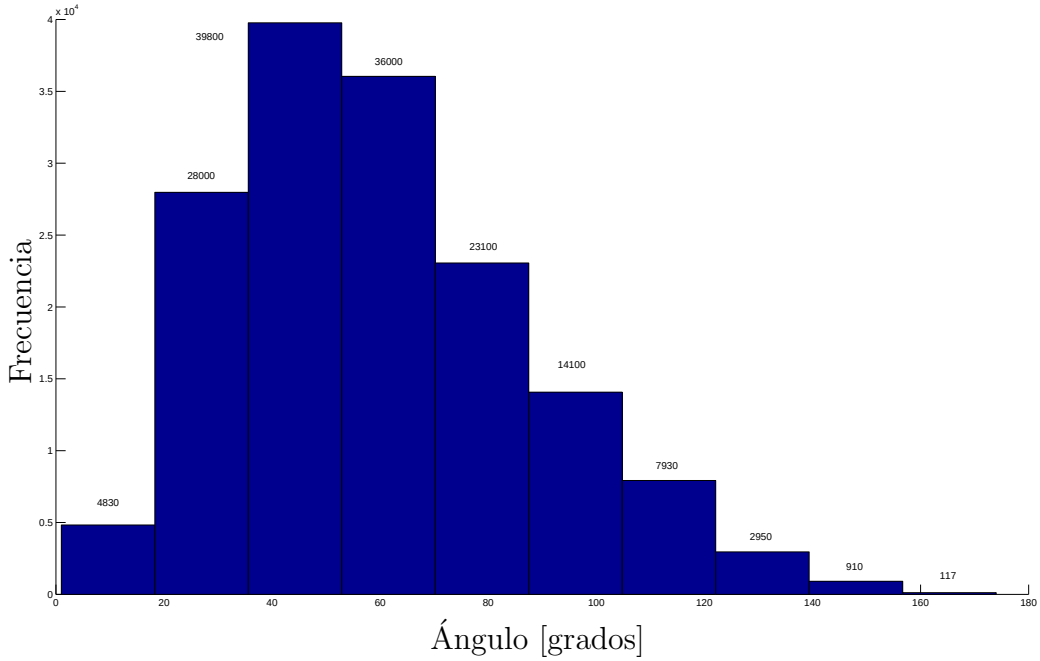


Figura 9.49: Distribución de los ángulos de la malla de superficie *memento*.

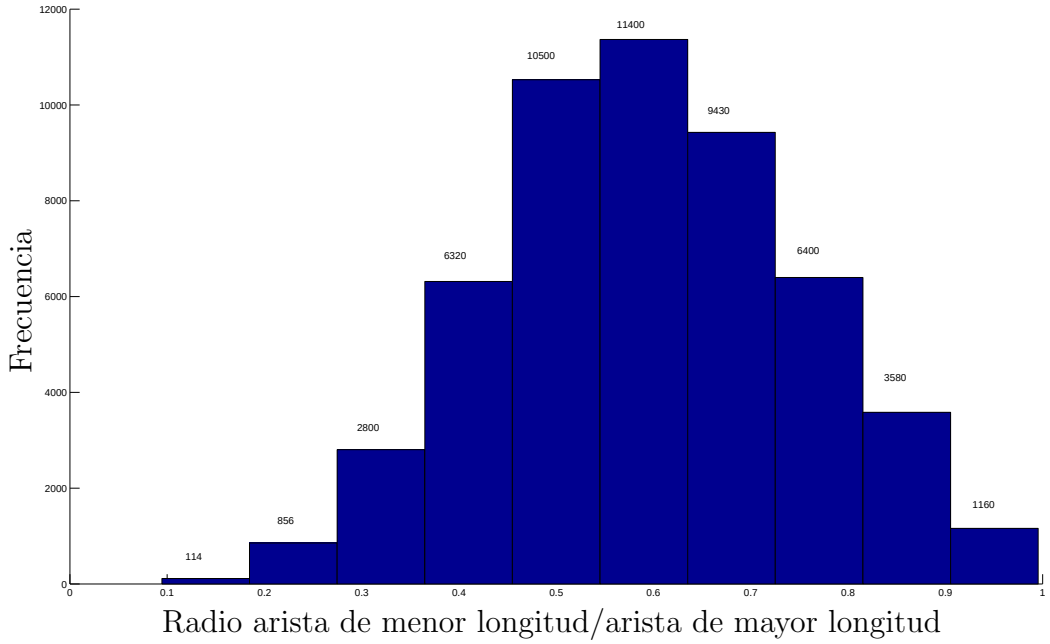


Figura 9.50: Distribución del radio (arista de menor longitud/arista de mayor longitud) de los triángulos de la malla de superficie *memento*.

B .2. Variación del Vector Normal Durante la Etapa de Contracción

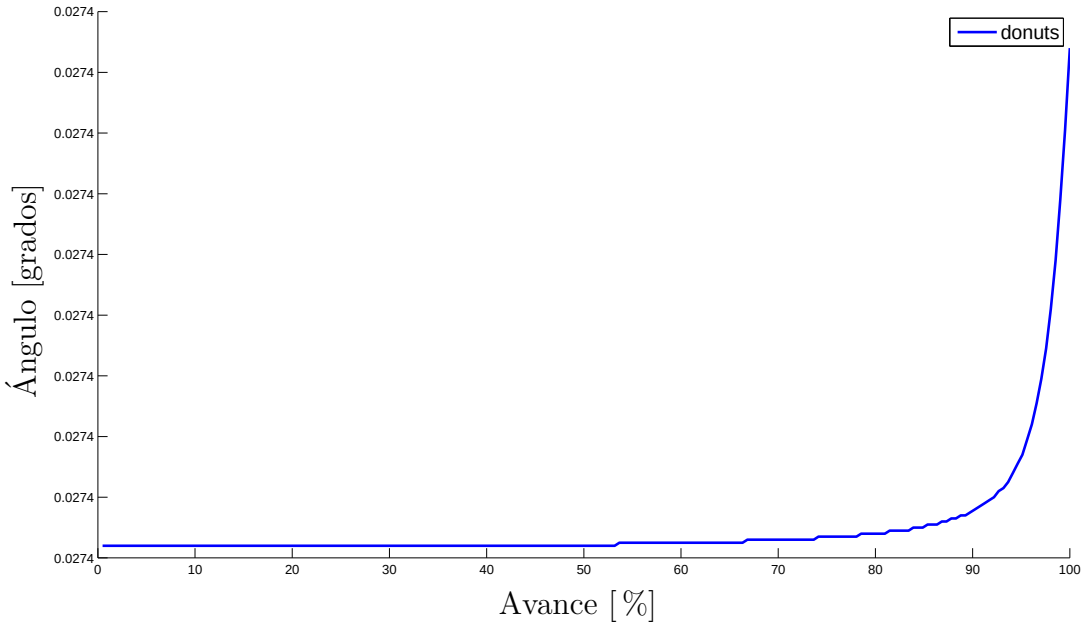


Figura 9.51: Promedio del ángulo entre el vector normal inicial y el vector normal recalculado de los vértices de la malla *donuts* durante la etapa de contracción. Esta malla fue contraída utilizando el método *GeometryContractionIII*.

B .3. Decaimiento del Área de las Mallas Durante la Etapa de Contracción

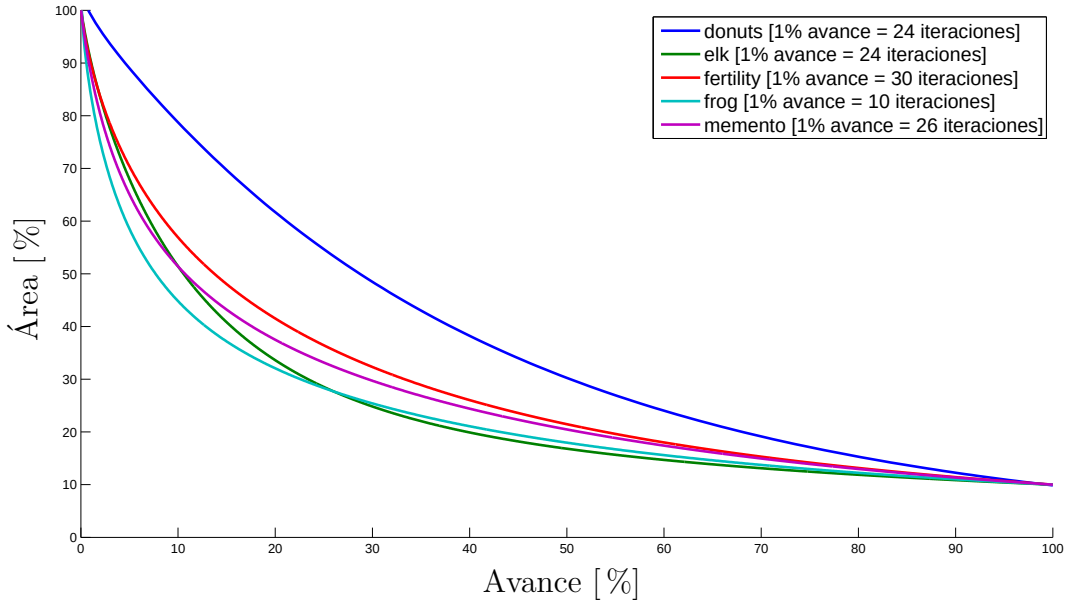


Figura 9.52: Decaimiento porcentual del área de las mallas generadas artificialmente durante la etapa de contracción (*geometry contraction*). Las mallas fueron contraídas utilizando el método escogido (*GeometryContractionI*).

B .4. Desplazamiento de los Vértices Durante la Etapa de Contracción

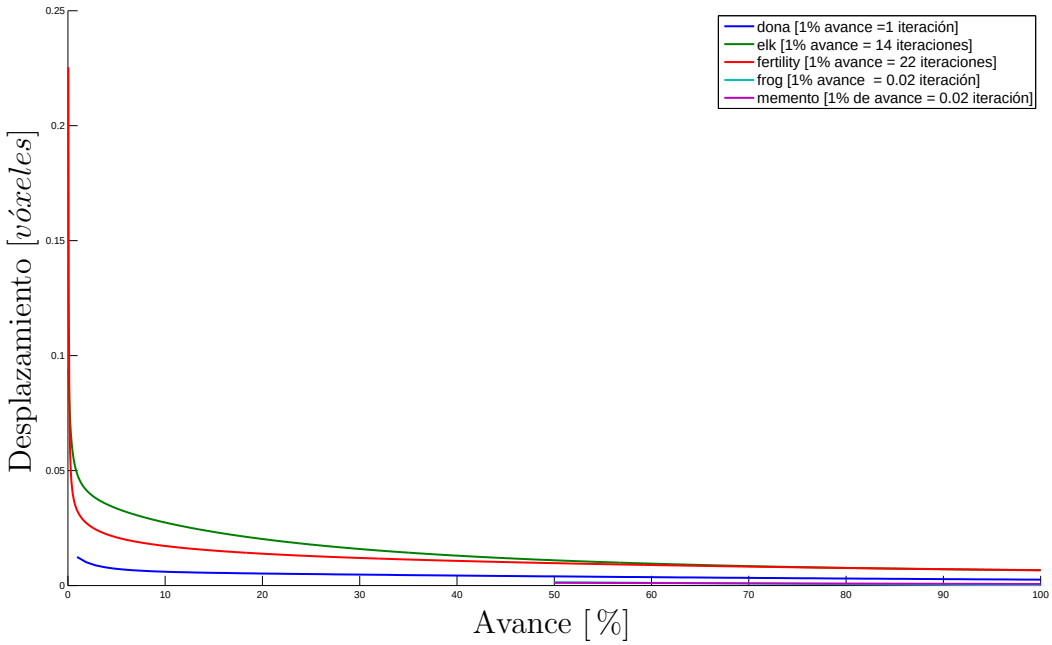


Figura 9.53: Promedio del desplazamiento de los vértices de las malla generadas artificialmente durante la etapa de contracción (*geometry contraction*). Para contraer las mallas se utilizó el método escogido (*GeometryContractionI*) hasta que el promedio de desplazamiento alcanzó el umbral 0.001.

B .5. Función de Costo Durante la Etapa de Colapso de Aristas

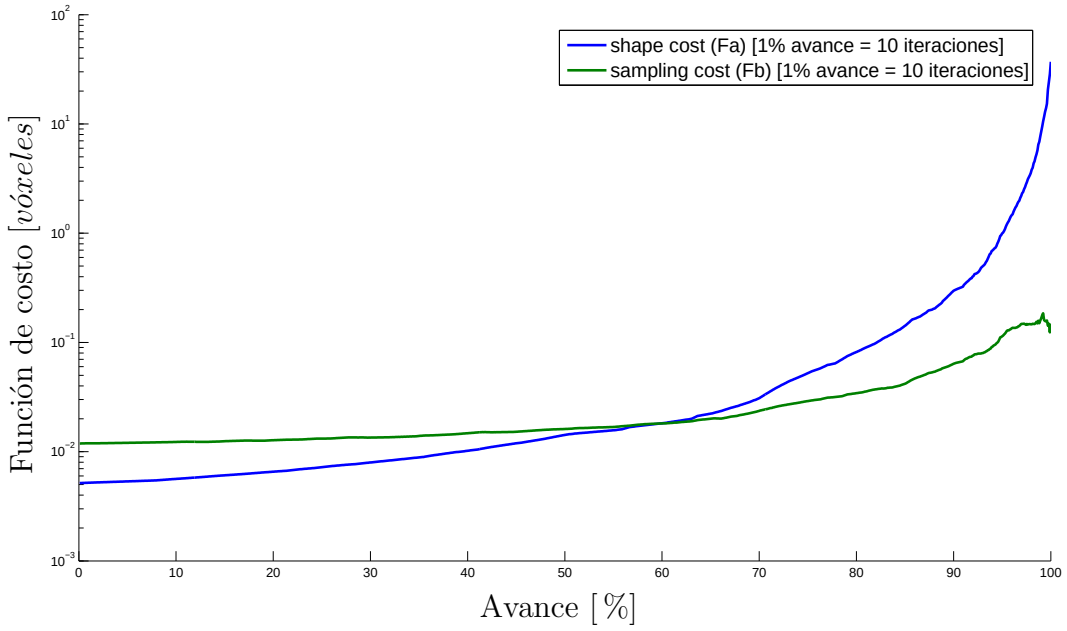


Figura 9.54: Promedio de las funciones de costo de la malla de superficie *donuts* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

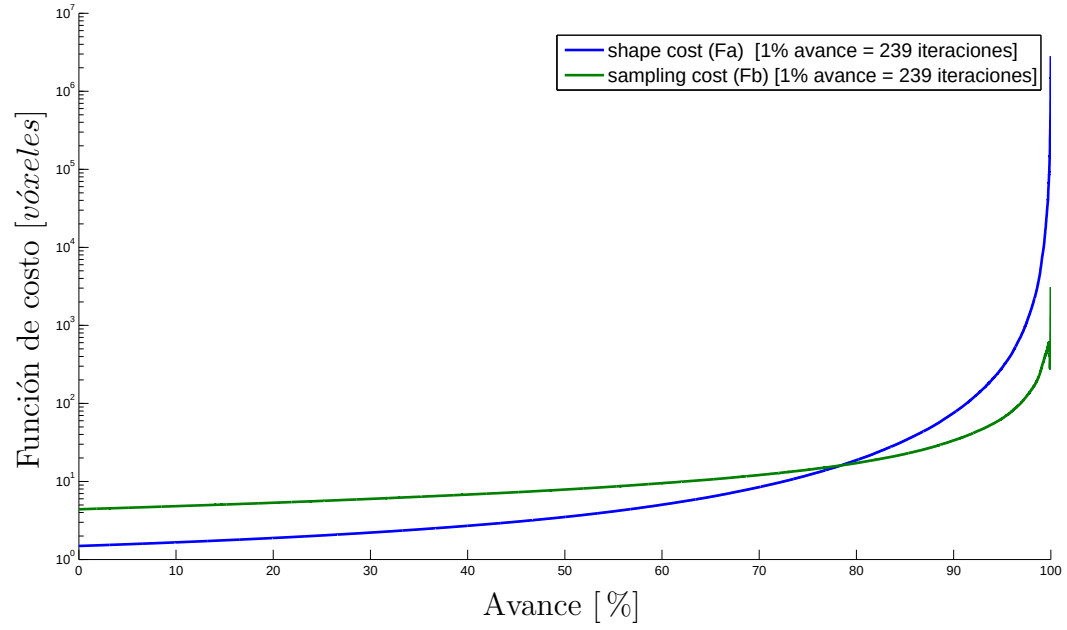


Figura 9.55: Promedio de las funciones de costo de la malla de superficie *elk* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

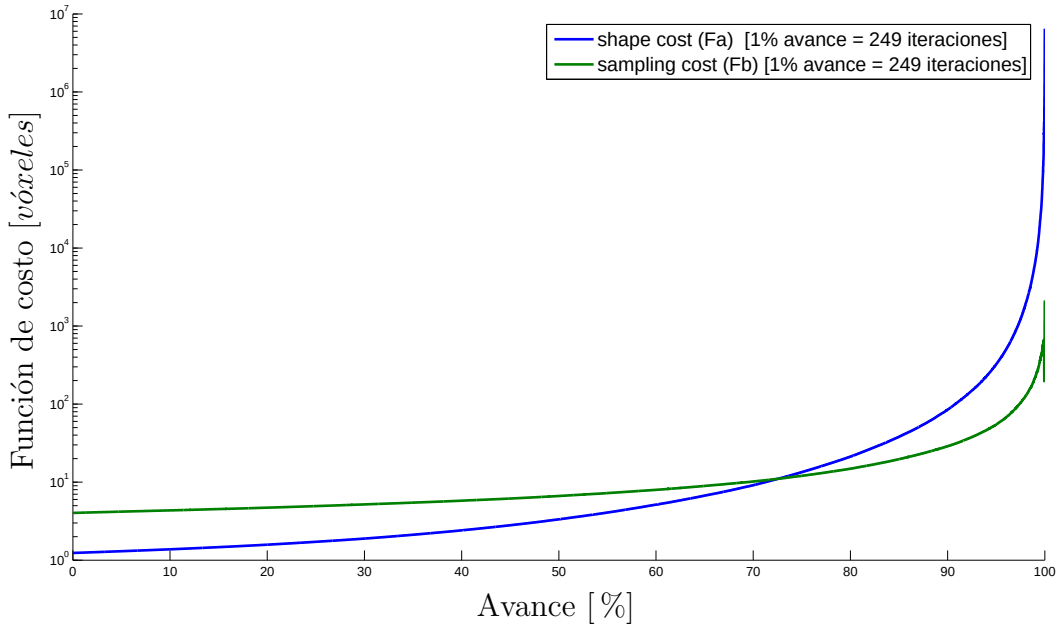


Figura 9.56: Promedio de las funciones de costo de la malla de superficie *fertility* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

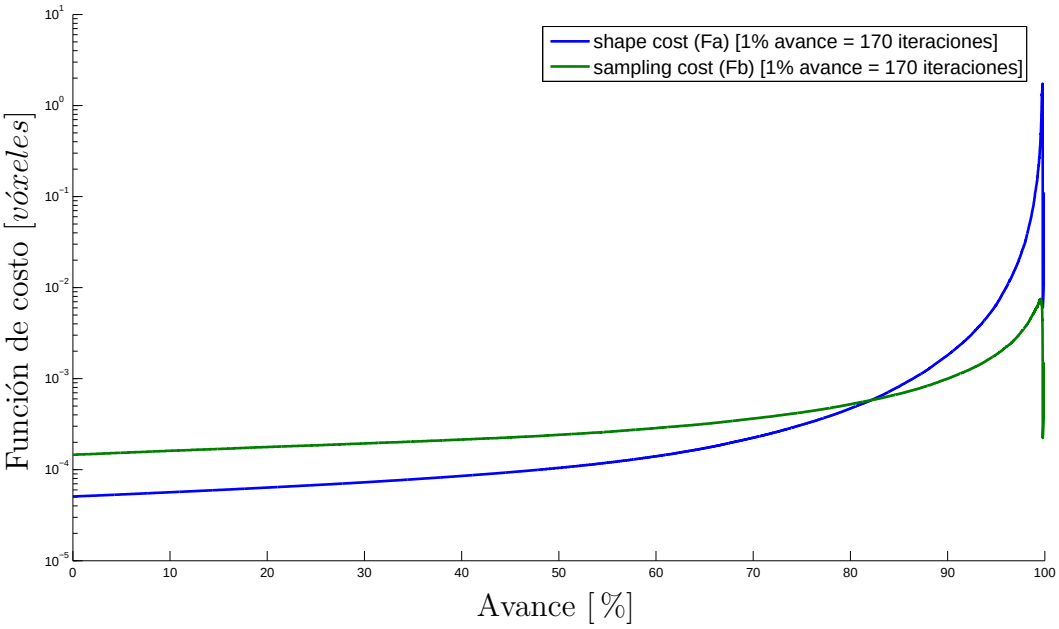


Figura 9.57: Promedio de las funciones de costo de la malla de superficie *frog* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

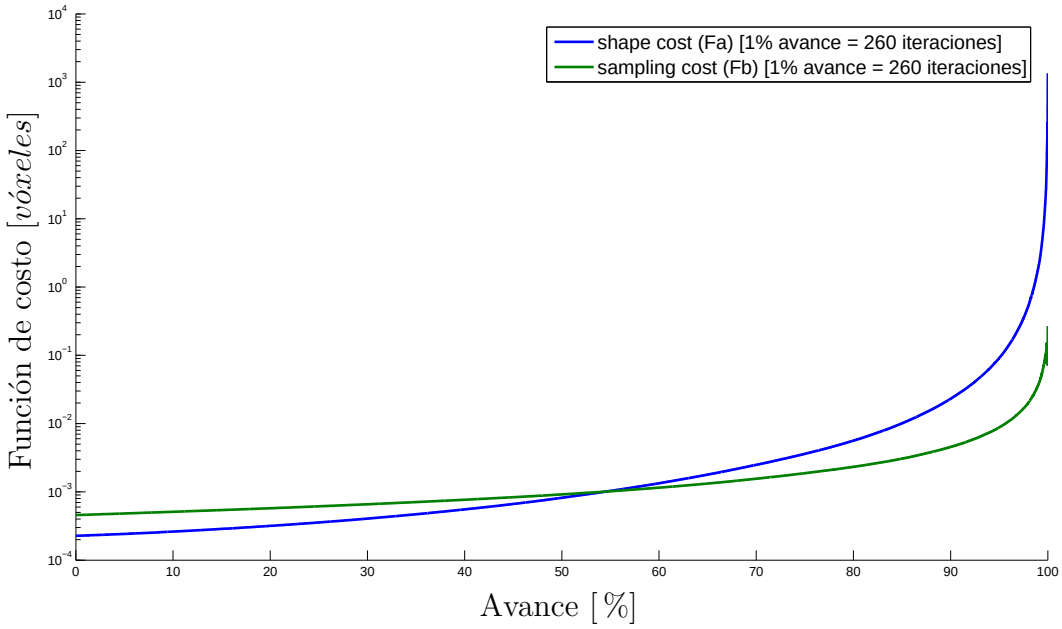


Figura 9.58: Promedio de las funciones de costo de la malla de superficie *memento* durante la etapa de colapso de aristas (*connectivity surgery*). Las definiciones de *shape cost* (F_a) y *sampling cost* (F_b) se encuentran en las Ecuaciones (4.11) y (4.17), respectivamente.

C . Tablas

C .1. Estadísticas de las Mallas

Malla	N° de vértices	N° de triángulos	N° de arcos
región de *retículo*	5720	11460	17190
neurona	13508	27028	40542
cresta neural	84513	169218	253827
retículo	92452	280398	186928
donuts	1024	2048	3072
elk	24013	48026	72039
fertility	24994	50000	75000
frog	17484	34964	52446
memento	26277	52550	78825

Cuadro 9.1: N° de componentes de las mallas.

Malla	*Skeleton* con ruido	*Skeleton* final
región de *retículo*	278	240
neurona	298	288
cresta neural	708	661
retículo	7573	7364
donuts	32	32
elk	87	44
fertility	88	47
frog	416	412
memento	218	134

Cuadro 9.2: Comparación entre el N° de nodos del *skeleton* con ruido y el N° de nodos del *skeleton* final (sin ruido).

Malla	Implementación Actual	Demo de [2]
región de *retículo*	0.0419	0.0360
neurona	0.0213	0.0248
cresta neural	0.0078	0.0086
retículo	0.0796	0.0669
donuts	0.0068	0.0351
elk	0.0018	0.0039
fertility	0.0018	0.0035
frog	0.0235	0.0212
memento	0.0050	0.0049

Cuadro 9.3: Radio N° de nodos del *skeleton*/N° de vértices de la malla

C .2. Rendimiento

Malla	Tiempo de procesamiento (dd - hh:mm:ss)		
	geometry contraction	*connectivity surgery*	*embedding refinement*
región de *reticulo*	00:00:00,10	00:00:57,20	00:00:00,03
neurona	00:00:01,60	00:08:27,80	00:00:00,08
cresta neural	00:06:16,60	17:26:48,60	00:00:00,16
retículo	00:00:6,75	102:09:00,00	00:00:02,48
donuts	00:00:00,08	00:00:01,20	00:00:00,00
elk	00:00:22,40	00:17:00,40	00:00:00,03
fertilty	00:00:28,80	00:15:39,90	00:00:00,06
frog	00:00:08,80	00:26:58,20	00:00:00,08
memento	00:00:25,70	00:31:21,00	00:00:00,06

Cuadro 9.4: Tiempo de ejecución de las etapas del algoritmo *Skeleton extraction by mesh contraction*.

D . Demostraciones Matemáticas

D .1. Matriz K

El costo de la forma o *shape cost* es uno de los dos términos de la función de costo asociada a cada arista de la malla durante la etapa de colapso de éstas (*connectivity surgery*) y se define de la siguiente manera:

$$F_i(p) = p^T \sum_{e_{(i,k)} \in \tilde{E}} (K_{i,k}^T K_{i,k})\ p = p^T Q_i p$$

donde p es el vértice resultante del colapso de $e_{(i,j)}$ (es decir, si el colapso es $(i \rightarrow j)$, entonces p es $\tilde{v_j}$ [2]) en su representación homogénea.

Esta ecuación define una matriz $K_{i,j}$ para cada arco $e_{(i,j)}$ en la malla contraída, tal que el valor $p^T(K_{i,j}^T K_{i,j})p$ es la distancia al cuadrado entre el punto p y la recta definida por el arco $e_{(i,j)}$ (Figura 9.59).

$$K_{i,j} = \begin{pmatrix} 0 & -a_z & a_y & -b_x \\ a_z & 0 & -a_x & -b_y \\ -a_y & a_x & 0 & -b_z \end{pmatrix}$$

donde

$$a = \frac{e_{(i,j)}}{\left\| e_{(i,j)} \right\|}$$
$$b = a \ \times \ \tilde{v}_i$$

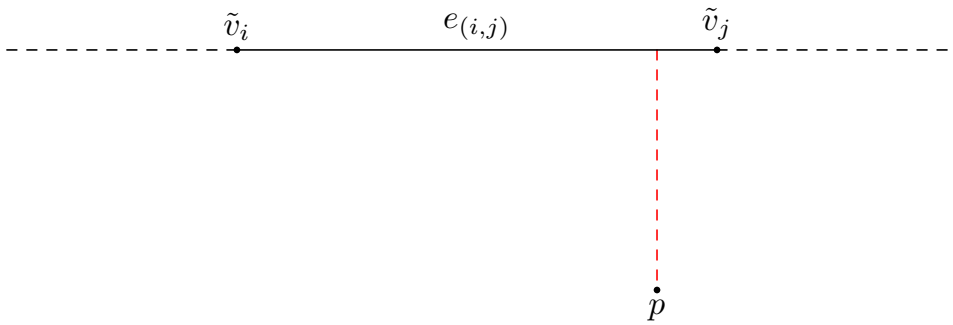


Figura 9.59: El término $p^T(K_{i,j}^T K_{i,j})p$ corresponde a la distancia al cuadrado desde el punto p a la recta definida por el arco $e_{(i,j)}$.

Demostración. La distancia de un punto a la recta definida por el arco $e_{(i,j)}$ es :

$$
\begin{aligned}
d &= \frac{(\tilde{v}_j - \tilde{v}_i) \times (\tilde{v}_i - p)}{||(\tilde{v}_j - \tilde{v}_i)||} \\
&= \frac{(\tilde{v}_j - \tilde{v}_i)}{||(\tilde{v}_j - \tilde{v}_i)||} \times \tilde{v}_i + \frac{(\tilde{v}_j - \tilde{v}_i)}{||(\tilde{v}_j - \tilde{v}_i)||} \times -p \\
&= b + a \times -p \\
&= b + p \times a
\end{aligned} \tag{9.1}
$$

Luego, la distancia al cuadrado entre el punto p a la línea definida por el arco $e_{(i,j)}$ es:

$$
\begin{aligned}
d^2 &= d \cdot d \\
&= (b + p \times a) \cdot (b + p \times a)
\end{aligned} \tag{9.2}
$$

El vector p es su representación homogénea tiene la forma:

$$
p = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}
$$

$$
p^T = \begin{pmatrix} p_x & p_y & p_z & 1 \end{pmatrix}
$$

Luego, el término $p^T(K_{i,j}^T K_{i,j})p$ es equivalente a :

$$p^T K_{i,j}^T K_{i,j} p = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}^T \begin{pmatrix} 0 & a_z & -a_y \\ -a_z & 0 & a_x \\ a_y & -a_x & 0 \\ -b_x & -b_y & -b_z \end{pmatrix} \begin{pmatrix} 0 & -a_z & a_y & -b_x \\ a_z & 0 & -a_x & -b_y \\ -a_y & a_x & 0 & -b_z \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}^T \begin{pmatrix} a_z^2 + a_y^2 & -a_y a_x & -a_z a_x & -a_z b_y + a_y b_z \\ -a_y a_x & a_z^2 + a_x^2 & -a_z a_y & a_z b_x - a_x b_z \\ -a_z a_x & -a_z a_y & a_y^2 + a_x^2 & -a_y b_x + a_x b_y \\ -a_z b_y + a_y b_z & a_z b_x - a_x b_z & -a_y b_x + a_x b_y & b_x^2 + b_y^2 + b_z^2 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

$$\begin{aligned} &= b_x^2 + 2b_x(p_y a_z - p_z a_y) + p_y^2 a_z^2 - 2p_y p_z a_z a_y + p_z^2 a_y^2 + \\ &\quad b_y^2 + 2b_y(p_z a_x - p_x a_z) + p_z^2 a_x^2 - 2p_x p_z a_z a_x + p_x^2 a_z^2 + \\ &\quad b_z^2 + 2b_z(p_x a_y - p_y a_x) + p_x^2 a_y^2 - 2p_x p_y a_y a_x + p_y^2 a_x^2 \end{aligned}$$

$$= (b_x + (p_y a_z - p_z a_y))^2 + (b_y + (p_z a_x - p_x a_z))^2 + (b_z + (p_x a_y - p_y a_x))^2$$

$$= \begin{pmatrix} b_x + p_y a_z - p_z a_y \\ b_y + p_z a_x - p_x a_z \\ b_z + p_x a_y - p_y a_x \end{pmatrix} \cdot \begin{pmatrix} b_x + p_y a_z - p_z a_y \\ b_y + p_z a_x - p_x a_z \\ b_z + p_x a_y - p_y a_x \end{pmatrix}$$

$$= (b + p \times a) \cdot (b + p \times a)$$

que es equivalente a la Ecuación (9.2).

□

E . Diagramas de Clase

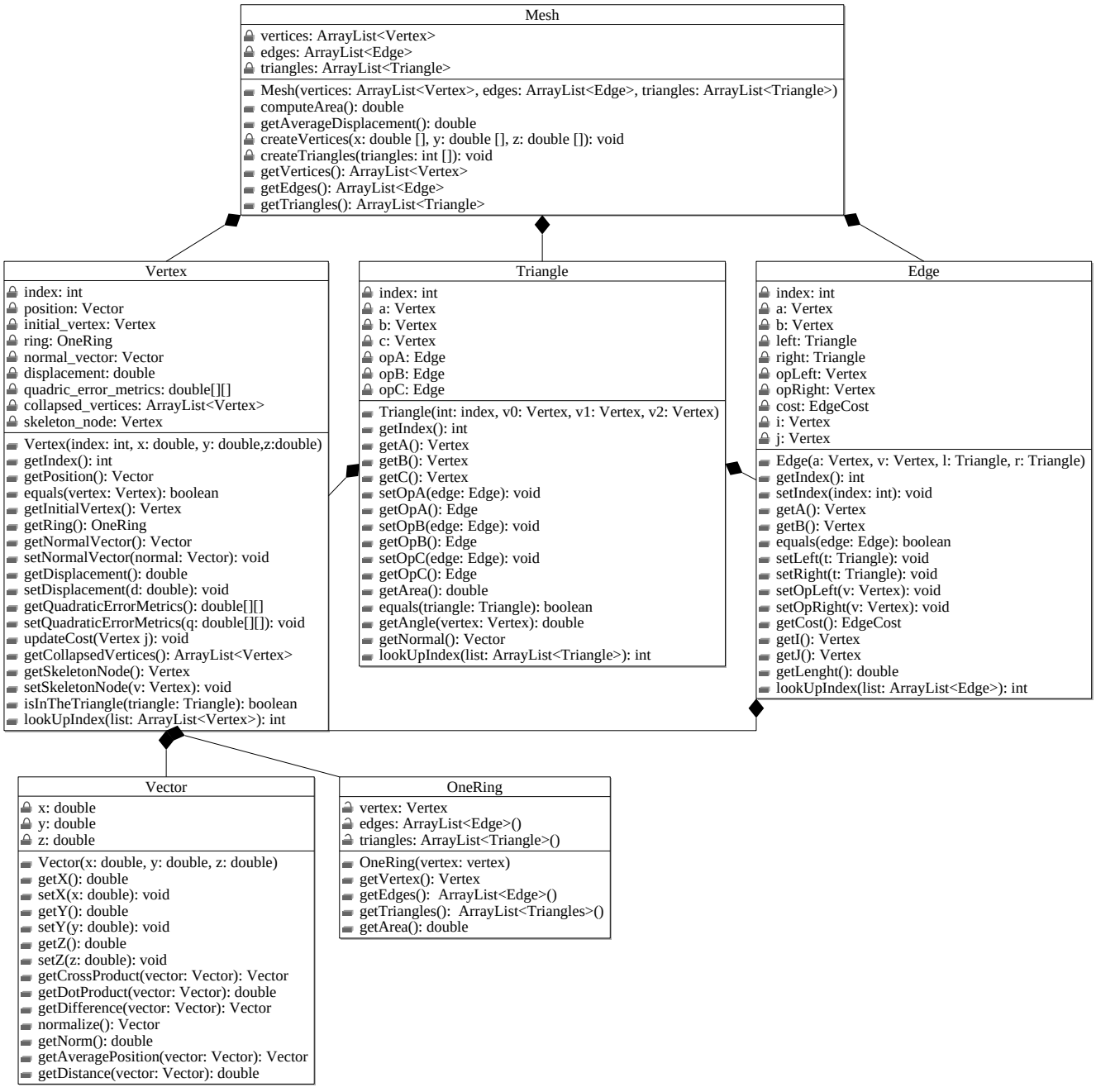


Figura 9.60: Diagrama de clases de las estructuras asociadas a la malla.

Figura 9.61: Diagrama de clases de las estructuras asociadas al *skeleton*.

Figura 9.62: Diagrama de clases de la etapa de contracción de la malla (*geometry contraction*).

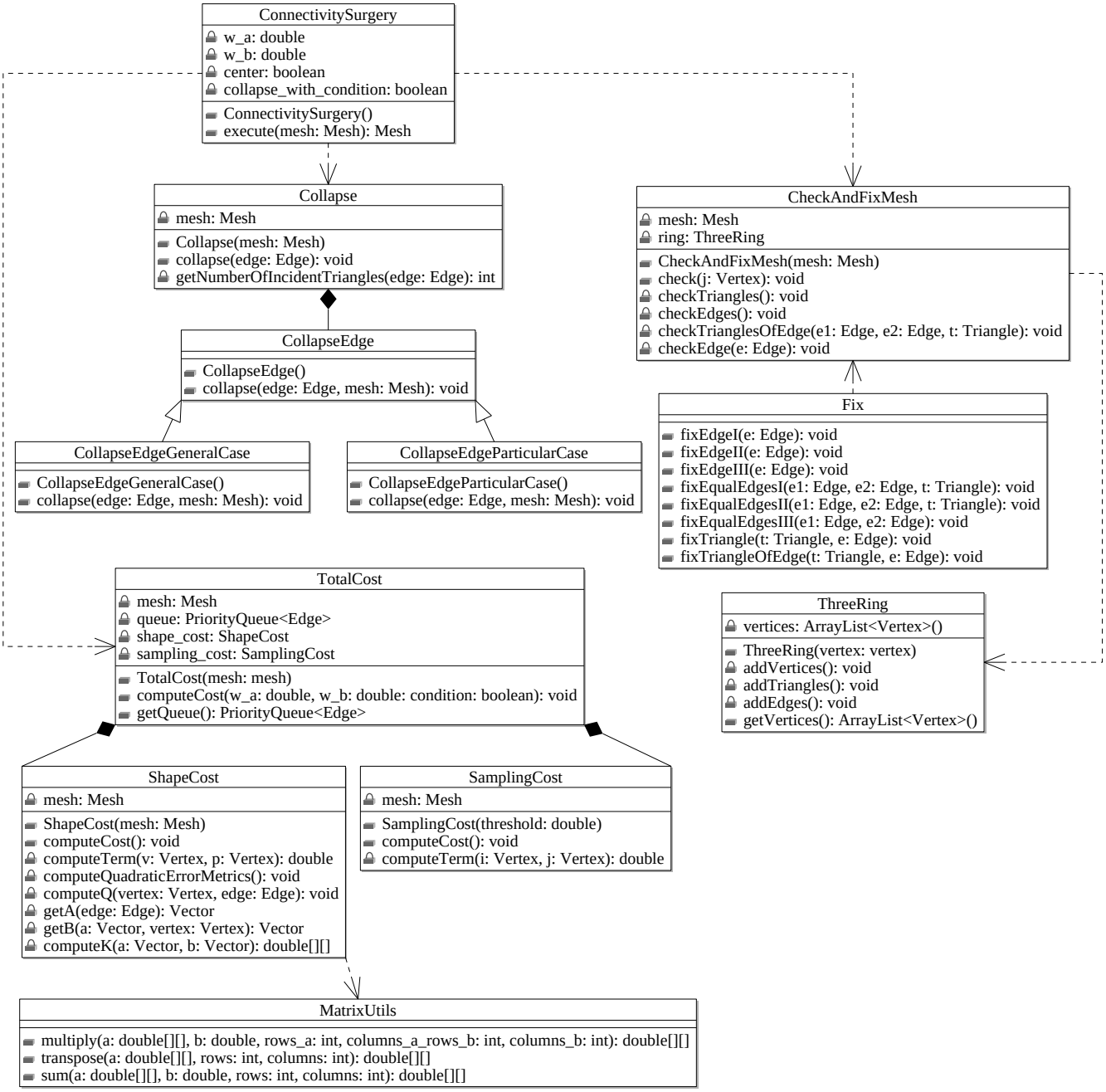


Figura 9.63: Diagrama de clases de la etapa de colapso de aristas (*connectivity surgery*).

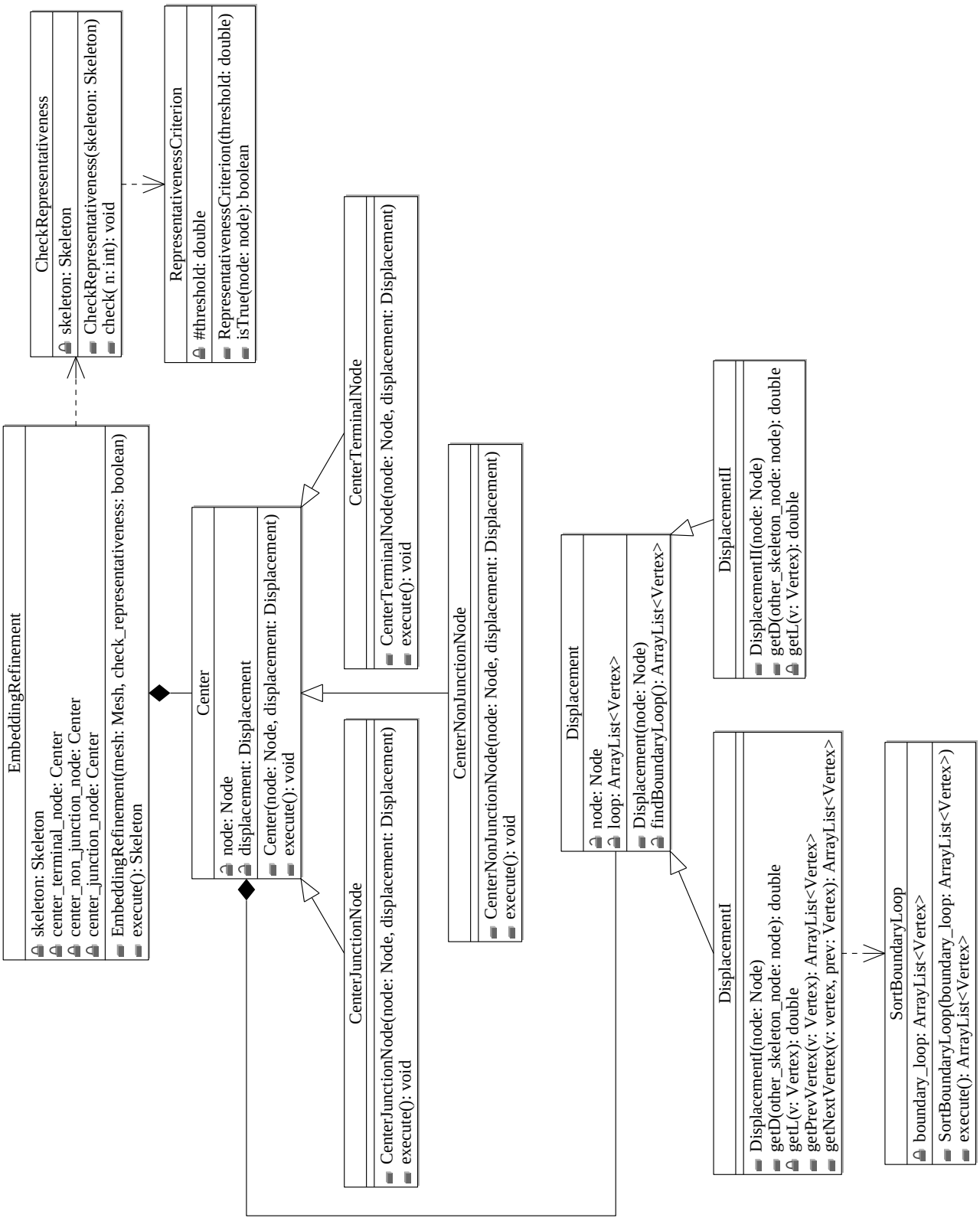


Figura 9.64: Diagrama de clases de la etapa de centrado del *skeleton* (*embedding refinement*).

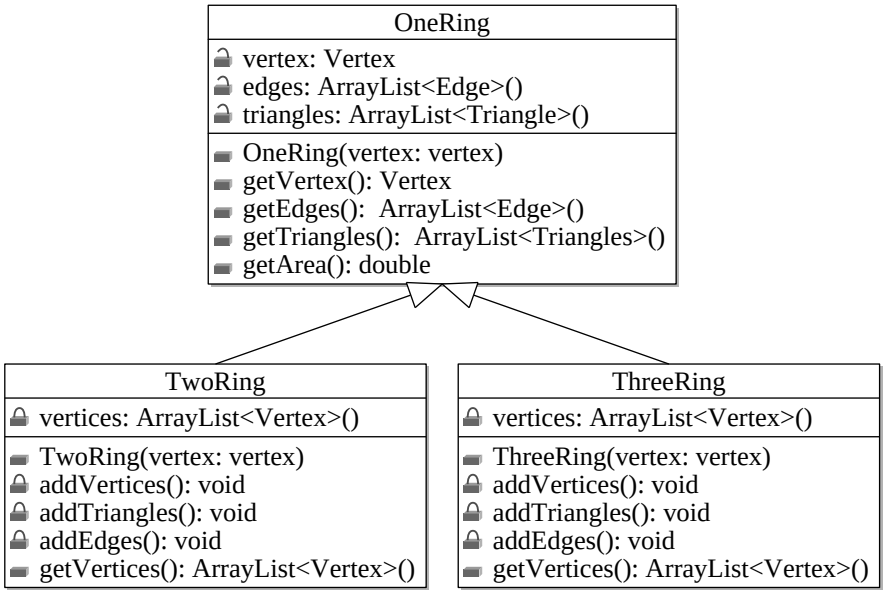


Figura 9.65: Diagrama de clases de los vecinos de un vértice (*one ring*) [13].

F . Código

Figura 9.66: Integración con IDL

Figura 9.67: Integración con IDL

Figura 9.68: Integración con IDL