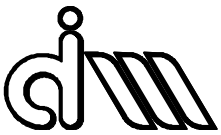
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Ingeniería Mecánica y de Materiales



Trabajo Fin de Máster en Ingeniería Mecánica

DETECCIÓN AUTOMÁTICA DE HUESO EN IMÁGENES MÉDICAS 3D A PARTIR DE SEGMENTACIÓN CON MALLADO ANISÓTROPO ADAPTATIVO

Presentado por: D. JUAN IGNACIO GARCÍA NICOLÁS
Dirigido por: Dr. D. JUAN JOSÉ RÓDENAS GARCÍA
Codirigido por: D. LUCA GIOVANELLI
Codirigido por: Dra. Dña. SIMONA PEROTTO

Valencia, Septiembre de 2017

AGRADECIMIENTOS:

Empezaré agradeciendo a mi director de tesis, Juanjo, quien me dio la oportunidad de formar parte de este trabajo y quien siempre se ha preocupado por encauzar el trabajo por buen camino. Tampoco me quiero olvidar de Fede y de Luca, quienes me han ayudado mucho con todos los quebraderos de cabeza que me ha dado Matlab.

Mi piacerebbe ringraziare a i miei tutore a Milano, Simona e Stefano, chi mi hanno aiudato tantissimo e hanno habuto pacienzia con tutte le probleme che ho habuto con FreeFem. Anche vorrebe salutare a tutta la gente che ho conosciuto a Milano e mi hanno dato così buone momenti.

Por último, me gustaría acordarme de mis amigos de mi tierra, Valencia, quienes indirectamente también me han ayudado en este trabajo, en especial, de mi madre y mi hermano.

RESUMEN:

En los análisis de Elementos Finitos a partir de imágenes médicas, obtenidas mediante TAC y resonancias magnéticas, existe una problemática debido a la igualdad de tonalidad de escala de color de distintas partes del cuerpo humano, por ejemplo, en ocasiones el hueso trabecular y la musculatura pueden presentar niveles de color similares, lo que conlleva indeterminaciones a la hora de asignar propiedades de material a los diferentes tejidos. Esta tesis de máster pretende obtener una aplicación que detecte automáticamente el hueso, u otras partes del cuerpo, en imágenes médicas para su posterior análisis mediante el cartesian grid Finite Element Method (cgFEM).

En la metodología desarrollada existe un preproceso, donde se preparan las imágenes para la segmentación. Posteriormente se realiza la segmentación propiamente dicha, donde se emplea un mallado anisótropo adaptativo bidimensional, a partir de los desarrollados del Departamento de Matemática del Politécnico di Milano. Finalmente, hay un postproceso, donde se construye una matriz lógica, a partir del mallado obtenido en la segmentación, que diferencia aquellas partes de la imagen que son útiles para el análisis con cgFEM de aquellas que no.

Las pruebas realizadas con distintas imágenes médicas devuelven resultados satisfactorios, como los que se detallan en la memoria, cumpliendo así el objetivo del trabajo.

Palabras clave: detección automática de hueso, método de los elementos finitos, cgFEM, Galerkin, segmentación, mallado anisótropo adaptativo, TAC, resonancia magnética, imagen médica, DICOM, PGM, escala Hounsfield, escala de grises, hueso cortical, hueso trabecular, matriz lógica, FreeFem++.

RESUM:

En els anàlisis de Elements Finits a partir de imatges mediques, obtingudes mitjançant TAC i ressonàncies magnètiques, existeix una problemàtica degut a la igualtat de tonalitat de escala de color en distintes parts del cos humà, per eixample, en ocasions l'os trabecular i la musculatura poden presentar nivells de color similars, implicant indeterminacions a la hora de assignar propietats de material als diferents teixits. Aquest treball de màster pretén obtenir una aplicació que detecti el os, o altres parts del cos, en imatges mediques per al seu posterior anàlisis mitjançant el cartesian grid Finite Element Method (cgFEM).

En la metodologia desenvolupada hi ha un preprocés, on se preparen les imatges mediques. Posteriorment es realitza la segmentació pròpiament dita, on s'emplea un mallat anisòtrop adaptatiu bidimensional, a partir de les desenvolupades en el Departament de Matemàtiques del Politècnic de Mila. Finalment, hi ha un post procés, on es construeix una matriu lògica, a partir del mallat obtingut en la segmentació, que diferencia aquelles parts de la imatge que son útils per als anàlisis de cgFEM de aquelles que no.

Les proves realitzades con distintes imatges mèdiques donen resultats satisfactoris, com els que se detallen en la memòria, complint aixina l'objectiu del treball.

Paraules clau: detecció automàtica d'os, mètode dels elements finits, cgFEM, Galerkin, segmentació, mallat anisòtrop adaptatiu, TAC, ressonància magnètica, imatges mèdiques, DICOM, PGM, escala Hounsfield, escala de grises, os cortical, os trabecular, matriu lògica, FreeFem++.

ABSTRACT:

In Finite Element analysis of medical images, obtained by CT Scan and magnetic resonance, there is a problem in the calculus because of the same color scale of different body parts, for example, sometimes the cancellous bone and the muscles have the same color value, involving indeterminations assigning material properties to the different tissues. This master's thesis tries to obtain an application that automatically detects the bone, or other body parts, in medical images for its subsequent analysis using the cartesian grid Finite Element Method (cgFEM).

In the methodology developed there is a preprocessing, where the images are prepared for the segmentation. Afterwards, a segmentation is done, obtained by a two-dimensional anisotropic mesh adaptation, developed in the Mathematics Department of the Polytechnic in Milano. Finally, there is a postprocessing, where a logical matrix is created, using the mesh in the segmentation, that distinguish between those parts of the image that are useful for its analysis with cgFEM and those that are not.

The tests done with different medical images shows satisfactory results, like those detailed in the report, achieving the thesis's goal.

Keywords: automatic bone detection, finite element method, cgFEM, Galerkin, segmentation, anisotropic mesh adaptation, CT Scan, magnetic resonance, medical images, DICOM, PGM, Hounsfield scale, gray scale, cancellous bone, cortical bone, logical matrix, FreeFem++.

Índice

MEMORIA

1. Precedentes

En la actualidad, la medicina personalizada al paciente está en auge respecto a la medicina generalizada, ya que permite realizar un tratamiento más específico. En ésta, destaca el uso de imágenes médicas obtenidas de resonancias magnéticas o tomografías axiales computarizadas (TAC). Estas imágenes pueden ser postprocesadas para evaluar biomarcadores de distintas patologías. Resulta incluso posible generar un modelo numérico del paciente que permita realizar simulaciones mediante el Método de los Elementos Finitos (MEF), lo que permitiría, por ejemplo, simular a priori la estabilidad que tendría un implante de cadera.

En el Departamento de Ingeniería Mecánica y Materiales de la Universidad Politécnica de Valencia, se ha desarrollado un procedimiento, especialmente indicado para simulaciones en estructuras óseas, de modelos numéricos del paciente a partir de imágenes médicas para su posterior simulación mediante el método de los elementos finitos con mallados cartesianos independientes de la geometría. El mallado se compone de dos fases, una primera donde se crea una malla de aproximación y una segunda donde se crea una malla de integración. La malla de aproximación solo necesita cubrir completamente la extensión del problema, como se muestra en la Figura 1. 1. Posteriormente, la malla de integración emplea estructuras jerárquicas de mallas, lo que permite reducir el coste computacional del proceso. Dentro de la jerarquía existen niveles de mallas de elementos cuadrados, donde el nivel cero es una malla con un único elemento y el resto de niveles tienen 2^{2n} elementos, siendo *n* el nivel de la malla. [5]

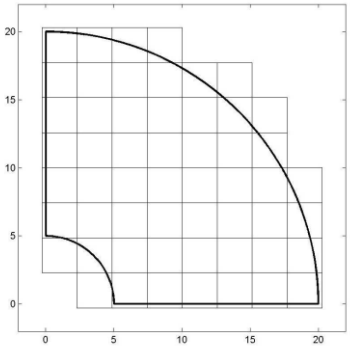


Figura 1. 1 Malla de aproximación

En la Figura 1. 2 se puede ver los distintos niveles en la jerarquía de mallas, mientras que en la Figura 1. 3 se puede ver la jerarquía de mallas superpuesta con un dominio.

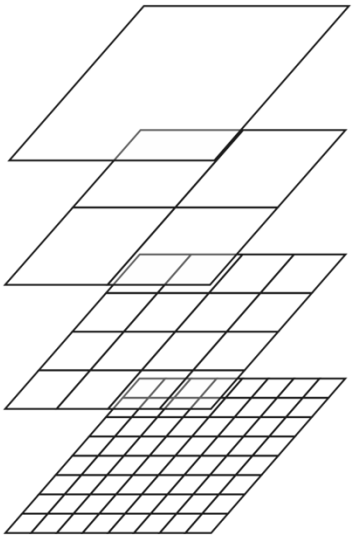


Figura 1. 2 Estructura jerárquica de mallas

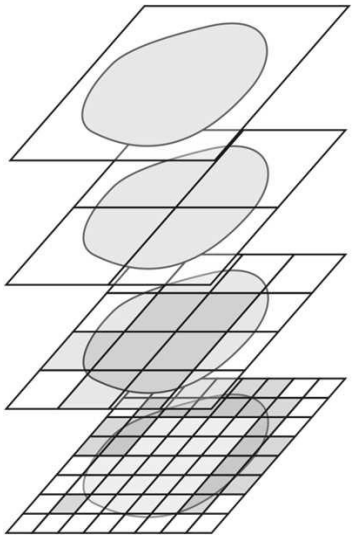


Figura 1. 3 Superposición con el dominio de cálculo

Uno de los mayores problemas en los cálculos, es el uso de la información de toda la imagen, lo que supone un tiempo de cálculo muy alto, cuando solo se necesita información de ciertas partes de la imagen, como por ejemplo aquellas referentes al hueso, tejido muscular o riñones, entre otros. La eliminación de las partes que no sean de interés para el análisis tiene como primer objetivo el de disminuir la cantidad de vóxeles de la imagen a considerar con el fin de reducir el número de elementos finitos a utilizar en los análisis. A diferencia de otros procesos de análisis mediante el MEF a partir de imágenes médicas, que requieren una gran precisión a la hora de definir los distintos tipos de tejido presentes en la imagen, el procedimiento utilizado por cgFEM hace que no resulte necesario determinar con mucha precisión los límites de dichos tejidos. Por otro lado, considerar tejidos que no resultan de interés para las simulaciones a realizar resulta problemático dado que distintos tejidos pueden estar representados por intensidades de color semejantes en la imagen. Esto daría lugar a indeterminaciones a la hora de asignar propiedades de material necesarias para realizar las simulaciones mediante el MEF.

Actualmente existen aplicaciones semiautomáticas que permiten distinguir las diferentes partes del cuerpo, pero que requieren la atención del usuario durante el proceso.

En el departamento de Matematica del Politecnico di Milano, los profesores Simona Perotto y Stefano Micheletti han desarrollado un procedimiento capaz de detectar el contorno de formas en imágenes 2D a partir de una segmentación con una malla de elementos finitos. Aplicando este procedimiento a imágenes médicas 3D, realizando la segmentación capa a capa, permite detectar el borde de diferentes partes del cuerpo humano, con lo que se puede analizar aquellas partes que se desee, desechando el resto y, por tanto, reduciendo el tiempo computacional. [1]

2. Objeto del trabajo

El presente trabajo tiene por título *Detección automática de hueso en imágenes médicas 3D a partir de segmentación con mallado anisótropo adaptativo*.

El objetivo del trabajo es el de obtener un procedimiento automático que detecte la estructura ósea en imágenes médicas 3D. Para ello se estudiará la viabilidad de adaptar el procedimiento de segmentación de imágenes 2D del departamento de Matematica del Politecnico di Milano, con el fin de obtener una matriz lógica que diferencie aquellas zonas de las imágenes que sean relevantes de aquellas que no lo sean.

El procedimiento incluirá 3 fases:

- Preproceso: partición y transformación de escala de color de las imágenes al formato PGM.
- Segmentación: mallado anisótropo de las imágenes para la detección de bordes.
- Postproceso: obtención de una matriz lógica que diferencie los píxeles relevantes del resto dadas las mallas obtenidas en la segmentación.

La base del trabajo se ha programado con el software comercial *Matlab* en su versión 16, mientras que para la segmentación se ha empleado el software libre *FreeFem++.*

El trabajo se ha realizado como Trabajo Final del Máster de Ingeniería Mecánica para el Departamento de Ingeniería Mecánica y Materiales de la Universidad Politécnica de Valencia, en colaboración con el Departamento de Matemáticas del Politécnico de Milano.

3. El método de los elementos finitos

El método de los elementos finitos (MEF) es una herramienta muy utilizada en ingeniería y matemáticas para dar solución a problemas estructurales, de transferencia de calor o de dinámica de fluidos, entre otros. MEF discretiza el sistema continuo, por lo que la solución obtenida no es exacta. No obstante, se puede aproximar a la exacta incrementando el grado de refinamiento del modelo.

Existen varios puntos de vista que desarrollan el MEF, entre los más importantes están el método de Rayleigh-Ritz y el método de Galerkin. El programa que realiza la segmentación en este trabajo, FreeFem++, emplea el método de Galerkin, por lo que es el que se va a explicar a continuación. [2]

El procedimiento a desarrollar se basa en que las imágenes médicas 3D están formadas por una sucesión de imágenes 2D a las que resulta posible aplicar el procedimiento de segmentación desarrollado en el Politécnico de Milano, que ha sido implementado para el caso 2D. Por lo tanto, toda la teoría descrita a continuación hace referencia al caso bidimensional.

3.1. Ecuaciones elípticas

Se considera un dominio $\Omega \subset R^2$ limitado por su frontera $\partial\Omega$ como el mostrado en la Figura 3.1, donde se pueden dar las siguientes condiciones de contorno [2]:

- Neumann, $\nabla u \cdot n = \frac{\partial u}{\partial n} = g$ en Γ_N
- Dirichlet, $u = h$ en Γ_D

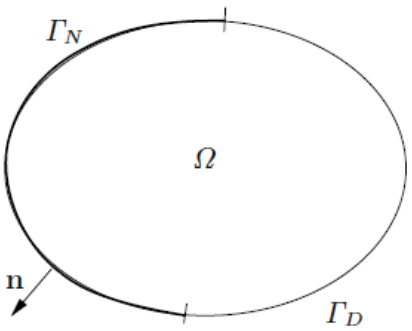


Figura 3. 1 Condiciones de contorno

Se define el problema de Poisson según su formulación fuerte en la Ec.1:

$$\begin{aligned} -\Delta u &= f \text{ en } \Omega \\ u &= 0 \text{ en } \Gamma_D \\ \frac{\partial u}{\partial n} &= g \text{ en } \Gamma_N \end{aligned} \qquad \text{Ec. 1}$$

El objetivo es obtener la solución $u \in V$ que satisfaga la ecuación 1. Donde V es un espacio infinito de Hilbert y $H^1(\Omega)$ es un subespacio del espacio V.

Para ello se transforma la formulación fuerte, en una formulación más manejable que recibe el nombre de formulación débil, lo que se conoce como formulación variacional. Primeramente, se aplica la fórmula de Green, en la ecuación 2:

$$-\int_{\Omega} \Delta u \cdot v \, d\Omega = \int_{\Omega} \nabla u \cdot \nabla v \, d\Omega - \oint_{\partial\Omega} \frac{\partial u}{\partial n} \cdot v \, d\gamma \qquad \text{Ec. 2}$$

Donde el término $\oint_{\partial\Omega} \frac{\partial u}{\partial n} \cdot v \, d\gamma$ hace referencia a las condiciones de contorno del problema y v es una función arbitraria o función test que debe cumplir las siguientes condiciones:

$$\begin{aligned} H^1(\Omega) &= \left\{ v: \Omega \to R \;\; tal\ que \;\; v \in L^2 \; y \;\; \frac{\partial u}{\partial n} \in L^2(\Omega) \right\} \\ H_0^1(\Omega) &= \{ v \in H^1(\Omega) \;\; tal\ que \;\; v = 0 \;\; en \;\; \partial\Omega \} \end{aligned} \qquad \text{Ec. 3}$$

La primera condición se refiere a la existencia de la función v en el dominio Ω. Mientras que la segunda exige que la función sea nula en la frontera $\partial\Omega$.

Con todo ello se construye la formulación débil.

$$\int_{\Omega} \nabla u \cdot \nabla v \, d\Omega = \int_{\Omega} f \cdot v \, d\Omega + \int_{\Gamma_N} \frac{\partial u}{\partial n} \cdot v \, dS \qquad \text{Ec. 4}$$

En la ecuación anterior se define:

- $a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v \, d\Omega$ como la forma bilineal del espacio VxV en R
- $F(v) = \int_{\Omega} f \cdot v \, d\Omega + \int_{\Gamma_N} \frac{\partial u}{\partial n} \cdot v \, dS$ como la forma lineal de V en R

En el caso de haber términos no homogéneos, la formulación fuerte vendría reflejada como se muestra en la ecuación 5:

$$\begin{gathered} -div(\mu \cdot \nabla u) + b \cdot \nabla u + \sigma \cdot u = f \text{ en } \Omega \\ u = 0 \text{ en } \Gamma_D \\ \frac{\partial u}{\partial n} = g \text{ en } \Gamma_N \end{gathered} \qquad \text{Ec. 5}$$

Donde:

- El término $-div(\mu \cdot \nabla u)$ hace referencia a la difusión.
- El término $b \cdot \nabla u$ hace referencia al transporte o convección.
- El término σu hace referencia a la absorción o reacción.

En este caso, la formulación débil vendría representada por la ecuación 6:

$$\begin{aligned} \int_\Omega \mu \cdot \nabla u \cdot \nabla v \, d\Omega + \int_\Omega b \cdot \nabla u \cdot v \, d\Omega + \int_\Omega \sigma \cdot u \cdot v \, d\Omega \\ = \int_\Omega f \cdot v \, d\Omega + \int_{\Gamma_N} \frac{\partial u}{\partial n} \cdot v \, dS \end{aligned} \qquad \text{Ec. 6}$$

Definiendo de nuevo:

- $a(u,v) = \int_\Omega \mu \cdot \nabla u \cdot \nabla v \, d\Omega + \int_\Omega b \cdot \nabla u \cdot v \, d\Omega + \int_\Omega \sigma \cdot u \cdot v \, d\Omega$
- $F(v) = \int_\Omega f \cdot v \, d\Omega + \int_{\Gamma_N} \frac{\partial u}{\partial n} \cdot v \, dS$

En aplicaciones con términos no homogéneos, resulta interesante definir el número de Peclet, esta variable define la ratio de convección por la ratio de difusión.

$$Pe \propto \frac{b}{\mu} \qquad \text{Ec. 7}$$

El número de Peclet indica la inestabilidad del sistema, a mayor número de Peclet, mayor inestabilidad.

3.2. Método Galerkin

El método de Galerkin define un espacio finito dentro del espacio infinito de Hilbert V. Se define V_h como la familia de espacios finitos dependientes del parámetro h, siendo este el tamaño del elemento.

El objetivo es encontrar la solución aproximada u_h perteneciente al espacio finito V_h, tal que se cumpla la ecuación $a(u_h, v_h) = F(v_h)$.

El problema de Galerkin define una base φ_i, donde $i = 1,2,3, \dots, N_h$, siendo N_h la dimensión del espacio V_h, de forma que se cumple la ecuación 8:

$$a(u_h, \varphi_i) = F(\varphi_i) \qquad \text{Ec. 8}$$

Con lo que se obtiene $v_h = \varphi_i$.

Dado que u_h existe en V_h, la solución se puede obtener interpolando la solución en los nodos, como se demuestra en la ecuación 9:

$$u_h(x) = \sum_{j=1}^{N_h} u_j \cdot \varphi_j(x) \qquad \text{Ec. 9}$$

Donde x son las coordenadas espaciales, u_j es la solución en los nodos y $u_h(x)$ es la solución de cualquier punto dentro de este espacio.

Sustituyendo la ecuación 8 en la formulación débil de la ecuación 4 se obtiene:

$$\int_\Omega \left[\sum_{j=1}^{N_h} u_j \cdot \varphi_j(x) \right] \cdot \nabla v_h \, d\Omega = \int_\Omega f \cdot v_h \, d\Omega + \int_{\Gamma_N} g \cdot v_h \, dS \qquad \text{Ec. 10}$$

Reordenando los términos:

$$\sum_{j=1}^{N_h} u_j \cdot \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\Omega = \int_\Omega f \cdot \varphi_i \, d\Omega + \int_{\Gamma_N} g \cdot \varphi_i \, dS \qquad \text{Ec. 11}$$

Se define la matriz de rigidez A como una matriz definida positiva y en banda, donde cada término de la matriz se evalúa como $a_{ij} = \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\Omega \in R^{N_h x N_h}$.

Se define el vector de fuerzas f como la suma de las fuerzas externas y las internas, donde cada término del vector $f_i = \int_\Omega f \cdot \varphi_i \, d\Omega + \int_{\Gamma_N} g \cdot \varphi_i \, dS \in R^{N_h}$.

Se define el vector solución u como la solución en los nodos, donde cada término del vector $u_j = u_h(N_j) \in R^{N_h}$.

En la ecuación 11 se puede escribir en forma matricial como:

$$[A] \cdot \{u\} = \{f\} \qquad \text{Ec. 12}$$

3.3. Métrica

Dado un dominio Ω y un punto arbitrario de este dominio x, se define la métrica $M(x)$ de este punto como una matriz simétrica y positiva. [3]

$$M(x) = \begin{pmatrix} a_x & b_x \\ b_x & d_x \end{pmatrix}$$

En la construcción de triangulaciones, la métrica es un operador muy importante ya que indica la dirección que deben seguir los elementos y el tamaño de éstos respecto a su dirección.

Se define una métrica isotrópica cuando tiene control del tamaño de los elementos en cualquier punto del espacio. Se define una métrica anisótropa cuando tiene control del tamaño y direccionalidad de los elementos.

La construcción de la métrica se realiza a partir de la información de los elementos de la triangulación.

Supóngase un elemento de referencia $\widehat{K}$, un triángulo equilátero como el de la Figura 3. 2. Este elemento está contenido en una circunferencia. Al deformar el elemento de referencia en un elemento real K como el de la Figura 3. 3, la circunferencia donde estaba contenido también se deforma transformándose en una elipse. La información de esta elipse viene definida por los vectores correspondientes a los ejes de la elipse, r_1 y r_2, donde r_1 será el mayor, y por los pesos de estos vectores, λ_{1k} y λ_{2k}, siendo el tamaño de los semiejes.

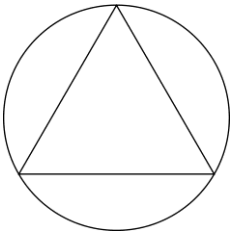


Figura 3. 2 Elemento de referencia dentro de una circunferencia

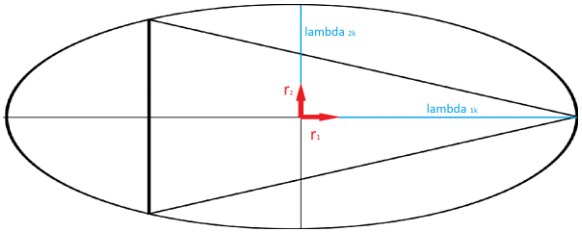


Figura 3. 3 Elemento dentro de una elipse

Esta información se guarda en los elementos a_x, b_x y d_x de la métrica $M(x)$. El hecho de que haya cuatro variables de la elipse para tres elementos de la métrica es porque las variables de la elipse son dependientes entre sí ya que los ejes r_1 y r_2 son perpendiculares entre sí.

Del mismo modo que se construye la métrica a partir de la información de la elipse del triángulo, también se puede construir las elipses del triángulo a partir de la información de la métrica.

En un mallado adaptativo resulta muy útil construir elementos conforme a la métrica elegida, donde cada iteración sigue la siguiente pauta:

1) Se parte de una malla o triangulación inicial $\mathcal{T}_i$.
2) Se construye la métrica M_i a partir de la triangulación $\mathcal{T}_i$.
3) Se construye la triangulación $\mathcal{T}_{i+1}$ a partir de la métrica creada.

3.4. Mallado anisótropo adaptativo

Esta técnica ha sido desarrollada por los profesores del departamento de Mátematica del Politecnico di Milano, Simona Perotto y Stefano Micheletti. [1]

Resumiendo la teoría explicada a continuación, el objetivo de esta técnica es pasar de una malla inicial de triángulos a una malla final, estirando los triángulos en aquellas zonas donde existe una variación importante en el color de los píxeles, es decir, en los contornos. La técnica forma parte de un proceso iterativo, donde cada iteración se va adaptando a la geometría de la imagen, la precisión de la malla, en cada iteración, depende de una tolerancia, que se indica como τ. El proceso itera hasta que alcanza la convergencia, definida con una tolerancia que se indica como *Tol*.

En el siguiente ejemplo se pretende demostrar el resultado de aplicar esta técnica a una imagen, Figura 3. 4, mostrando las diferentes iteraciones del proceso adaptativo. Se observa como en cada iteración la malla, representada en las figuras Figura 3. 5, Figura 3. 6 y Figura 3. 7, se va ajustando al contorno de la imagen original empleando elementos estirados en los contornos.

Figura 3. 4 Imagen original

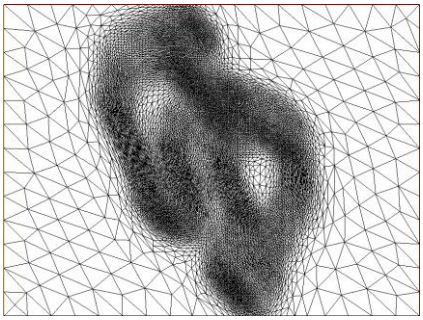


Figura 3. 5 Primera iteración

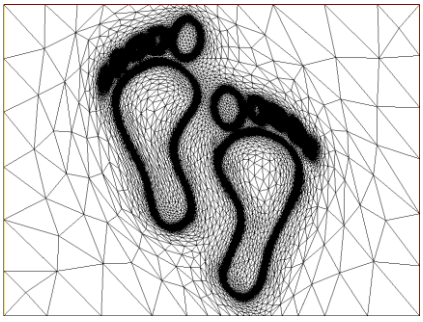


Figura 3. 6 Segunda iteración

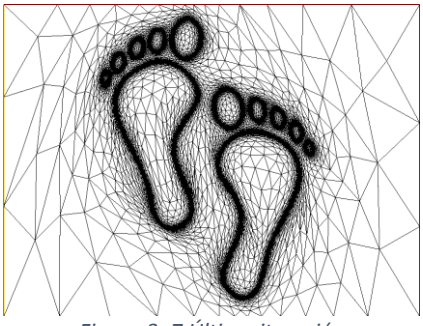


Figura 3. 7 Última iteración

En este trabajo, se propone un estimador de error Zienkiewicz-Zhu a posteriori en 2D. La novedad reside en la inclusión de la información geométrica de la malla, útil para un mallado anisótropo adaptativo. Pese a su comportamiento heurístico, el estimador devuelve resultados satisfactorios.

Se distinguen tres partes: procedimiento de recuperación del gradiente, estimador anisótropo y procedimiento adaptativo.

3.4.1. Procedimiento de recuperación del gradiente

Dado un problema no homogéneo ADR (advección, difusión y reacción) como el de la ecuación 6, se define $\mathcal{T}_h = \{K\}$ como la partición o malla triangular de Ω, u_h como la solución del método de Galerkin al problema, y ∇u_h como el gradiente de la solución. La solución u_h puede ser desde el flujo de un fluido hasta la escala de grises de una imagen.

Se define el *patch* Δ_K de un elemento K como el conjunto de elementos que comparten al menos un nodo con K.

Se propone un gradiente recuperado $P^r_{\Delta_K}(\nabla u_h)$ con un grado *r* sobre el *patch* Δ_K, tal como se ve en la ecuación 13:

$$\int_{\Delta_K} \left(\nabla u_h - P^r_{\Delta_K}(\nabla u_h)\right) \cdot w \cdot dx = 0 \qquad \text{Ec. 13}$$

Para el caso particular $r = 0$, el gradiente recuperado del elemento K se calcula como el gradiente medio de los elementos del *patch* Δ_K multiplicado por el peso de cada elemento, siendo este el área, tal y la ecuación 14:

$$P^0_{\Delta_K}(\nabla u_h) = \frac{1}{|\Delta_K|} \sum_{T \in \Delta_K} |T| \cdot \nabla u_h|_T \qquad \text{Ec. 14}$$

3.4.2. Estimador anisótropo

Se propone un estimador de error anisótropo a posteriori, siendo $E^r_{\Delta_K} = P^r_{\Delta_K}(\nabla u_h) - \nabla u_h$ la aproximación del error en función del gradiente.

Se define el estimador local para cada elemento K, tal como se muestra en la ecuación 15:

$$\left[\eta^r_{K,A}\right]^2 = \frac{1}{\lambda_{1,K} \cdot \lambda_{2,K}} \sum_{i}^{2} \lambda^2_{i,K} \cdot (r^T_{i,K} \cdot G_{\Delta_K}(E^r_{\Delta_K}) \cdot r_{i,K}) \qquad \text{Ec. 15}$$

Donde $r_{1,K}$, $r_{2,K}$, $\lambda_{1,K}$ y $\lambda_{2,K}$ hacen referencia a la orientación y longitud de los ejes de la elipse donde está contenido el elemento K y $G_{\Delta_K}(\cdot)$ hace referencia a un operador similar al Hessiano.

Se define el estimador global, tal como se expresa en la ecuación 16:

$$[\eta_A^r]^2 = \sum_{K \in \mathcal{T}_h} \left[\eta_{K,A}^r\right]^2 \qquad \text{Ec. 16}$$

3.4.3. Procedimiento adaptativo

Se realiza el procedimiento adaptativo en base a la métrica que se obtiene a partir del estimador η_A^r. El objetivo es el de obtener una métrica M_K capaz de generar una malla que minimice el estimador $\eta_{K,A}^r$ respecto a la orientación y longitud, $r_{i,K}$, $\lambda_{i,K}$, mediante un criterio de equidistribución del error.

El estimador $\eta_{K,A}^r$ se obtiene a partir de la cardinalidad de la malla inicial, $\#\mathcal{T}_h$, y una tolerancia fijada, τ, tal y la ecuación 17:

$$\eta_{K,A}^r{}^2 = \frac{\tau^2}{\sqrt{\#\mathcal{T}_h}} \qquad \text{Ec. 17}$$

A partir del estimador $\eta_{K,A}^r$ se obtienen los parámetros de la elipse $r_{i,K}$, $\lambda_{i,K}$ con los que se construye la métrica M_K. Esta métrica viene definida para cada elemento K, pero se debe definir para cada nodo N ya que FreeFem trabaja con información de los nodos.

4. Consideraciones previas

Este trabajo tiene como objetivo acelerar los procesos de generación del modelo de EF y de análisis de estructuras óseas descritas mediante una imagen médica 3D. Resulta necesario eliminar la información de la imagen médica que no corresponda a las estructuras óseas dado que no son de interés para los problemas que se desean analizar. Se pretende, por tanto, crear una matriz lógica que permita seleccionar las zonas de la imagen 3D a tener en cuenta. Normalmente estos procesos exigen el uso de complejas herramientas de software de segmentación y, en ocasiones, de expertos en anatomía. Se pretende desarrollar un procedimiento que permita eliminar de la imagen 3D las zonas que no correspondan a estructuras óseas, aunque no se exige que esta eliminación resulte muy precisa. Aun cuando la reducción del coste computacional resulta un factor muy importante, se primará la automatización del proceso, de manera que el proceso no requiera el uso de expertos para realizar la segmentación.

Se utilizarán matrices 3D que contienen mucha información y, por tanto, muy grandes. El almacenamiento de toda la información en memoria puede afectar muy negativamente al rendimiento computacional del proceso, por lo que, en la medida de lo posible, se mantendrá en memoria la mínima parte de información posible.

Una de las consideraciones previas que se han realizado es la de seleccionar aquellos píxeles cuyo valor sea característico del hueso, de esta forma se obtendría una matriz con estos píxeles. No obstante, es posible que existan píxeles con el mismo nivel de color del hueso sin que sean hueso o el caso opuesto, píxeles que no tengan el mismo nivel de color que el hueso pero sean útiles y se desechen. Por este motivo se ha optado por realizar un mallado anisótropo adaptativo bidimensional, que permita determinar los bordes de las estructuras óseas para, posteriormente, seleccionar los píxeles que se encuentren hacia el interior de dichos bordes.

La segmentación realiza un mallado bidimensional, no tridimensional. Por tanto, para obtener la matriz lógica tridimensional, se realiza la segmentación por cortes en el plano especificado. Se propone realizar el proceso para cada uno de los planos de la matriz, *XY*, *YZ* y *XZ*, para mayor confiabilidad del proceso.

4.1. Imágenes médicas

Las imágenes con las que se van a trabajar suelen estar en formato DICOM en unidades de escala Hounsfield. Se define a continuación algunos términos de interés para el trabajo.

4.1.1. Resolución de las imágenes

La resolución de las imágenes obtenidas mediante TAC depende principalmente del tamaño y número de los elementos detectores y la distancia entre corte y corte.

Cada imagen tiene tantos píxeles en cada dimensión como canales tiene los elementos detectores de la máquina que realiza el TAC. Por ejemplo, una máquina con 1024 canales devuelve imágenes con una resolución de 1024x1024.

La tercera dimensión se consigue generando cortes cada cierta distancia. Actualmente, la resolución suele ser del orden de 1-2 mm para objetos del rango de metros a decímetros. Para objetos del rango de decímetros a centímetros, se utiliza maquinaria "*High-resolution*" con resolución del orden de 100-200 micrómetros. Para objetos del rango de centímetros a milímetros, se utiliza maquinaria "*Ultra-high-resolution*" con resolución del orden de micrómetro.

La microtomografía, actualmente en desarrollo, trabaja dentro del rango de micrómetro. Esta técnica, utiliza una gran cantidad de radiación, por lo que no es muy empleada como herramienta de diagnóstico. (3)

4.1.2. DICOM

DICOM (*Digital Imaging and Comumunications in Medicine*) es un tipo de almacenamiento de imágenes médicas obtenidas mediante tomografías axiales computarizadas (TAC), radiografías, ultrasonidos y resonancias magnéticas entre otros. Es muy utilizado en medicina ya que permite disponer datos del paciente. (6)

4.1.3. Escala Hounsfield

La escala Hounsfield es el resultado de la transformación de la escala de coeficientes de atenuación lineal de rayos X en una nueva escala en la cual el valor de atenuación del agua destilada en condiciones normales de presión y temperatura se define como 0 unidades Hounsfield (HU), mientras que la atenuación del aire en condiciones normales de presión y temperatura se define como -1000 HU. Esta escala es muy utilizada en aplicaciones médicas, como radiografías, debido a que tiene un rango bastante amplio que permite diferenciar entre partes del cuerpo humano, como tejido muscular, hueso, grasa, etc.

Los valores son obtenidos experimentalmente, por lo que según la fuente pueden variar. A continuación, se muestran dos fuentes de valores en la Tabla 4. 1 (1) y en la Figura 4. 1 (4):

Tabla 4. 1 Escala Hounsfield (1)

Substance	HU
Air	−1000
Lung	−500
Fat	−120 to −90
Water	0
Urine	-5 to +15
Bile	-5 to +15
Cerebrospinal fluid	15
Kidney	+20 to +45
Blood	+30 to +45
Muscle	+35 to +55
Grey matter	+37 to +45
White matter	+20 to +30
Liver	+40 to +60
Soft Tissue, Contrast	+100 to +300
Bone	+700 (cancellous bone) to +3000 (cortical bone)

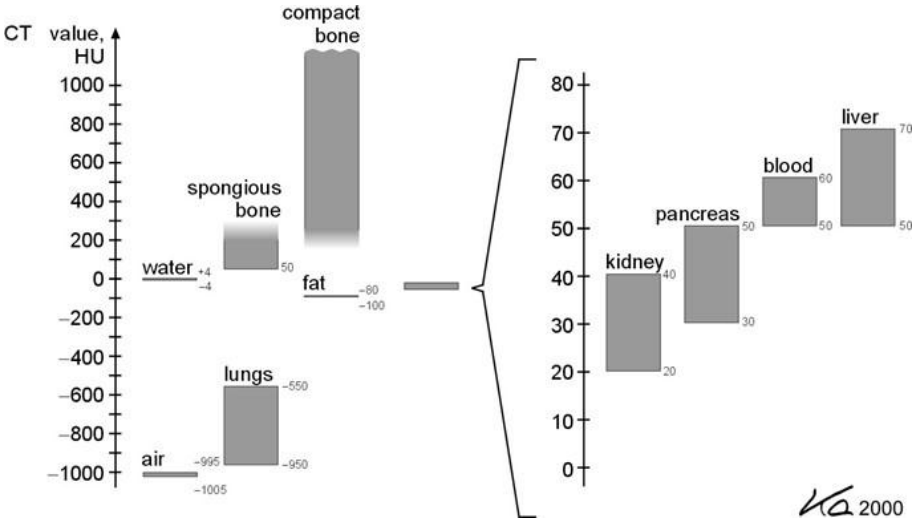


Figura 4. 1 Valores de la escala Hounsfield (4)

4.1.4. PGM

PGM (*Portable GrayMap*) es un formato de imágenes que las almacena en escala de grises.

Las imágenes pueden guardarse en código *ASCII* (representado como *P2*) o *raw* (representado como *P5*). El primero es mucho más sencillo de leer y de transferir entre programas, mientras que el segundo es un código binario, mucho más eficiente en el tamaño de archivos, pero más difícil de transferir entre programas. (2)

4.2. Freefem++

Freefem++ es un software libre, desarrollado por Olivier Pironneau de la Universidad Pierre et Marie Curie, que se centra en la resolución de ecuaciones diferenciales parciales (EDP) usando el método de los elementos finitos. Tiene su propio lenguaje, basado en *C++*. (5)

Los archivos de FreeFem son ficheros de texto con extensión *.edp*, y se ejecutan, en Windows, en la ventana de comandos CMD.

FreeFem solo admite imágenes en escala de grises de 255 valores, en el formato PGM y en código *ASCII*. Hay que tener en cuenta que FreeFem transpone estas imágenes.

5. Preproceso

Durante el preproceso se preparan las imágenes para que la segmentación sea lo más eficiente posible. Es un apartado importante ya que influye en la segmentación y el postproceso.

En esta parte del trabajo se extraen capas 2D de las imágenes médicas 3D, de forma que durante el proceso se trabajará capa a capa, se cambia la escala de color, se hace una partición de las capas y se guardan a formato PGM. Cada una de estas fases se describe a continuación.

5.1. Cambio de escala de color

FreeFem++ solo puede leer imágenes en la escala de grises de 0 a 255, mientras que las imágenes médicas suelen venir en otra escala, como por ejemplo la escala Hounsfield, con un rango de valores entre -1000 y 3000. Por lo tanto, se debe realizar un cambio de escala de color.

Para acentuar los bordes, se elige un rango de valores de la escala Hounsfield para convertirlo a la escala de grises. Este rango de valores está comprendido entre un valor máximo y un valor mínimo, y se encuentra la información de las partes del cuerpo que se deseen, de acuerdo con los valores de la Tabla 4. 1 y en la Figura 4. 1. Todo ello se realiza la siguiente operación:

$$V_{GS} = \frac{V_{HF} - V_{min}}{V_{max} - V_{min}} \cdot 255$$

Donde:

- V_{max}, valor máximo.
- V_{min} valor mínimo.
- V_{HF}, valor en la escala Hounsfield de cada píxel.
- V_{GS}, valor en la escala de grises de cada píxel.

En el Gráfico 5. 1 se ha realizado un cambio de escala de color con valores de la escala Hounsfield comprendidos entre 100 y 700.

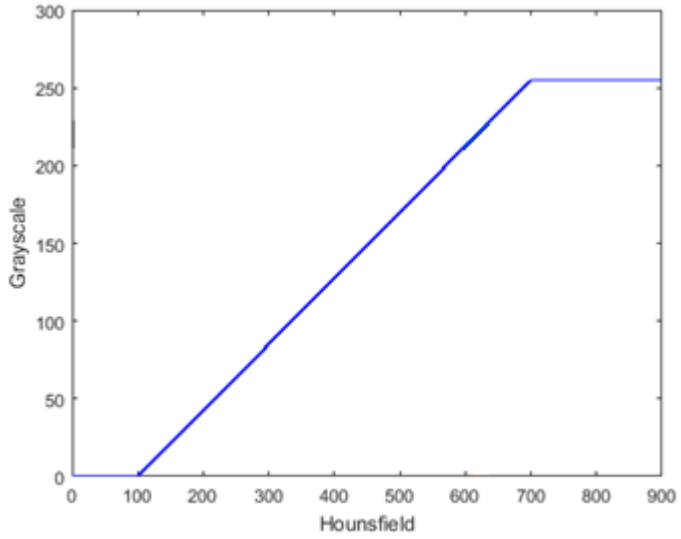


Gráfico 5. 1 Cambio de escala de color

Es interesante destacar que realizando este filtrado de imagen se simplifica el trabajo de la segmentación ya que se acentuará el contraste de los bordes de aquellas partes del cuerpo dentro del rango elegido. Esto se puede observar en la Figura 5. 1 y la Figura 5. 2, donde en la primera no se ha filtrado la imagen y en la segunda sí. Además, se puede apreciar el contraste respecto a la primera imagen.

Figura 5. 1 Matriz sin filtrar

Figura 5. 2 Matriz filtrada

5.2. Partición de la imagen

Como se ha comentado, se trabaja con imágenes 3D de las que se extraen capas 2D para realizar el proceso. Resulta interesante partir las capas para que la segmentación no tenga que trabajar con tanta información y se pueda paralelizar el proceso. A continuación, se pueden observar tres ejemplos donde se ha hecho una partición 2x2 en la figura 5.3, una partición 4x4 en la figura 5.4 y una partición 2x4 en la figura 5.5.

Figura 5. 3 Imagen dividida en 2x2

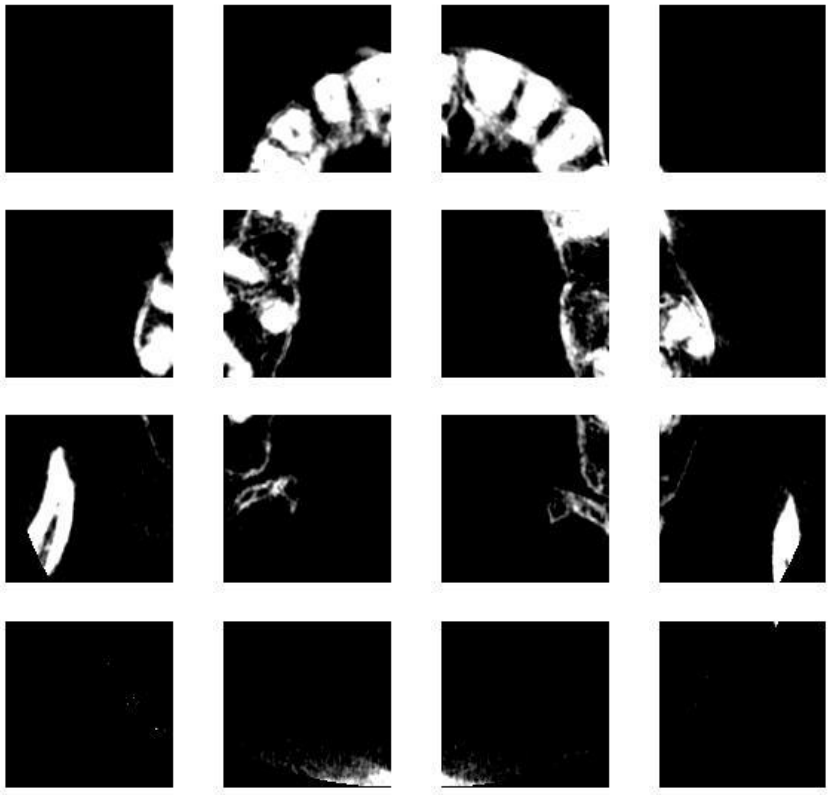


Figura 5. 4 Imagen dividida en 4x4

Figura 5. 5 Imagen dividida en 2x4

El número óptimo de divisiones depende en parte, del número de procesadores del ordenador. Este número se estima que será múltiplo del número de procesadores, ya que cada procesador trabajará aproximadamente con el mismo número de imágenes.

6. Segmentación

La segmentación es el núcleo fundamental de este trabajo, en ella se genera las mallas a partir de las imágenes creadas en el preproceso. Esta parte del trabajo es la más costosa, tanto de implementación como computacionalmente.

En la segmentación se obtiene un mallado anisótropo adaptativo a partir de la teoría desarrollada por S. Perotto y S. Micheletti [1]. El procedimiento se explica en el apartado 3.4. Mallado anisótropo adaptativo.

En los bordes, los elementos de la malla aparecen estirados, como se puede ver en la Figura 6. 1 del siguiente ejemplo.

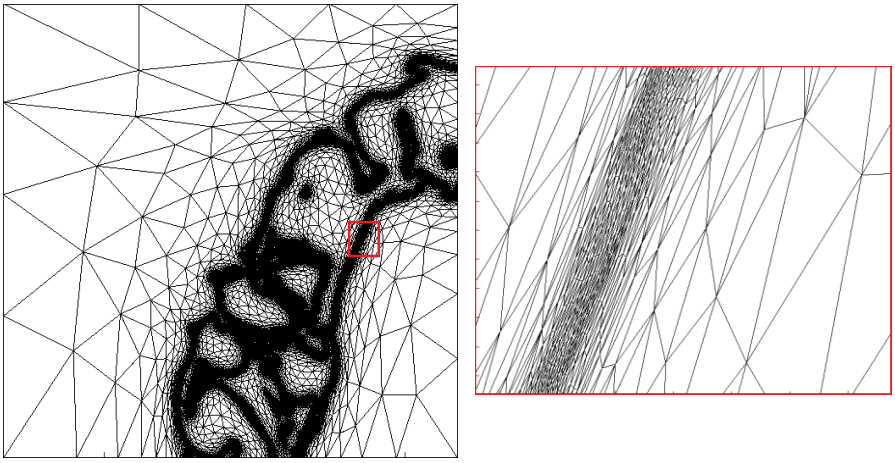


Figura 6. 1 Mallado y detalle del borde

6.1. Tolerancias

Como en todo proceso iterativo, debe existir una tolerancia que indique que se ha alcanzado la convergencia. En el proceso de segmentación existirán dos tolerancias a las que se referirán como *Tol* y τ:

- *Tol* es la tolerancia usada para determinar si el proceso adaptativo ha alcanzado la convergencia, luego su valor influirá sobre el número de iteraciones del proceso.
- τ es la precisión de la malla en cada iteración del proceso adaptativo. Esta tolerancia viene definida en el apartado 3.4.3 Procedimiento adaptativo.

Es importante elegir un valor adecuado para las tolerancias ya que, un valor muy bajo implica mayor tiempo y coste computacional, mientras que un valor muy alto implica poca precisión del mallado. El objetivo de la segmentación es encontrar aquella malla que represente los bordes en el menor tiempo posible.

A continuación, se van a exponer dos ejemplos con diferentes casos para mostrar los resultados de diferentes valores para cada una de las tolerancias.

En el primer ejemplo, se van a suponer tres casos de mallado variando únicamente la tolerancia τ, en el primero se realiza un mallado muy basto, con una malla poco precisa, en el segundo se realiza un mallado muy preciso, con un coste computacional muy alto, y en el tercero se realiza un mallado más eficiente que los dos primeros.

1) Mallado basto

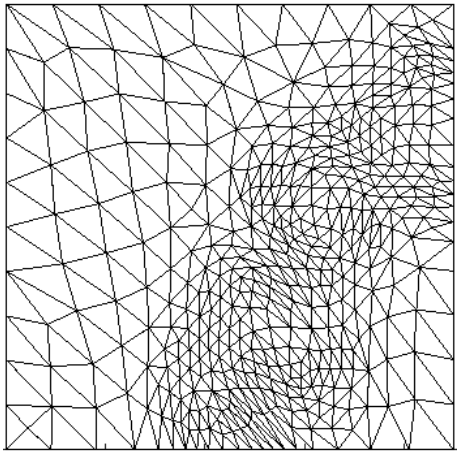


Figura 6. 2 Ejemplo de mallado con poca precisión

Se observa un mallado con un valor de tolerancia τ muy alto, igual a 1, donde prácticamente no se distinguen los bordes. El tiempo de cálculo ha sido bajo, pero este mallado no es útil.

2) Mallado muy preciso

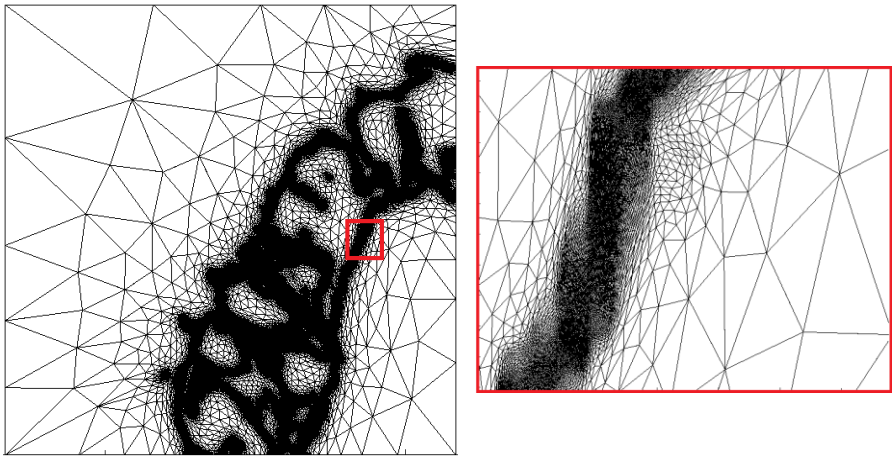


Figura 6. 3 Ejemplo de mallado con excesiva precisión y detalle del borde

Este mallado, con un valor de tolerancia τ muy bajo, igual a 0.05, representa con mucha precisión los contornos, como se observa en el detalle. No obstante, requiere mucho tiempo computacional, por lo que no resulta eficiente.

3) Mallado eficiente

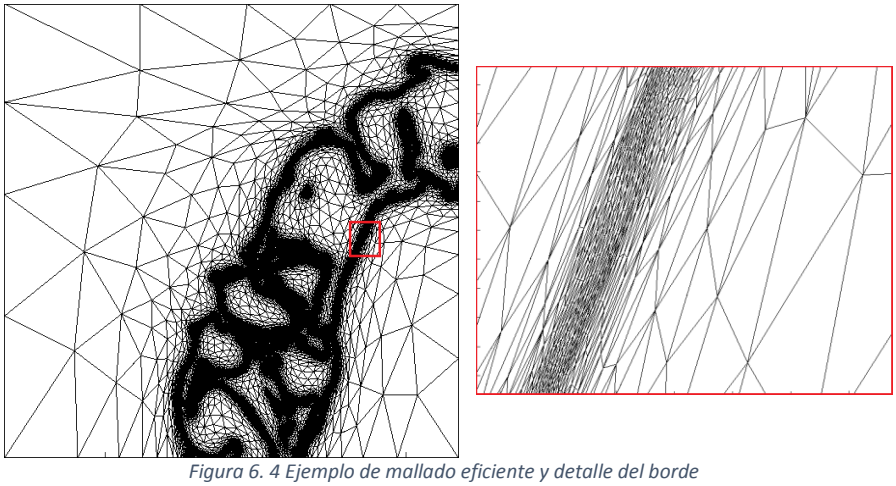


Figura 6. 4 Ejemplo de mallado eficiente y detalle del borde

Este mallado representa con suficiente precisión el borde sin necesitar mucho tiempo computacional, por lo tanto, resulta más eficiente que los dos anteriores.

En el segundo ejemplo se va a realizar lo mismo que en el ejemplo anterior, pero variando la tolerancia *Tol*, de forma que se observe las diferentes mallas según el número de iteraciones. Se mostrará un caso donde se elija un valor muy alto de tolerancia *Tol*, otro con un valor intermedio, y un último con un valor muy bajo.

1) Valor de *Tol* alto

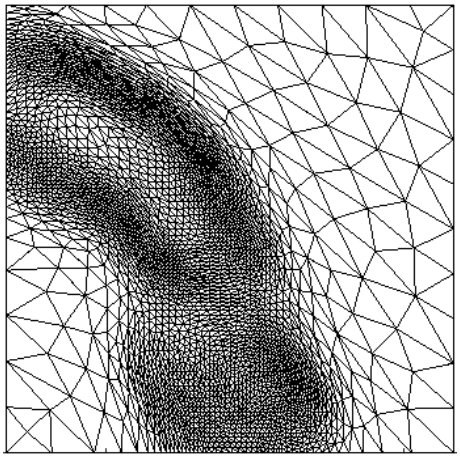


Figura 6. 5 Tolerancia Tol alta

Se observa un mallado con un valor de tolerancia *Tol* muy alto, igual a 5, donde hay un número bajo de iteraciones. En la Figura 6.5 se distinguen ligeramente los bordes, pero no lo suficiente. El tiempo de cálculo ha sido bajo debido a que se han realizado pocas iteraciones, aun así, este mallado no es útil.

2) Valor de *Tol* medio

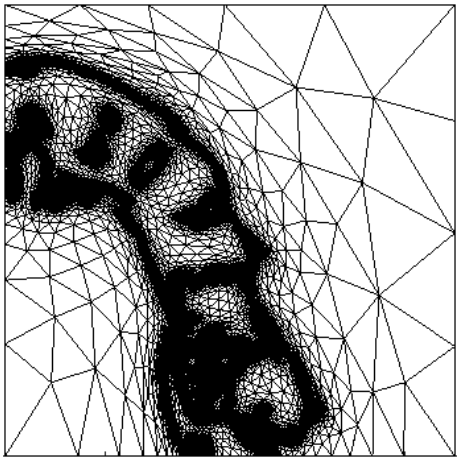


Figura 6. 6 Tolerancia Tol media

Como se ve en la Figura 6. 7, el mallado con valor de tolerancia *Tol* medio, igual a 0.8, lo que implica un número aceptable de iteraciones. En este caso representa correctamente los bordes, sin emplear un coste computacional elevado.

3) Valor de *Tol* bajo

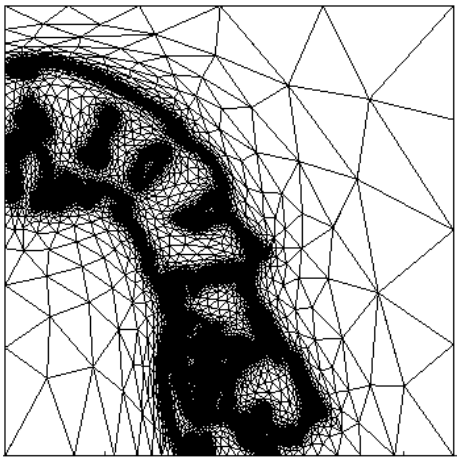


Figura 6. 7 Valor de tolerancia Tol bajo

Este mallado, con un valor de tolerancia *Tol* muy bajo, igual a 0.05, representa con mucha precisión los contornos, aun así, no muestra mucha diferencia respecto al caso con un valor de tolerancia *Tol* medio, pese a suponer mucho más coste computacional, debido al alto número de iteraciones, por lo que no resulta eficiente.

En vista de los resultados de los ejemplos mostrados se obtienen dos conclusiones, la primera es que la precisión de la malla viene principalmente gobernada por la tolerancia τ, pese a que un buen valor de la tolerancia *Tol* también es importante. La segunda conclusión es que el rango de tolerancias óptimo está comprendido en los valores mostrados en la Tabla 6. 1:

Tol	$0.2 - 1$
τ	$0.1 - 0.2$

Tabla 6. 1 Rango de valores óptimo de tolerancia

6.2. Ejemplos de mallado

En este apartado se van a mostrar dos ejemplos de mallado de dos cortes de la mandíbula, figuras Figura 6. 8 y Figura 6. 9.

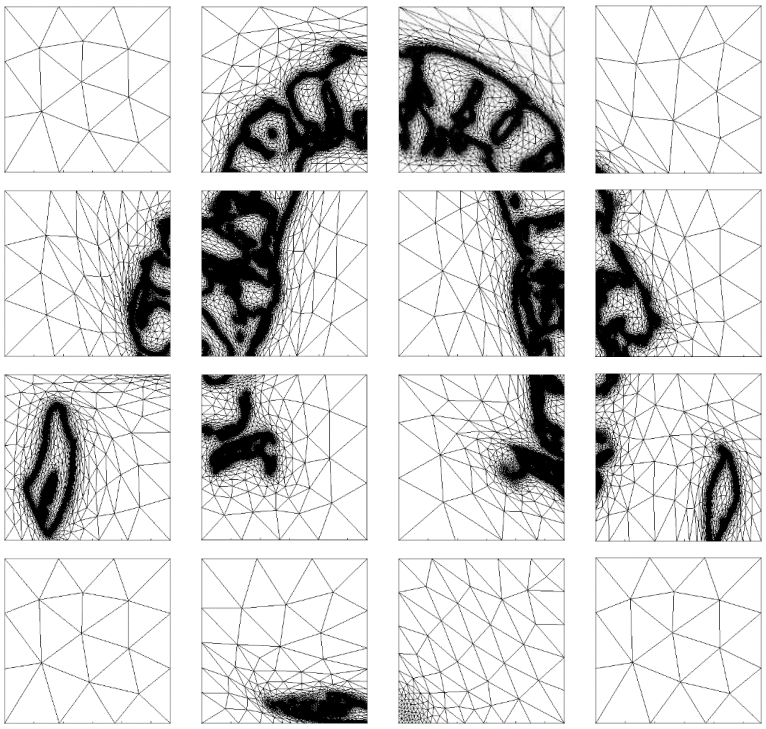


Figura 6. 8 Malla correspondiente al corte 200 de la mandíbula

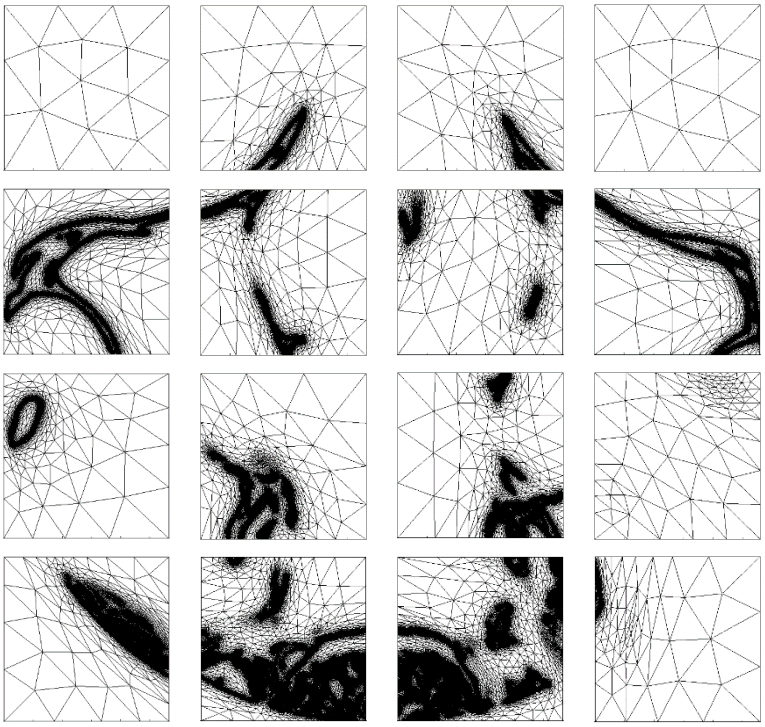


Figura 6. 9 Malla correspondiente al corte 100 de la mandíbula

7. Postproceso

Durante el postproceso se genera la matriz lógica a partir de las mallas obtenidas en la segmentación. En el ejemplo mostrado en la Figura 7. 1 se puede ver la imagen mallada con elementos triangulares, y en la Figura 7. 2 se ha construido la matriz lógica a partir de la malla. Para ello se han deseleccionado los píxeles que se encuentran dentro de los elementos que no son válidos, posteriormente se eliminan los dominios de píxeles innecesarios y se crea una capa de píxeles de seguridad.

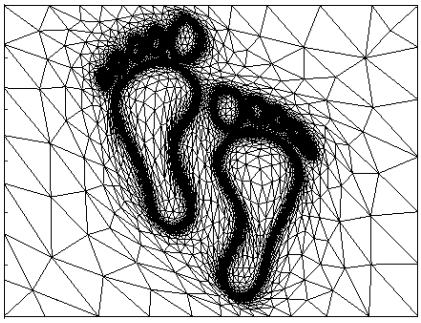


Figura 7. 1 Imagen mallada

Figura 7. 2 Matriz lógica a partir de la imagen mallada

7.1. Selección de elementos no válidos

La idea es seleccionar aquellos elementos de la malla que no son válidos y quitar los píxeles contenidos en estos elementos de la matriz lógica. Se seleccionan los elementos que no son válidos porque, pese a tener mayor superficie que los elementos que sí que interesan, el número de elementos es menor y, con ello, se reduce el coste computacional.

Los elementos se seleccionan de manera expansiva, es decir, partiendo de un elemento, se seleccionan aquellos que lo envuelven, que son aquellos que al menos comparten un nodo, tal y como muestra el ejemplo de la Figura 7. 3, y cumplen una serie de criterios que se explican más adelante. Posteriormente, para cada elemento seleccionado se repite el mismo proceso que con el elemento previo, seleccionando los elementos que lo envuelven, como se muestra en la Figura 7. 4, y así sucesivamente con los nuevos elementos seleccionados.

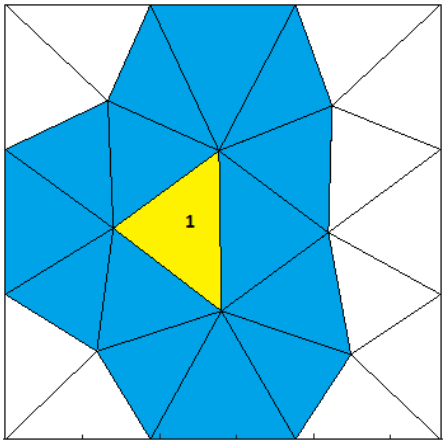


Figura 7. 3 Elementos alrededor del elemento 1

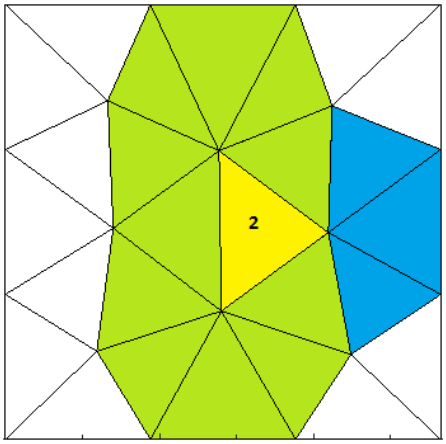


Figura 7. 4 Elementos alrededor del elemento 2

Se denomina una región de elementos al conjunto de elementos que está confinado entre bordes, ya sea bordes de fin de malla o elementos que al estar lo suficientemente estirados delimitan un borde. Dentro de la malla pueden existir diferentes regiones de elementos inconexas entre sí, como se muestra en la Figura 7. 5, donde se aprecian dos regiones. En cada región se selecciona el elemento de mayor área y se realiza el proceso explicado anteriormente hasta seleccionar todos los elementos de la región.

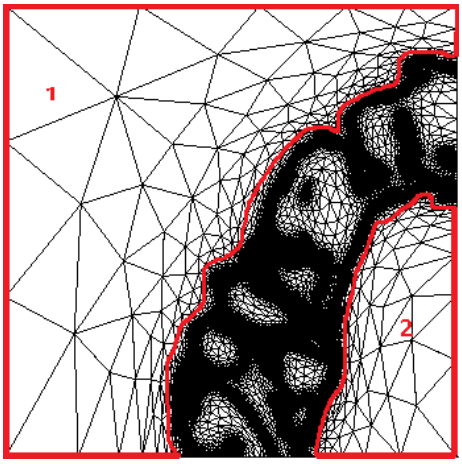


Figura 7. 5 Diferentes regiones de elementos

La selección de los elementos se realiza en base a tres criterios:

1) *Tamaño*: los elementos se seleccionan en base a un porcentaje del tamaño del elemento con mayor área de la región. En el ejemplo mostrado a continuación, se pretende demostrar la importancia de elegir un tamaño de selección de elementos adecuados. En la Figura 7. 6 se muestra un corte de la mandíbula y en la Figura 7. 7 la malla de este corte. El tamaño del elemento no debe ser ni muy pequeño, ya que es posible que se pierda información relevante de la matriz lógica, como se puede ver en los detalles rojos de la Figura 7. 8, ni muy grande, ya que la matriz lógica contendrá información irrelevante, como se puede ver en la elipsis roja de la Figura 7. 9.

Figura 7. 6 Corte de la mandíbula

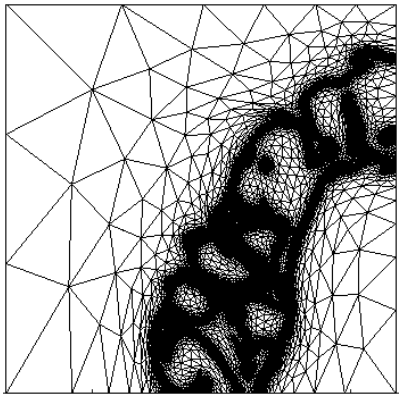


Figura 7. 7 Malla del corte

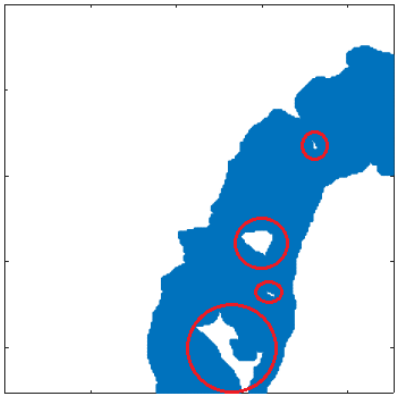


Figura 7. 8 Matriz lógica con pérdida de información relevante

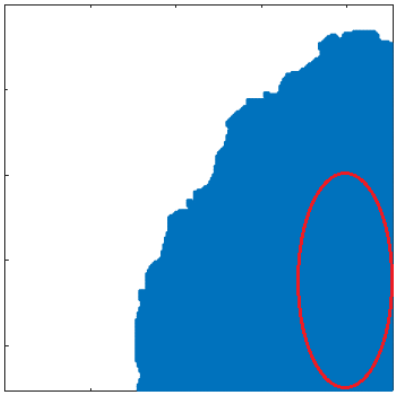


Figura 7. 9 Matriz lógica con información irrelevante

2) *Nivel de gris*: El nivel de gris define el valor, en la escala de grises, por debajo del cual el elemento se descarta. El nivel de gris se obtiene del píxel que contiene el baricentro del elemento, como se observa en la Figura 7. 10.

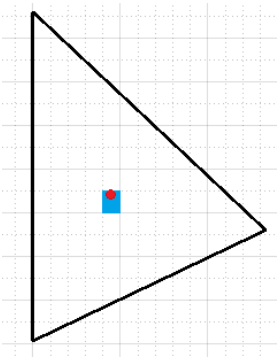


Figura 7. 10 Píxel que contiene el baricentro del triángulo

3) *Estiramiento*: El estiramiento de cada elemento triangular, definido por el valor del mínimo ángulo, indica la diferencia del elemento con respecto a un triángulo equilátero. El proceso de mallado sitúa elementos más estirados sobre los bordes y los alinea con dichos bordes. Por tanto, el nivel de estiramiento, a través de un valor umbral de ángulo mínimo del triángulo, servirá para identificar triángulos que se considera que definen un borde en la imagen.

Cada elemento tiene contenido en él un número de píxeles, según si el elemento se ha seleccionado o no, estos píxeles formarán o no parte de la matriz lógica. Los elementos tienen una geometría triangular, de la cual se conoce la posición de los nodos. Debido a que la numeración de los nodos del triángulo es antihorario, se puede determinar que un píxel estará contenido dentro del elemento si todos los productos vectoriales entre cada lado del triángulo y el píxel son positivos. En caso de que haya alguno negativo, el píxel no estará contenido dentro del elemento.

En el siguiente ejemplo explicativo se tiene un elemento triangular, referido con sus nodos 1, 2 y 3, dentro de una cuadrícula de píxeles, de los cuales se pretende averiguar cuáles están contenidos dentro del elemento. En las figuras Figura 7. 11. Figura 7. 12 y Figura 7. 13 se puede observar el resultado del producto vectorial de cada lado (marcado con línea gruesa) con todos los píxeles. Se marca en azul aquellos píxeles cuyo producto vectorial es mayor o igual a 0 y en rojo los que son negativos.

En la Figura 7. 14 se ha combinado los resultados de los tres productos vectoriales y se observa los píxeles que caen dentro del triángulo en azul y aquellos que caen fuera en rojo.

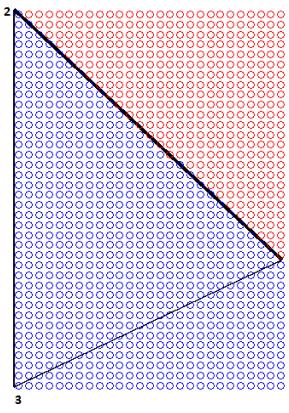


Figura 7. 11 Producto vectorial respecto 1-2

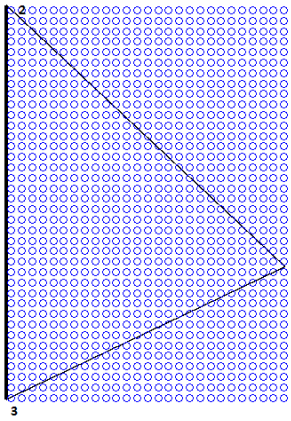


Figura 7. 12 Producto vectorial respecto 2-3

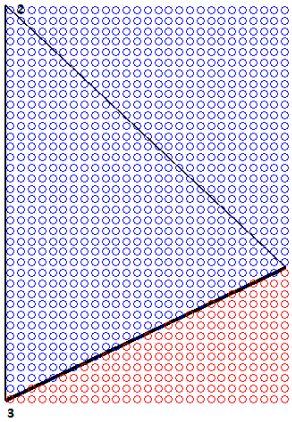


Figura 7. 13 Producto vectorial respecto 3-1

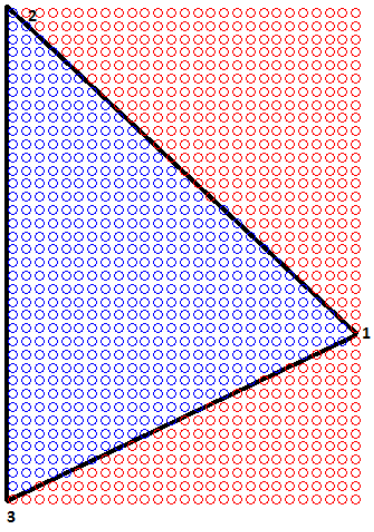


Figura 7. 14 Píxeles dentro del triángulo

7.2. Combinación de matrices lógicas

El programa ofrece la posibilidad de generar la matriz lógica considerando los tres planos de corte: XY, YZ y XZ, pudiendo seleccionar la información de uno, dos o tres planos de corte para crear la matriz lógica. Según el número de planos de corte que se haya realizado en el proceso, se crean diferentes casos para obtener la matriz lógica:

- 1 plano de corte, la matriz lógica se crea como la matriz lógica de este plano de corte.
- 2 planos de corte, la matriz lógica se crea como la combinación de estos dos planos de corte. Para crear la matriz lógica se considerarán los píxeles que al menos están contenidos en una de las dos matrices de cada plano.
- 3 planos de corte, la matriz lógica se crea como la combinación de estos tres planos de corte. Para crear la matriz lógica se considerarán los píxeles que al menos están contenidos en dos de las tres matrices de cada plano.

En la Figura 7. 15 se puede ver las matrices lógicas de un corte de la mandíbula obtenidas mediante los tres planos de corte.

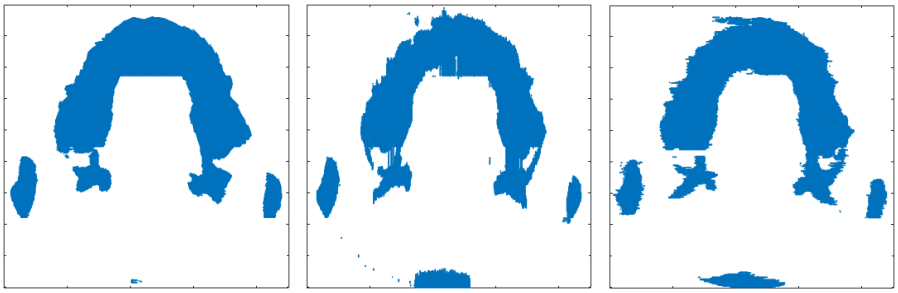


Figura 7. 15 Matrices lógicas de los planos XY, YZ y XZ, respectivamente

Y a continuación se puede ver la combinación de las tres matrices en la Figura 7. 16.

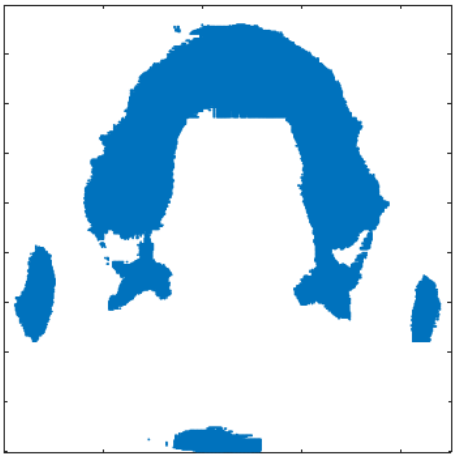


Figura 7. 16 Combinación de las matrices lógicas de cada plano

7.3. Eliminación de dominios de elementos inconexos

En las imágenes médicas, existen partes del cuerpo que tienen el mismo nivel de gris, como por ejemplo en algunas ocasiones tejido muscular y hueso trabecular, por lo que en la matriz lógica pueden aparecer dominios de píxeles inconexos.

En el programa desarrollado en el DIMM no se permite trabajar con más de un dominio, por ello, se eliminan estos dominios inconexos, de forma que solo quede aquel dominio que sea el más grande, que se supone el hueso que interesa de la imagen.

En la Figura 7. 17 se observa un dominio de píxeles inconexo al resto de la matriz, marcado en una elipse roja, que posteriormente se elimina. Es cierto que también existen otros dos dominios inconexos, marcados con una elipse verde, no obstante, la imagen es un corte de una matriz tridimensional, y estos dos dominios se conectan al dominio más grande en la matriz tridimensional.

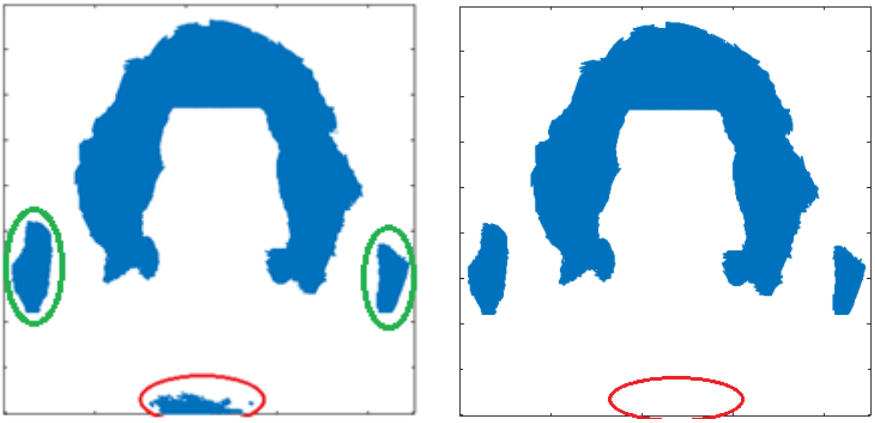


Figura 7. 17 Matriz lógica con partes inconexas y habiendo quitado las partes inconexas

7.4. Capa de seguridad de píxeles

Por seguridad, se crea una capa de píxeles alrededor de la matriz lógica en las tres dimensiones, X, Y, Z, y en las diagonales XY, YZ, XZ y XYZ. Esta capa sirve tanto como margen de seguridad, como para rellenar posibles huecos que hayan podido aparecer dentro de la matriz, como se puede comprobar en la Figura 7. 18.

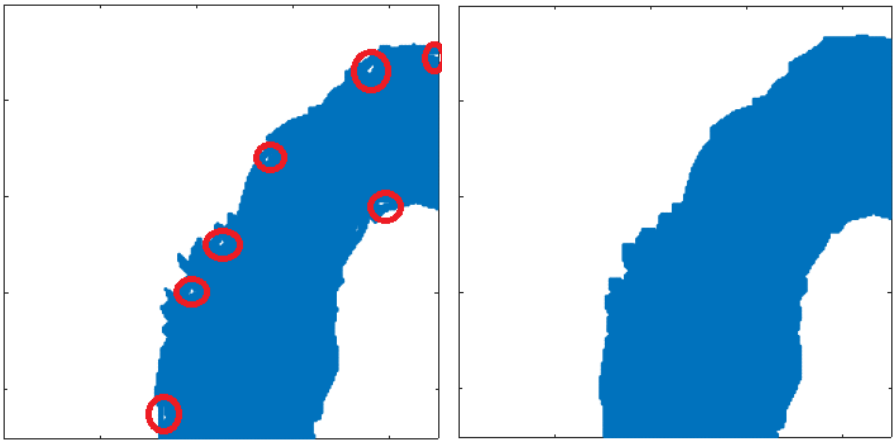


Figura 7. 18 Matriz lógica con agujeros y matriz lógica después de crear la capa de seguridad

Dentro de Figura 7. 19, en la imagen de la izquierda se puede ver la matriz lógica antes de crear la capa de seguridad, y en la imagen de la derecha se puede ver de color azul la capa de seguridad.

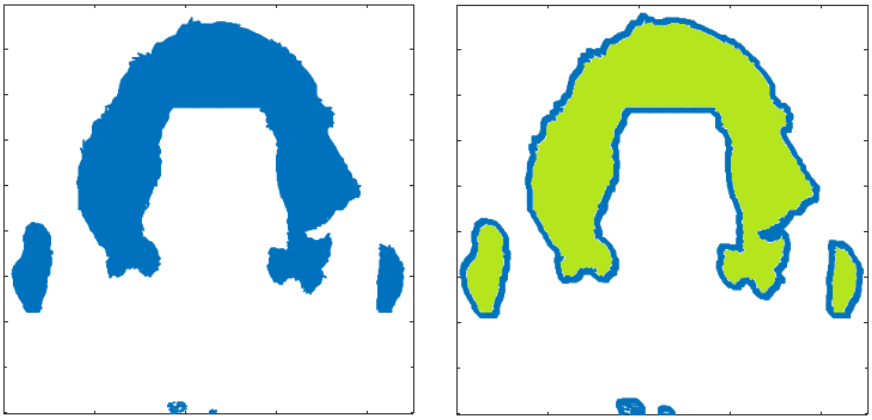


Figura 7. 19 Capa de seguridad de píxeles

8. Comparación de resultados entre filtrado de imágenes y tolerancias

La segmentación está muy relacionada con el filtrado de imágenes que se realiza en el preproceso, explicado en el apartado Cambio de escala de color. En una imagen donde se acentúan bien los bordes, no es necesaria una tolerancia τ muy baja, implicando una mayor precisión. En este aparatado se pretende comparar los resultados de realizar un buen filtrado de imagen frente a seleccionar una tolerancia de segmentación τ más baja. Esto se ilustrará con dos ejemplos.

En este primer ejemplo se pretende mostrar la diferencia de mallado entre una imagen a la que no se le ha realizado el filtrado de imagen, Figura 8. 1, y otra a la que sí, Figura 8. 3.

Figura 8. 1 Imagen sin filtrado

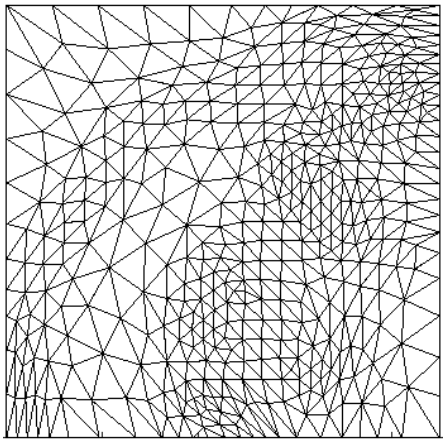


Figura 8. 2 Malla de la imagen sin filtrado

Figura 8. 3 Imagen filtrada

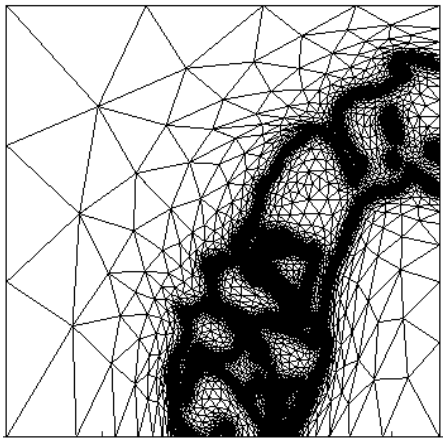


Figura 8. 4 Malla de la imagen filtrada

Como se observa en el primer caso, existe mucho ruido en la imagen y los bordes no aparecen tan acentuados como en el segundo caso. Por ello el mallado, Figura 8. 2, es más basto, este caso requeriría una tolerancia τ más baja para buscar más precisión en la malla, implicando mayor coste computacional. En el segundo caso se observan bordes más acentuados y una precisión de malla, Figura 8. 4, mayor respecto al primer caso.

En el segundo ejemplo se pretende hacer una comparativa de tiempos entre realizar un buen filtrado de la imagen y emplear una tolerancia τ más baja al representar un borde difícil como el que se muestra en el detalle rojo de la Figura 8. 5. Se supondrán dos casos, en el primero se mantendrá fijada la tolerancia τ y se harán pruebas variando el rango de valores Hounsfield del filtrado, en el segundo se mantendrá fijado el rango de valores Hounsfield y se harán pruebas variando la tolerancia τ.

Por defecto, los valores fijados del rango de Hounsfield del filtrado estarán comprendidos entre el valor mínimo, 50, y el valor máximo, 350, y la tolerancia τ será 0.15.

En la Tabla 8. 1 se muestra el valor de tolerancia τ y los valores del rango de Hounsfield con los que se ha hecho la prueba, el tiempo que ha tomado para la imagen entera, la malla generada y la imagen filtrada del detalle marcado en rojo de la Figura 8. 5.

8. Comparación de resultados entre filtrado de imágenes y tolerancias

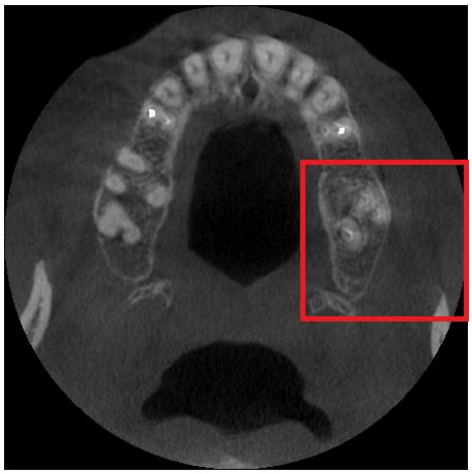
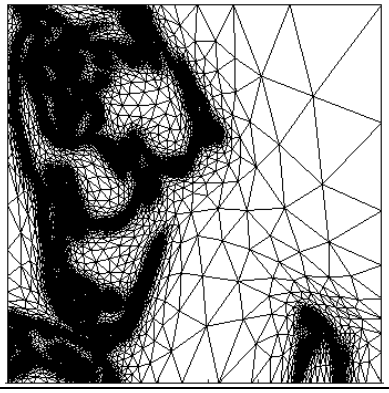
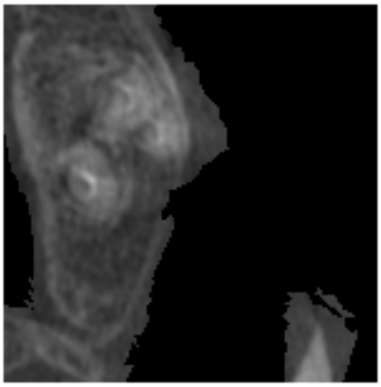
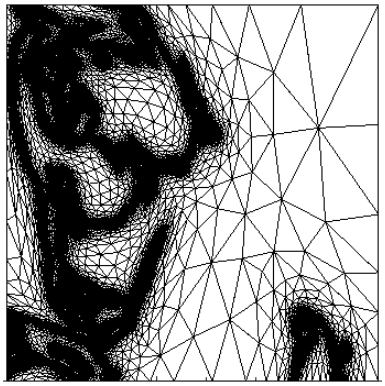
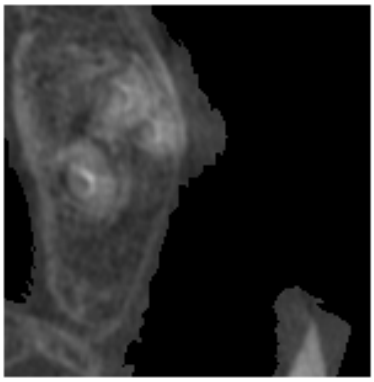
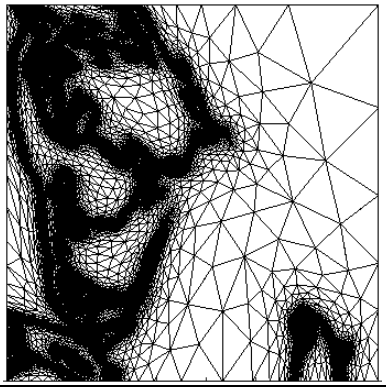
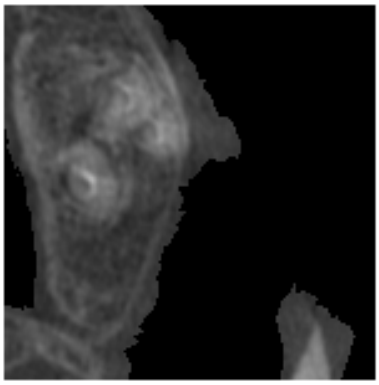
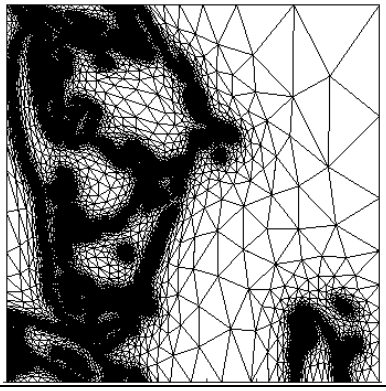
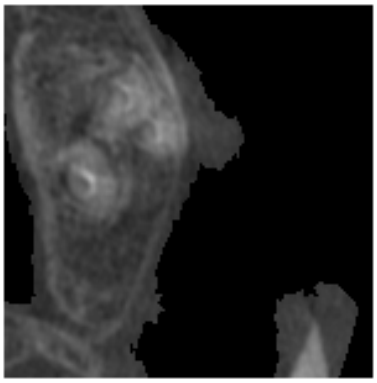
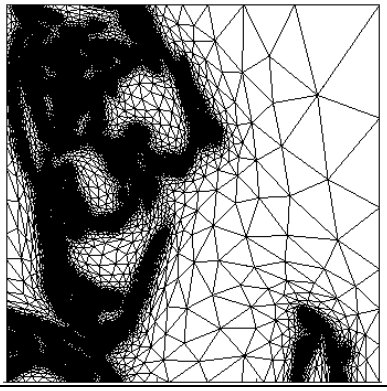
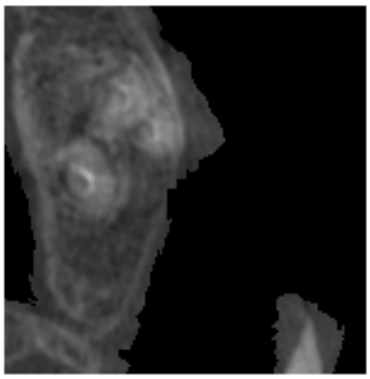
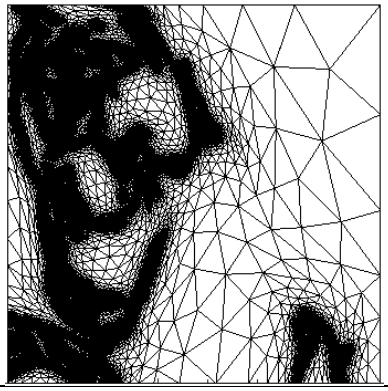
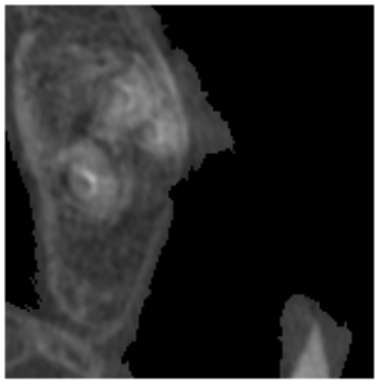
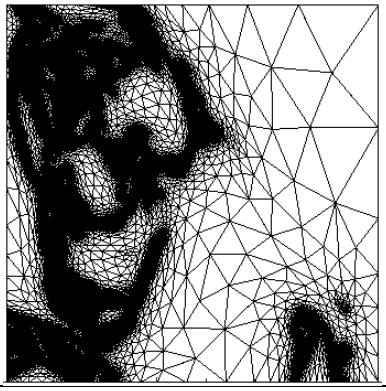
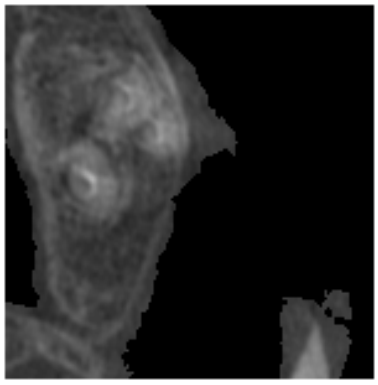


Figura 8. 5 Corte de la mandíbula con detalle

	Valor de tolerancia fijada	
$\tau = 0.15$ Hounsfield=50-250 Tiempo=41 s		

$\tau = 0.15$ Hounsfield=50-200 Tiempo=49 s		
$\tau = 0.15$ Hounsfield=50-150 Tiempo=60 s		
$\tau = 0.15$ Hounsfield=50-100 Tiempo=78 s		

Rango de valores Hounsfield fijado

$\tau = 0.1$ Hounsfield=50-350 Tiempo=90 s		
$\tau = 0.09$ Hounsfield=50-350 Tiempo=126 s		
$\tau = 0.08$ Hounsfield=50-350 Tiempo=169 s		

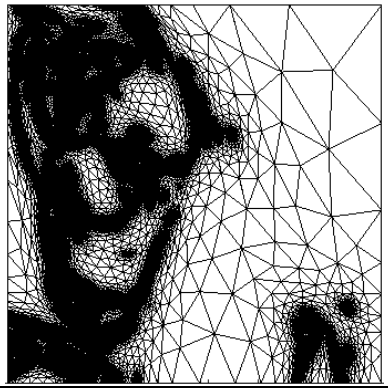
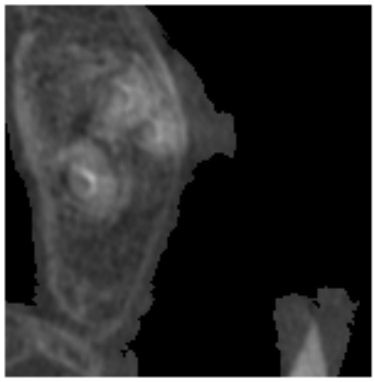
$\tau = 0.07$ Hounsfield=50-350 Tiempo=245 s		

Tabla 8. 1 Resultados de la comparativa

Rango de Hounsfield	Tiempo (s)
50-250	41
50-200	49
50-150	60
50-100	78

τ	Tiempo (s)
0.1	90
0.09	126
0.08	169
0.07	245

Como se puede observar los resultados de tiempo en la Tabla 8. 1,para llegar a una malla muy lograda el filtrado de imágenes resulta más eficiente, ya que tarda menos tiempo que emplear una tolerancia τ más baja, del orden de tres veces menos.

No obstante, aplicar un rango de filtrado muy bajo puede generar ruido en la imagen, como se puede ver en las elipses rojas mostradas en la Figura 8. 6, lo que genera una matriz lógica con información irrelevante, como se observa en la Figura 8. 7.

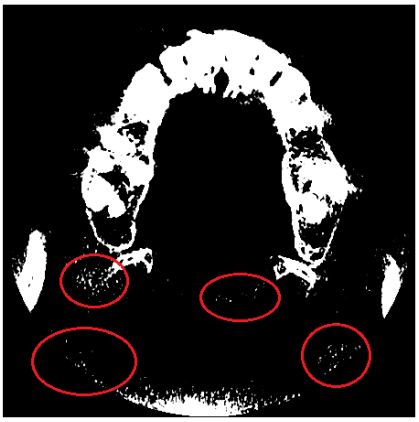


Figura 8. 6 Imagen filtrada con un rango de Hounsfield muy bajo, 50-100

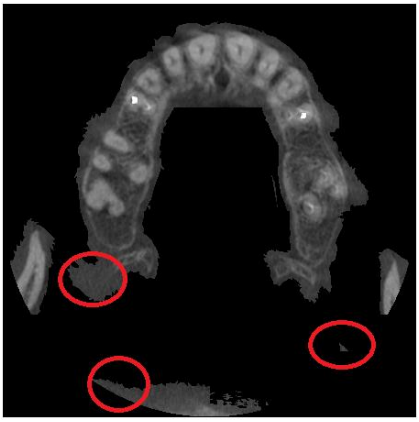


Figura 8. 7 Resultado de la imagen filtrada

9. Validación del trabajo

En este apartado se pretende demostrar la validez del presente trabajo comparando las imágenes médicas con el resultado de filtrarlas con la matriz lógica.

Como se ve en los ejemplos, la matriz lógica ha conseguido descartar aquellas partes de la imagen que no son útiles, dejando solo las que interesan.

9.1. Mandíbula

En las siguientes imágenes se muestran dos cortes de la mandíbula, antes y después de ser filtrados con la matriz lógica. En las figuras Figura 9. 1 y Figura 9. 2 se ha realizado el filtrado del corte 200 mientras que en las figuras Figura 9. 3 y Figura 9. 4 se ha realizado el filtrado del corte 300. Tras el filtrado se consigue eliminar el 73.7% de los píxeles.

Figura 9. 1 Corte 200 de la mandíbula

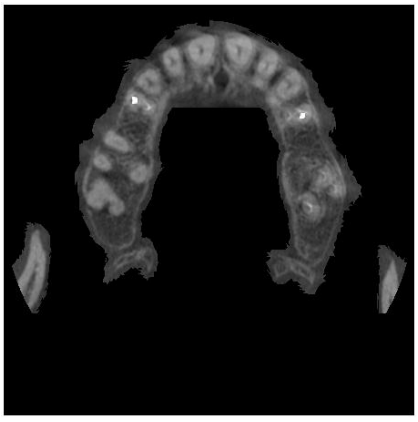


Figura 9. 2 Corte 200 filtrado

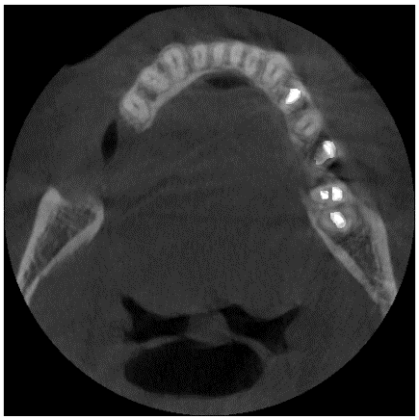


Figura 9. 3 Corte 300 de la mandíbula

Figura 9. 4Corte 300 filtrado

En la Figura 9. 5 se puede ver una vista 3D de la mandíbula que se obtiene con la matriz lógica. En la imagen se ha quitado la capa de seguridad que crea la matriz lógica, para poder visualizar mejor la mandíbula.

Figura 9. 5 Vista 3D de la mandíbula tras ser filtrada

9.1.1. Simulación con CgFEM

Para la validación del trabajo se ha realizado una prueba donde se ha simulado la mandíbula filtrada mostrada anteriormente con el procedimento CgFEM desarrollado en el Departamento de Ingeniería Mecánica y Materiales de la Universidad Politécnica de Valencia. Más que los resultados que se van a mostrar, en este apartado se pretende demostrar que el trabajo es adaptable al procedimiento CgFEM.

En la Figura 9. 6 se puede ver la malla cartesiana obtenida en la simulación a partir de la mandíbula filtrada generada en el presente trabajo.

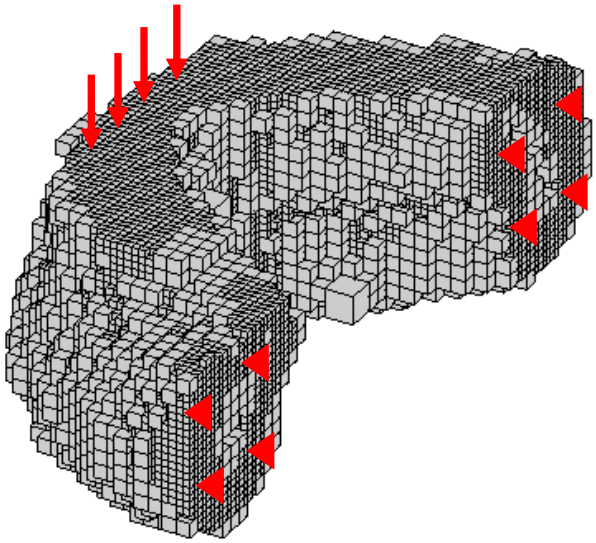


Figura 9. 6 Malla cartesiana

Para la simulación se han empotrado los nodos correspondientes a los dientes traseros de la mandíbula y se ha aplicado una presión vertical de -1MPa en los nodos de los dientes delanteros, como se ve en la Figura 9. 6. A continuación, se muestra algunas de las tensiones obtenidas en la simulación.

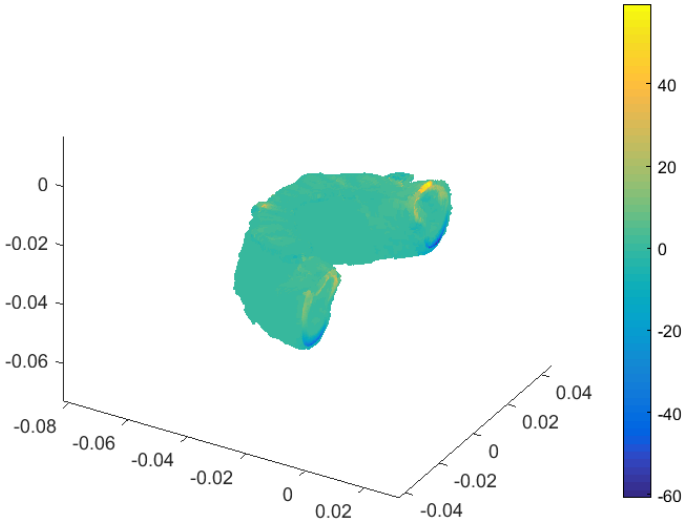


Figura 9. 7 Tensión en el eje X

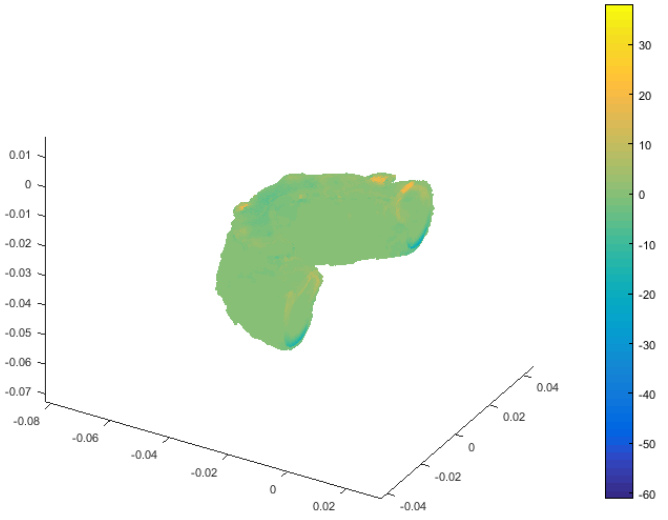


Figura 9. 8 Tensión en el eje Y

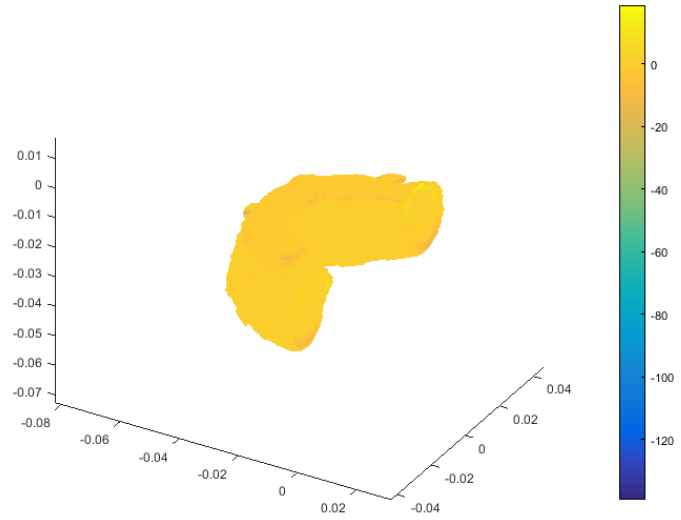


Figura 9. 9 Tensión en el eje Z

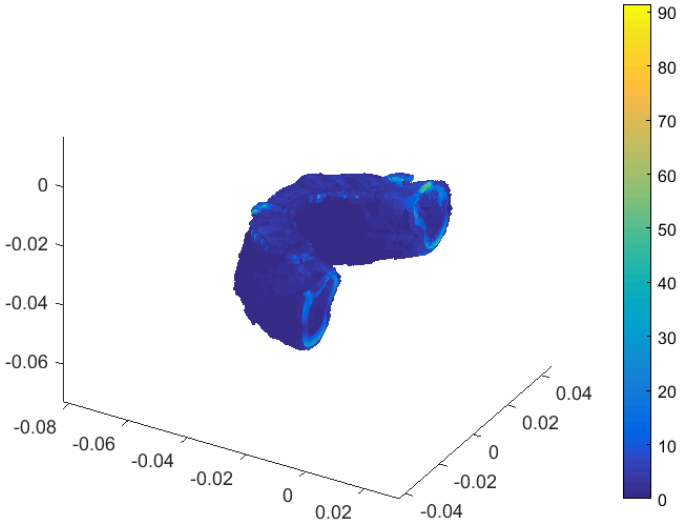


Figura 9. 10 Tensión de Von Mises

9.2. Fémur

En la Figura 9. 11 se puede ver un corte del fémur y en la Figura 9. 12 se puede ver el corte del fémur después de filtrarlo. En la Figura 9. 13 se puede ver una vista 3D del fémur. Tras el filtrado se consigue eliminar el 92% de los píxeles.

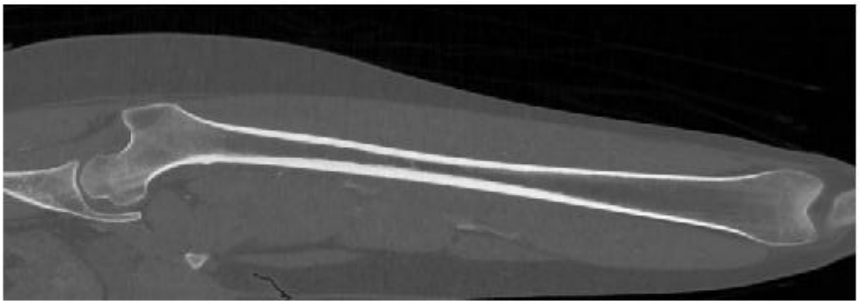


Figura 9. 11 Corte del femur

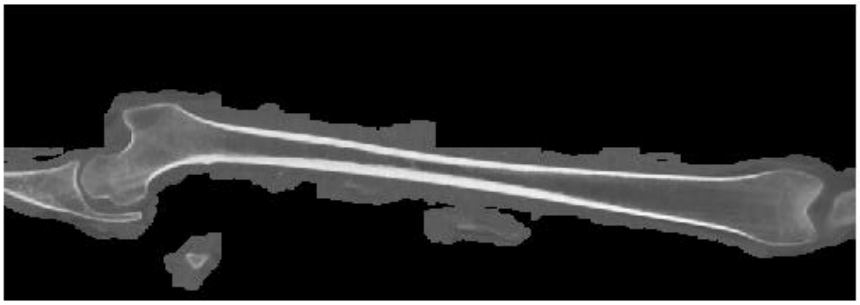


Figura 9. 12 Corte del fémur filtrado

Figura 9. 13 Vista 3D del fémur

9.3. Pulmón

El trabajo también permite la selección de diferentes partes del cuerpo que no sean hueso. En las siguientes imágenes se puede un corte del pulmón, figuras Figura 9. 14 y Figura 9. 15, y una vista 3D del pulmón filtrado con la matriz lógica, Figura 9. 16. Tras el filtrado se consigue eliminar el 78% de los píxeles.

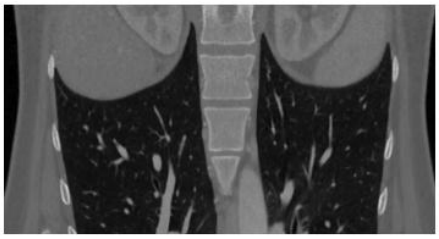


Figura 9. 14 Corte del pulmón

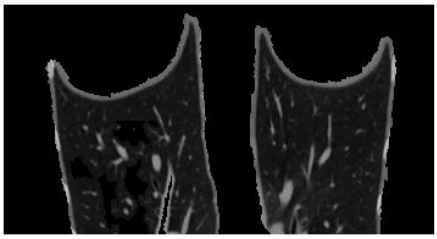


Figura 9. 15 Corte del pulmón filtrado

Figura 9. 16 Vista 3D del pulmón

10. Conclusiones

Tras la finalización del trabajo se concluye que consigue su objetivo ya que devuelve la matriz lógica correspondiente al hueso. Pese al inconveniente del tiempo de ejecución, cabe destacar que es un programa automatizado que solo requiere la intervención del usuario al principio.

Una de las primeras consideraciones era la de trabajar con un proceso paralelizado o con uno secuencial. A la vista de los resultados de tiempo obtenidos, el procesamiento paralelizado es más eficiente que el procesamiento secuencial. Pese a que el tiempo de cálculo en ejecutar un proceso sea menor, el procesamiento paralelizado lanza múltiples procesos simultáneamente, disminuyendo el tiempo total.

Viendo los resultados, se recomienda trabajar únicamente con un plano de corte, y que este plano sea el XY, por los siguientes motivos:

- Trabajar con más de un plano de corte implica mayor tiempo de ejecución del programa, resultando en poca mejora de los resultados, como se observa en el apartado Combinación de matrices lógicas.
- El plano XY es el que mayor resolución tiene, entre los tres planos de corte.

El 90% del tiempo de ejecución se consume en la segmentación. Resulta muy eficiente realizar un buen preproceso, seleccionando un número óptimo de divisiones y realizando un buen filtrado de imágenes para marcar un buen contraste en los bordes, ayudando de esta manera a reducir el tiempo de segmentación.

11. Futuros trabajos

Este trabajo es un primer intento de realizar un proceso automatizado que detecte el hueso en imágenes médicas.

Como futura línea de investigación se propone reducir el tiempo computacional del trabajo. Para ello se sugiere realizar el proceso con un mallado tridimensional [4], en lugar del mallado bidimensional por capas en el que se basa este trabajo. [1]

Otro futuro trabajo sería detectar distintas partes del cuerpo. Aunque el trabajo lo permite, está enfocado principalmente a detectar el hueso, por lo que detectar otras partes del cuerpo no está totalmente desarrollado.

12. Referencias

(1) https://en.wikipedia.org/wiki/Hounsfield_scale
(2) https://en.wikipedia.org/wiki/Netpbm_format
(3) http://www.ctlab.geo.utexas.edu/about-ct/resolution-and-size-limitations/
(4) http://slideplayer.com/slide/7574693/
(5) http://www.freefem.org/
(6) https://en.wikipedia.org/wiki/DICOM

13. Bibliografía

[1] Micheletti, S., Perotto, S.: Anisotropic Adaptation via a Zienkiewicz–Zhu Error Estimator for 2D Elliptic Problems (2010)
[2] Quarteroni, A.: Modellistica Numerica Per Problemi Differenziali
[3] George, P.L., Borouchaki, H.: Delaunay Triangulation and Meshing— Application to Finite Element (1998)
[4] Micheletti, S., Perotto, S.: An anisotropic Zienkiewicz–Zhu-type error estimator for 3D applications (2010)
[5] Navarro Jiménez, J.M., Ródenas García, J.J.: Obtención de modelos de elementos finitos con mallados cartesianos independientes de la geometría a partir de imágenes médicas y su uso en la simulación de interacción entre prótesis y tejidos vivos (2013)

ANEXOS

A. Manual del usuario

A.1. Instalación de la aplicación

En este trabajo se emplean los softwares Matlab, en su versión 16, y FreeFem++, en su versión 3.51.

FreeFem++ se puede descargar gratuitamente de la siguiente página:

http://www.freefem.org/ff++/download.php

Como se explica en el manual del programador, es necesario que el usuario cree tantas instancias de FreeFem++ como workers tiene su versión de Matlab. Esto es tan sencillo como copiar el archivo *FreeFem++.exe* y renombrarlo como *FreeFem++1.exe, FreeFem++2.exe, FreeFem++3.exe,* etc.

A.2. Acceso a la aplicación

La aplicación se encuentra dentro de la función *LogicalMatrix* de Matlab. Esta función junto al resto de funciones de Matlab y archivos EDP de FreeFem++ deben estar contenidas en el mismo directorio.

A.3. Imágenes

La aplicación pregunta por el directorio donde se encuentran las imágenes. Es importante que los únicos archivos de este directorio sean las imágenes y estén todos en el mismo formato.

A.4. Ventana de adquisición de datos

Una vez llamada la función, aparece una ventana, como se ve en la Figura A. 1, donde el usuario debe introducir los datos con los que va a trabajar la aplicación.

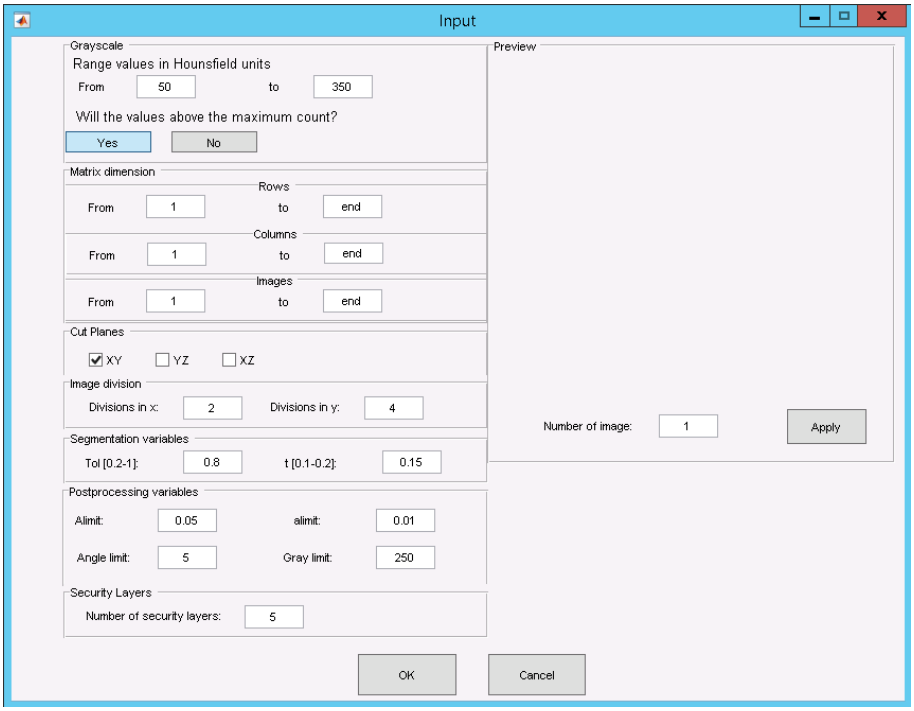


Figura A. 1 Ventana de adquisición de datos

- **Grayscale**: Primeramente, pide el rango de valores en la escala Hounsfield en los que se desea que trabaje la aplicación. Con esto se consigue acentuar el borde de las partes del cuerpo que se deseen, reduciendo el tiempo de segmentación. Los valores se obtienen de la Tabla A. 1 o la Figura A. 2:

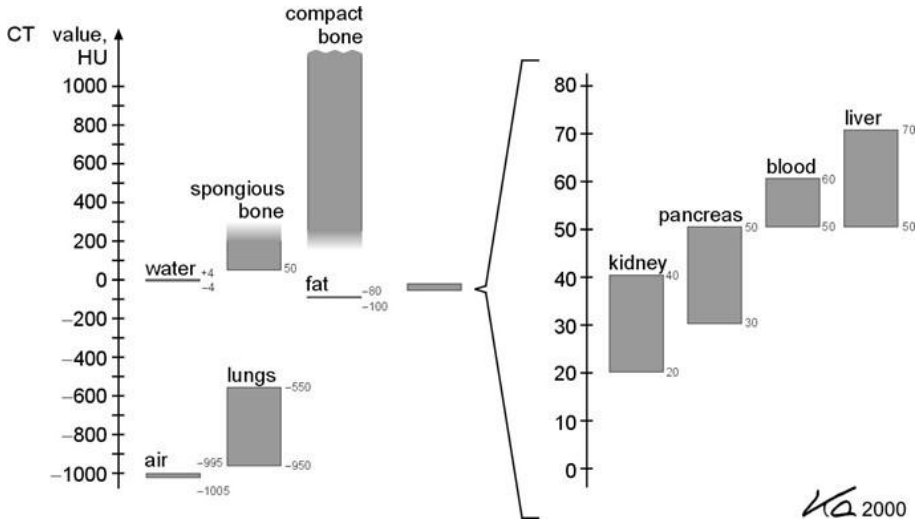


Figura A. 2 Valores de la escala Hounsfield (4)

Tabla A. 2 Escala Hounsfield (1)

Substance	HU
Air	−1000
Lung	−500
Fat	−120 to −90
Water	0
Urine	-5 to +15
Bile	-5 to +15
CSF	15
Kidney	+20 to +45
Blood	+30 to +45
Muscle	+35 to +55
Grey matter	+37 to +45
White matter	+20 to +30
Liver	+40 to +60
Soft Tissue, Contrast	+100 to +300
Bone	+700 (cancellous bone) to +3000 (cortical bone)

A continuación, pregunta si los valores por encima del valor máximo cuentan, de esta manera, eligiendo sí, los valores por encima del rango tendrán un valor de 255 en la escala de grises 255, por lo que se trabaja con estos valores también. Resulta muy útil cuando se quiere trabajar con el hueso, ya que este tiene los valores más altos en la escala Hounsfield. Eligiendo no, los valores por encima del rango tendrán un valor de 0, trabajando únicamente con los valores del rango.
Por defecto, el rango de valores está comprendido entre **50** y **350** unidades Hounsfield, y los valores por encima del máximo sí que cuentan.

- **Matrix Dimensions**: se introducen las dimensiones de la matriz, que contiene la información de las imágenes, con la que trabajará la aplicación. Es posible que solo se desee la información de una parte de la matriz, por lo que el usuario debe indicar el rango de filas, columnas e imágenes con la que quiere trabajar. La aplicación está programada para que, si el usuario introduce un valor máximo de filas, columnas o imágenes superior a las dimensiones de la matriz de imágenes, la aplicación se quede con las dimensiones de la matriz, de forma que no genere error.
  Por defecto, las dimensiones son las de la matriz donde vienen recopiladas las imágenes.

- **Preview**: en esta ventana se muestra el corte de la matriz seleccionado con los valores del filtrado de imágenes y dimensiones de la matriz elegidos en *Grayscale* y *Matrix Dimensions*, respectivamente. De esta manera el usuario tiene constancia del preproceso que está realizando, y le permite seleccionar mejor las variables para ajustarlas a su caso.

Figura A. 3 Preview del corte 200 filtrado

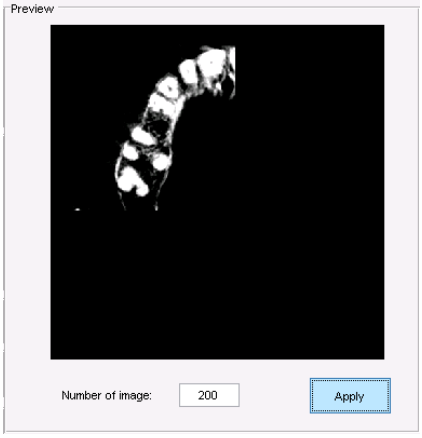


Figura A. 4 Preview del corte 200 filtrado y cortado

El recuadro *Number of image* indica el corte de la matriz, en el plano XY, que se quiera visualizar.

Con el botón *Apply* se visualiza la imagen. Es probable que la visualización tarde entre 10 y 20 segundos.

Para matrices de imágenes superiores a 2GB no se puede visualizar las imágenes, ya que tal y como está programado, Matlab no lo permite.

- **Cut Planes**: Posteriormente, la aplicación pide el plano de corte donde se va a generar la segmentación, pudiendo ser XY, YZ o XZ. También es posible hacer la segmentación en dos o tres planos de corte, resultando la matriz lógica una combinación de las matrices lógicas obtenidas en cada plano de corte.
    - En caso de elegir un plano de corte, la matriz lógica resultante será la matriz lógica de ese plano de corte.
    - En caso de elegir dos planos de corte, en la matriz lógica resultante, están los píxeles que al menos están contenidos en una de las dos matrices de cada plano.
    - En caso de elegir tres planos de corte, en la matriz lógica resultante, están los píxeles que al menos están contenidos en dos de las tres matrices de cada plano.

  Se considera el plano XY con las dimensiones de las filas y columnas de las imágenes, o las dos primeras dimensiones de la matriz de imágenes, si están

contenidas en una matriz 3D inicialmente. Se considera el plano el plano YZ con las dimensiones de las filas y el número de imágenes. Se considera el plano XZ con las dimensiones de las columnas y el número de imágenes.
Por defecto, viene establecido el **plano XY**.

- **Image Division**: Para facilitar la segmentación se pueden dividir las imágenes. La aplicación pregunta el número de divisiones en x e y de cada imagen.
  Por defecto, se realizan **2x4 divisiones**.

- **Segmentation Variables**: El apartado de variables de la segmentación es uno de los más importantes, ya que la segmentación es la tarea que más tiempo ocupa del trabajo.

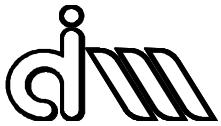
  La variable ***Tol*** está relacionada con el número de iteraciones en el mallado adaptativo. Un valor muy bajo genera muchas iteraciones, resultando en mucho tiempo de ejecución, mientras que un valor muy alto genera pocas iteraciones, resultando en una mala precisión de la malla. Por defecto, viene establecido un valor de **0.8**.

  La variable ***t*** está relacionada con la precisión en cada iteración del mallado adaptativo. Un valor muy bajo genera mucho tiempo de ejecución en cada iteración, un valor muy alto genera poca precisión en la malla. Por defecto, viene establecido un valor de **0.15**.

- **Postprocessing variables**: Las variables del postproceso ayudan a descartar aquellos elementos, de la malla obtenida en la segmentación, que no se deseen. Se selecciona un primer elemento y se van seleccionando los elementos que lo envuelven hasta llegar a un borde.

  Los elementos se seleccionan en base a su tamaño, estiramiento y nivel de gris.

  - *Alimit*: define el porcentaje de área, en tanto por 1, que deben de tener los primeros elementos que se eligen respecto al área del elemento más grande de la malla. Por defecto, viene establecido un valor de **0.05**.



- *alimit*: define el porcentaje de área, en tanto por 1, que deben de tener los elementos que envuelven al primer elemento, respecto al área del elemento más grande de la malla. Por defecto, viene establecido un valor de **0.01**.
- *Angle limit*: define el ángulo a partir del cual se considera un elemento estirado o no. Por defecto, viene establecido un valor de **5**.
- *Gray limit*: define el valor en la escala de grises 255, por debajo del cual el elemento no es de interés. Por defecto, viene establecido un valor de **250**.

- **Security Layer**: Para generar un margen de seguridad, se genera una capa de píxeles alrededor de los bordes dentro de la matriz lógica. Con esto también se consigue rellenar los pequeños huecos que se hayan podido generar dentro de la matriz. En el apartado *Security Layers* se introduce el número de capas de píxeles que se deseen.
  Por defecto, viene establecido un valor de **2 capas**.

A.5. Obtención de datos

Existen diversos directorios donde se almacena la información de interés una vez ha finalizado la aplicación.

- En la carpeta ***LogicMats*** se guarda las matrices lógicas de cada plano de corte y la matriz lógica *LogicMat* combinación de éstas.
- En la carpeta ***Meshes*** se guarda las variables *N* y *E* referentes a los nodos y elementos de las mallas. Dentro de la carpeta hay tres subcarpetas referentes a cada plano de corte con el que se haya trabajado, siendo 1 al plano XY, 2 al plano YZ y 3 al plano XZ.
- En la carpeta ***Variables*** se guarda las variables con las que ha trabajado la aplicación.

Además, la función devuelve en memoria la matriz lógica *LogicMat*.

A.6. Ejemplo práctico

En este apartado se ilustra un ejemplo para que el usuario tome como referencia al usar la aplicación. Se desea obtener la matriz lógica del fémur de la Figura A. 5.

Figura A. 5 Fémur

Se llama a la aplicación con la función *LogicalMatrix* dentro de Matlab.

Primeramente, aparece la ventana para indicar la carpeta donde se encuentran los archivos, en este caso, *Femur Dicom*.

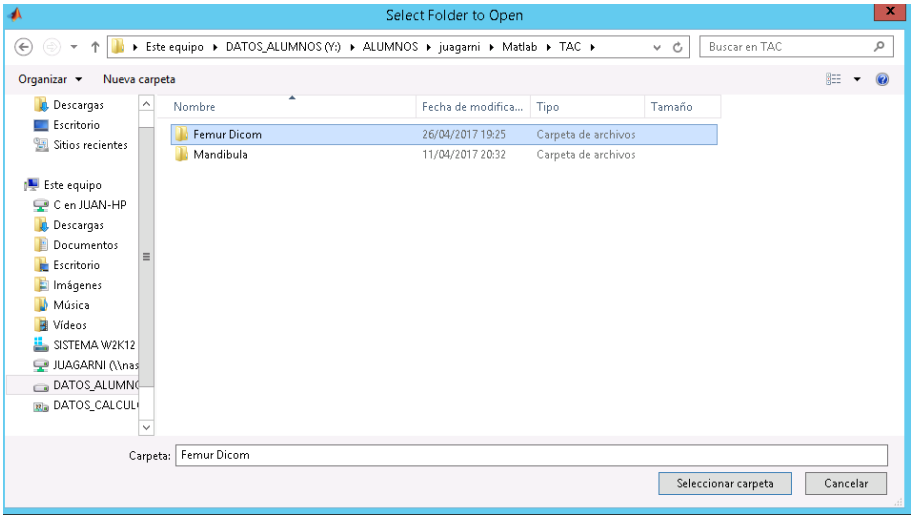


Figura A. 6 Directorio de imágenes

Aparece la ventana de adquisición de datos *Input*.

Se introduce un rango entre 50 y 650 en el apartado *Range values in Hounsfield Units*, y se escoge la opción de *Yes* a la pregunta *Will the values above the máximum count?*.

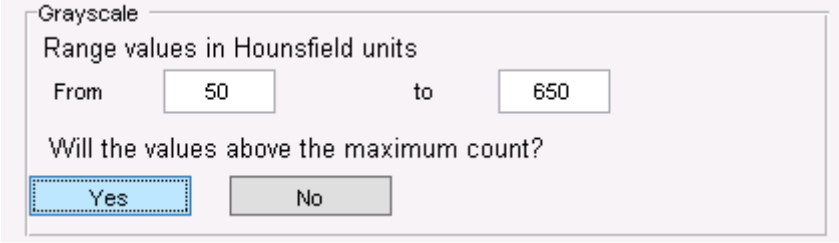


Figura A. 7 Variables del filtrado

El fémur está contenido dentro de la matriz que contiene la información de todo el cuerpo, entre las filas 170- 300, las columnas 1-260 y las imágenes 650-1400.

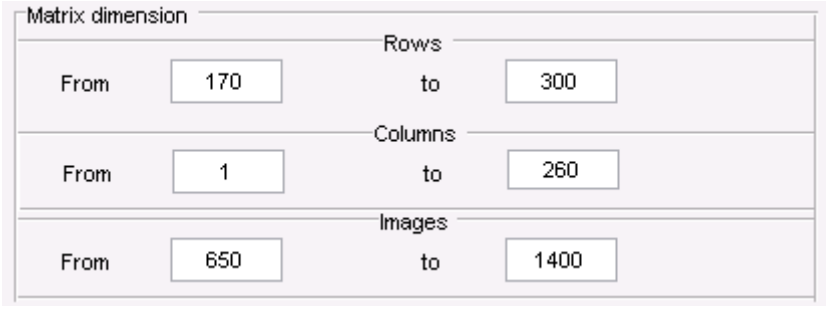


Figura A. 8 Dimensiones de la matriz

Se escoge el plano XZ, ya que es el que mejor representa la vista del fémur.

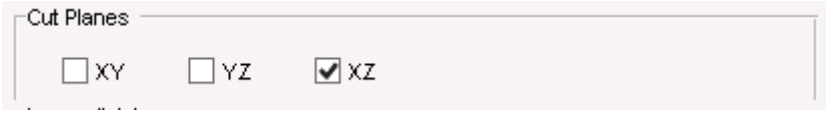


Figura A. 9 Plano de corte

Dada las dimensiones de la matriz, se escogen 2 divisiones en X y 8 divisiones en Y.

Figura A. 10 Número de divisiones

Se escogen las variables de la segmentación con una tolerancia *Tol* de 0.8 y una tolerancia *t* de 0.1, como se aconsejaba anteriormente.

Figura A. 11 Variables de la segmentación

Se escogen las variables del postproceso con los porcentajes *Alimit* y *alimit* de 0.04 y 0.01 respectivamente, un ángulo límite de 5 grados y un valor de gris límite de 220.

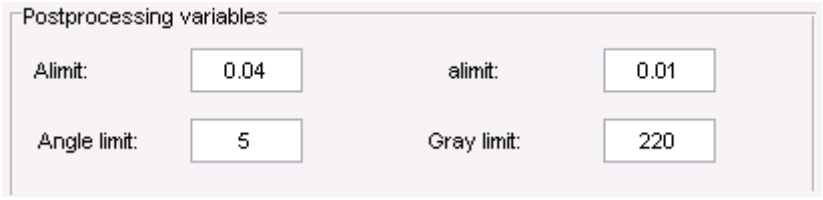


Figura A. 12 Variables del postproceso

Por último, se escogen el número de capas de seguridad de píxeles, con un valor de 3.

Figura A. 13 Número de capas de seguridad

Por último, se aprieta el botón *OK*.

El usuario ya no debe interactuar más con la aplicación hasta que ésta finalice devolviendo la matriz lógica *LogicMat*. En este caso, trabajando con 8 workers, la aplicación ha tardado alrededor de 30 minutos en obtener la matriz lógica.

A continuación, se muestra el resultado de la matriz lógica a la izquierda, y la matriz lógica multiplicada por la matriz que contiene el fémur a la derecha.

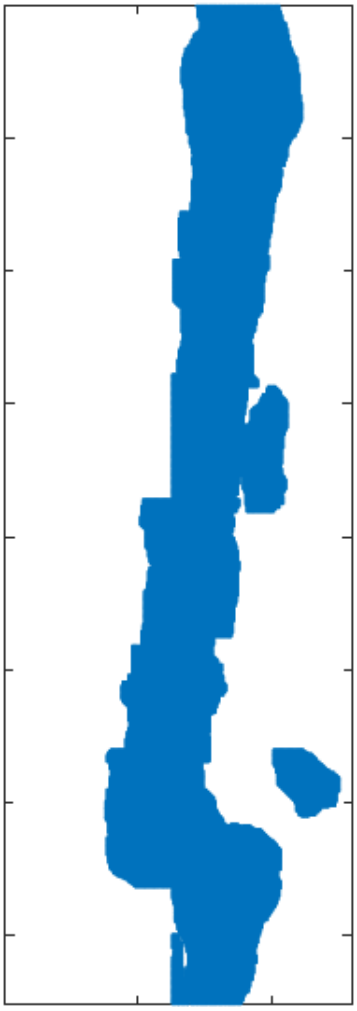


Figura A. 14 Matriz lógica del fémur

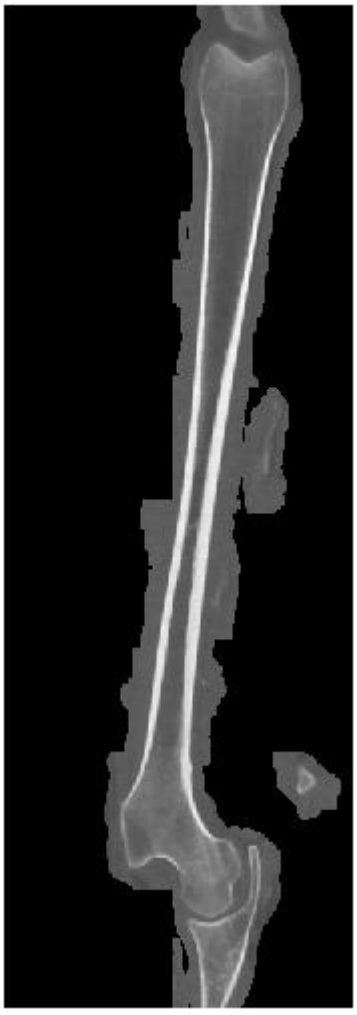


Figura A. 15 Fémur filtrado con la matriz lógica

A.7. Recomendaciones

Se recomienda al usuario realizar una pequeña prueba, con una sola imagen, antes de lanzar el proceso entero. De esta manera, el usuario puede buscar las variables que mejor se acoplen a su trabajo.

B. Manual del programador

En el siguiente anexo se va a detallar la estructura del programa con las funciones más relevantes que lo componen. El código está escrito principalmente en el programa Matlab, mientras que el código referente a la segmentación se ha escrito en el programa FreeFem++.

B.1. *LogicalMatrix*

LogicalMatrix es la función principal del programa, donde se desarrolla la estructura principal y se obtiene la matriz lógica *LogicMat*.

Tabla B.1. 1 Inputs y Outputs de la función LogicalMatrix

Inputs	Outputs
-	*LogicMat*

Subfunciones que componen *LogicalMatrix*.

Tabla B.1. 2 Subfunciones de la función LogicalMatrix

Nombre	Inputs	Outputs	Descripción
Directory	-	-	Crea los directorios *LogicMats*, *Meshes* y *Variables* en caso de que no existan, o los vacían de elementos en caso de que existan.
ImMat	-	*ImgFigure*	Lee las imágenes médicas y las apila en la matriz *ImgFigure* en formato 16 bit y escala de grises 255.
Variables	*planes* *xdiv* *ydiv* *tol* *t* *Alimit*	*ImgFigure*	Crea las variables necesarias para el programa a partir de una ventana de adquisición de datos.

	Alimit *Tetalimit* *Graylimit* *SecLayers* *ImgFigure*		
ImDivision	*Xdiv* *Ydiv* *ImgSize*	*PGMSizes*	Crea la matriz *PGMSizes* que contiene las dimensiones de las divisiones de la imagen.
Preprocessing	*j* *k* *PGMSizes* *SFigure*	-	Crea las imágenes en formato PGM.
Segmentation	*j* *k* *tol* *t*	-	Realiza la segmentación de las imágenes obtenidas en el preproceso.
Postprocessing	*i* *j* *k* *SFigure* *PGMSizes* *Alimit* *Alimit* *Tetalimit* *Graylimit*	*SLogicMat*	Obtiene la matriz lógica de las mallas obtenidas en la segmentación.
SaveLogicMat	*planes*	*LogicMat*	Combina las matrices lógicas de cada plano de corte en una única matriz lógica.
DeleteDomains	*LogicMat*	*LogicMat*	Elimina dominios inconexos.

Security	*LogicMat* *SecLayers*	*LogicMat*	Crea una capa de seguridad de píxeles alrededor de los bordes en la matriz lógica.

Variables de la función:

Tabla B.1. 3 Variables de la función LogicalMatrix

Nombre	Descripción
VMax	Valor máximo del rango de valores en la escala Hounsfield en los que trabaja el programa.
VMin	Valor mínimo del rango de valores en la escala Hounsfield en los que trabaja el programa.
VRange	Indica con un valor de 1 que los valores por encima del rango [VMax-VMin] tienen un valor de 255. Con un valor de 0 los valores por encima del rango tienen un valor de 0.
planes	Vector que indica los planos de corte con los que se trabaja. 1 para el plano XY, 2 para el plano YZ y 3 para el plano XZ.
rowmin	Valor mínimo del rango de filas de la matriz *ImgFigure* con el que se trabaja.
rowmax	Valor máximo del rango de filas de la matriz *ImgFigure* con el que se trabaja.
colmin	Valor mínimo del rango de columnas de la matriz *ImgFigure* con el que se trabaja.
colmax	Valor máximo del rango de columnas de la matriz *ImgFigure* con el que se trabaja.
imagemin	Valor mínimo del rango de imágenes de la matriz *ImgFigure* con el que se trabaja.
Imagemax	Valor máximo del rango de imágenes de la matriz *ImgFigure* con el que se trabaja.
xdiv	Número de divisiones en el eje X que se va a hacer en cada imagen.
ydiv	Número de divisiones en el eje Y que se va a hacer en cada imagen.
Tol	Valor de la tolerancia relacionado con el número de iteraciones en la segmentación.

t	Valor de la tolerancia relacionado con la precisión de malla en la segmentación.
Alimit	Valor del porcentaje de área máxima, en tanto por 1, que deben tener los primeros elementos que se eligen en el postproceso.
alimit	Valor del porcentaje de área máxima, en tanto por 1, que deben tener los elementos que envuelven al primer elemento elegido.
Tetalimit	Valor que define el ángulo por debajo del cual los elementos se consideran estirados.
Graylimit	Valor que define el nivel de gris en la escala de grises 255 por debajo del cual los elementos no se consideran relevantes.
SecLayers	Número de capas de seguridad de píxeles que envuelven a los bordes de la matriz lógica.
ImgFigure	Matriz que contiene las imágenes con las que se desea trabajar.
ImgSize	Vector con el tamaño de la matriz *ImgFigure*.
PGMSizes	Matriz que contiene la información de las dimensiones de cada división de la imagen.
logicmat	Matriz lógica de cada plano de corte.
i	Índice del bucle que recorre cada plano de corte.
j	Índice del bucle que recorre cada imagen de la matriz *ImgFigure* en el plano *i*.
k	Índice del bucle que recorre cada división de la imagen *j*.
LogicMat	Matriz lógica resultado de la combinación de cada matriz de cada plano de corte.

Primeramente, se ejecuta las funciones *Directory*, para crear los directorios donde se guardan los resultados del trabajo, *ImMat*, para guardar las imágenes con las que se va a trabajar y *Variables*, para crear las variables del proceso.

La función está compuesta por bucles y subbucles que recorren los planos de corte, imágenes y divisiones hasta crear la matriz lógica *LogicMat*.

Existe un bucle principal en el cual se especifica en que plano se va a realizar el proceso. Este bucle se realiza con un *for* con valores de *i* obtenidos en la variable *planes*. En este bucle se generan tres casos, siendo el primero para el plano XY, el

segundo para el plano YZ y el tercero para el plano XZ. En los tres casos se crea la matriz *PGMSizes* con la información de cada división, mientras que solo en los casos 2 y 3 (*YZ* y *XZ* respectivamente), se debe permutar la matriz *ImgFigure* para que las dos primeras dimensiones de la matriz sean *YZ* o *XZ*.

Dentro del bucle de cada plano de corte hay un bucle por número de imágenes, se realiza con un *for*, con valores desde *j* igual a 1 hasta el número de imágenes de la matriz *ImgFigure*.

Previamente a entrar en el bucle que hace un barrido por todas las imágenes, se inicializa la matriz lógica, *logicmat*. Esta matriz se irá actualizando con la matriz de salida de la función *Postprocessing*, ensamblándola en la parte correspondiente a la imagen *j* y la división *k*.

Dentro del bucle de cada imagen hay diferentes bucles para el preproceso, segmentación y postproceso. Cada uno de ellos van desde *k* igual a 1 hasta el número de divisiones por imagen.

La segmentación se realiza con un bucle *parfor* para paralelizar el proceso y optimizar el tiempo de ejecución, debido a que la segmentación es el proceso más costoso de todo el código. El preproceso y postproceso se realizan con un bucle en serie *for* y no en un bucle paralelizado debido a que se les pasan matrices muy grandes, que en un bucle paralelizado ralentizaría el proceso.

Al finalizar cada iteración del bucle principal se restaura la matriz *ImgFigure* y se permuta la matriz lógica obtenida *logicmat* para que todas las matrices de cada plano de corte estén orientadas de la misma manera y así poder combinarse entre sí. Además, se guarda la matriz en el directorio *LogicMats* con el nombre *LogicMatXY*, *LogicMatXZ* o *LogicMatYZ*, según el plano de corte elegido. De esta manera, al iniciar otra iteración con otro plano de corte, la matriz lógica de la iteración anterior no estará almacenada en memoria, no aumentando el coste computacional.

Para finalizar, se ejecuta la función *SaveLogicMat*, que combina las diferentes matrices lógicas de cada plano de corte que se haya realizado en una única matriz lógica, *LogicMat*, se ejecuta la función *DeleteDomains* para borrar los pequeños dominios de elementos inconexos y se ejecuta la función *Security* que crea una capa de seguridad de píxeles.

Por último, se guarda la matriz lógica *LogicMat* en la carpeta *LogicMats*.

Tabla B.1. 4 Líneas donde hay mayor coste computacional

Línea	Código	% Tiempo	Descripción
40	parfor k=1:xdiv*ydiv	86.5%	Segmentación
44	logicmat(PGMSizes(1,1,k):PGMSi...	12.9%	Postproceso

B.2. *ImMat*

ImMat es una subfunción dentro de la función *LogicalMatrix* que lee las imágenes médicas y las apila en una matriz en formato 16 bit en escala de grises.

Tabla B.2. 1 Inputs y Outputs de la función ImMat

Inputs	Outputs
-	*Matrix*

Subfunciones que componen *ImMat*.

Tabla B.2. 2 Subfunciones de la función ImMat

Nombre	Inputs	Outputs	Descripción
ReadCatImage	Files	Matrix	Lee las imágenes de un directorio y las guarda en la matriz *Matrix*.
ReadCatDicom	Files	Matrix	Lee las imágenes DICOM de un directorio y las guarda en la matriz *Matrix*.

Variables de la función:

Tabla B.2. 3 Variables de la función ImMat

Nombre	Descripción
Matrix	Matriz que contiene las imágenes del directorio.
Path	Link del directorio donde están contenidas las imágenes.
Files	Nombres de los archivos dentro del directorio.
Image	Variable lógica que dice si el primer archivo del directorio es un archivo DICOM o no.

Primeramente, pide el directorio donde están contenidas las imágenes médicas y añade el path. A continuación, guarda en *Files* los nombres de los ficheros del directorio. Es importante que los únicos archivos de este directorio sean las imágenes y estén todos en el mismo formato.

Según el tipo de extensión de las imágenes, la función llamará a *ReadCatDicom* para imágenes DICOM, o *ReadCatImage* para cualquier otro tipo de imágenes, ambas crearán la matriz *Matrix* con la información de las imágenes. La estructura de estas dos funciones es la misma, salvo que en la primera se leen las imágenes con el comando *dicomread* y en el segundo con *imread*. En ambas se busca el número de dimensiones relevantes, ya que es posible que éstas ya estén contenidas en una matriz 3D. Por tanto, se considera una dimensión relevante cuando es mayor que 1. Las imágenes se guardarán en formato doble.

Una vez obtenida la matriz *Matrix*, se guarda en la carpeta *Variables* para que pueda ser utilizada en la función *Input*.

B.3. *Variables*

Variables es una subfunción dentro de la función *LogicalMatrix* que obtiene las variables del programa mediante una ventana de adquisición de datos.

Tabla B.3. 4 Inputs y Outputs de la función Variables

Inputs	Outputs
planes	*Matrix*
xdiv	
ydiv	
tol	
t	
Alimit	
Alimit	
Tetalimit	
Graylimit	
SecLayers	

Matrix	

Subfunciones que componen *ImMat*.

Tabla B.3. 5 Subfunciones de la función Variables

Nombre	Inputs	Outputs	Descripción
Input	-	-	Accede a la interfaz gráfica para que el usuario introduzca las variables

Variables de la función:

Tabla B.3. 6 Variables de la función Variables

Nombre	Descripción
Matrix	Matriz que contiene las imágenes del directorio.
planes	Vector que indica los planos de corte con los que se trabaja. 1 para el plano XY, 2 para el plano YZ y 3 para el plano XZ.
xdiv	Número de divisiones en el eje X que se va a hacer en cada imagen.
ydiv	Número de divisiones en el eje Y que se va a hacer en cada imagen.
Tol	Valor de la tolerancia relacionado con el número de iteraciones en la segmentación.
t	Valor de la tolerancia relacionado con la precisión de malla en la segmentación.
Alimit	Valor del porcentaje de área máxima, en tanto por 1, que deben tener los primeros elementos que se eligen en el postproceso.
alimit	Valor del porcentaje de área máxima, en tanto por 1, que deben tener los elementos que envuelven al primer elemento elegido.
Tetalimit	Valor que define el ángulo por debajo del cual los elementos se consideran estirados.
Graylimit	Valor que define el nivel de gris en la escala de grises 255 por debajo del cual los elementos no se consideran relevantes.
SecLayers	Número de capas de seguridad de píxeles que envuelven a los bordes de la matriz lógica.

Primeramente, define los valores por defecto de las variables del proceso.

A continuación, llama a la función *Input* donde el usuario define las variables del proceso en una interfaz gráfica. El comando *uiwait* permite esperar a que se cierre la interfaz gráfica. Seguidamente lee las variables que se hayan generado en la carpeta *Variables*.

Crea la variable *planes* en función de los planos que se hayan elegido en la interfaz gráfica.

Se ajusta el tamaño de la matriz *Matrix* de acuerdo al rango de filas, columnas e imágenes que se ha especificado.

A continuación, se transforma los valores de Matrix de la escala Hounsfield a la escala de grises 255 en formato 16 bit. Para desechar valores de partes del cuerpo que no se deseen, se supone un rango de valores comprendidos entre *VMax* y *VMin*. Todo ello se realiza la siguiente operación:

$$V_{GS} = \frac{V_{HF} - V_{min}}{V_{max} - V_{min}} \cdot 255$$

Donde:

- V_{max}, valor máximo *VMax*.
- V_{min} valor mínimo *VMin*.
- V_{HF}, valor en la escala Hounsfield de cada píxel.
- V_{GS}, valor en la escala de grises 255 de cada píxel.

En caso de que la variable *VRange* tenga un valor de 1, todos los valores de *Matrix* por encima de *VMax* tendrán un valor de 255 en la escala de grises 255, mientras que, si *VRange* tiene un valor de 0, todos los valores de *Matrix* por encima de *VMax* tendrán un valor de 0.

Es posible que exista un offset para que el mínimo valor de la matriz sea 0. La función lo tiene en cuenta, y aplica este offset a las variables *VMax* y *VMin*.

B.4. *ImDivision*

ImMat es una subfunción dentro de la función *LogicalMatrix* que crea la matriz con las dimensiones de cada división.

Tabla B.4. 1 Inputs y Outputs de la función ImDivision

Inputs	Outputs
Xdiv *Ydiv* *ImgSize*	*PGMSizes*

Subfunciones que componen *ImDivision*.

Tabla B.4. 2 Subfunciones de la función ImDivision

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

Tabla B.4. 3 Variables de la función ImDivision

Nombre	Descripción
PGMSizes	Matriz que contiene la información de las dimensiones de cada división de la imagen.
xdiv	Número de divisiones en el eje X que se va a hacer en cada imagen.
ydiv	Número de divisiones en el eje Y que se va a hacer en cada imagen.
ImgSize	Vector con el tamaño de la matriz *ImgFigure*.
Srows	Número de filas de cada división.
Scols	Número de columnas de cada división.
x	Índice del bucle que recorre las divisiones en x.
y	Índice del bucle que recorre las divisiones en y.

Frecuentemente, dividiendo el número total de píxeles en cada dimensión por *xdiv* o *ydiv*, no se obtendrá un número entero, lo cual genera un problema ya que no se puede dividir un píxel, por lo que se ha propuesto redondear al alza al siguiente entero el número de píxeles por división excepto en la última, que tendrá el número de píxeles hasta llegar al número de píxeles de la imagen original. De esta forma, no existirá discordancia entre los píxeles de la imagen original y de las partidas

(p. ej. dividiendo una imagen de *451x451* en *4x4* partes, cada parte debería tener *112.75x112.75*, al redondear al alza, cada parte tendrá *113x113*, excepto las últimas que tendrán *112x112* para cuadrar el número de píxeles).

Primeramente, se obtiene el número de filas y columnas de cada división, *Srows* y *Scols*, cogiendo el número entero al alza de la división de las dimensiones de la matriz *Matrix*.

Posteriormente se inicializa la matriz *PGMSizes* y se rellena en un bucle que recorre cada división en x e y. En *PGMSizes* se escribe el píxel donde empieza, el píxel donde acaba y el número total de píxeles de cada división, las filas en la primera columna y las columnas en la segunda columna. Tiene una tercera dimensión para cada división.

B.5. *Preprocessing*

Preprocessing es una subfunción dentro de la función *LogicalMatrix* que crea los archivos PGM.

Tabla B.5. 1 Inputs y Outputs de la función Preprocessing

Inputs	Outputs
j *k* *PGMSizes* *SFigure*	-

Subfunciones que componen *Preprocessing*.

Tabla B.5. 2 Subfunciones de la función Preprocessing

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

Tabla B.5. 3 Variables de la función Preprocessing

Nombre	Descripción

j	Índice del bucle que recorre cada imagen de la matriz *ImgFigure* en el plano *i*.
k	Índice del bucle que recorre cada división de la imagen *j*.
PGMSizes	Matriz que contiene las dimensiones de cada división de la imagen.
SFigure	Matriz que contiene la información de la división *k* de la imagen *j*.

La función *Preprocessing* genera una imagen en formato PGM con el nombre *Figure_j_k.pgm*, donde *j* representará el número de imagen, y *k* la división de cada imagen.

Hay que reseñar que *FreeFem++* lee las imágenes traspuestas, por lo que la matriz *SFigure* que se le pase a *Preprocessing* tiene que estar traspuesta.

B.6. *Segmentation*

Segmentation es una subfunción dentro de la función *LogicalMatrix* que realiza la segmentación.

Tabla B.6. 1 Inputs y Outputs de la función Segmentation

Inputs	Outputs
j *k* *tol* *t*	-

Subfunciones que componen *Segmentation*.

Tabla B.6. 2 Subfunciones de la función Segmentation

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

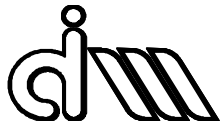


Tabla B.6. 3 Variables de la función Segmentation

Nombre	Descripción
j	Índice del bucle que recorre cada imagen de la matriz *ImgFigure* en el plano *i*.
k	Índice del bucle que recorre cada división de la imagen *j*.
Tol	Valor de la tolerancia relacionado con el número de iteraciones en la segmentación.
t	Valor de la tolerancia relacionado con la precisión de malla en la segmentación.

La función devuelve la malla generada en FreeFem en forma de archivo con el nombre *Mesh_j_k.msh*, donde *j* es el número de imagen y *k* el número de división.

Dentro de la función se genera el archivo con el que trabaja FreeFem. Todo el código de FreeFem se escribe en este archivo con el comando *fprintf* y se guarda como *Code_j_k.edp*.

Una vez generado el fichero se ejecuta con el comando *system* y posteriormente se borran las imágenes PGM y los códigos EDP, para no acumular ficheros ya utilizados.

Es muy importante resaltar que cuando se trabaja con la ventana de comandos desde Matlab, se debe que finalizar el proceso una vez ha acabado, en caso contrario, el programa se queda atascado en este punto. En el sistema operativo Windows, esto se realiza con el comando del CMD *TaskKill* y el nombre del programa que se quiere finalizar. Poniendo este comando junto con FreeFem++.exe al final del código EDP, finaliza Freefem. No obstante, dentro de un bucle paralelizado, genera un problema, ya que el primer proceso que llegue al final del código finalizará todo FreeFem, finalizando el resto de procesos que no han acabado.

Para solucionar este problema, se ha propuesto crear tantas instancias de FreeFem como *workers* tenga Matlab. Esto es tan sencillo como copiar el archivo *FreeFem++.exe* y renombrarlo.

El worker en el que trabaja cada iteración del bucle paralelizado se obtiene con el comando *getCurrentTask*, en la subestructura *ID*. Por ello, la última línea de código del archivo EDP debe llamar al comando *TaskKill* para que finalice el programa

FreeFem++ID.exe, donde *ID* se corresponde al worker que está trabajando. De esta manera, cuando se finalice un proceso, no afectará al resto.

B.7. *Postprocessing*

Postprocessing es una subfunción dentro de la función *LogicalMatrix* que obtiene la matriz lógica *SLogicMat* de las mallas obtenidas en la segmentación.

Tabla B.7. 1 Inputs y Outputs de la función

Inputs	Outputs
i *j* *k* *SFigure* *PGMSizes* *Alimit* *Alimit* *Tetalimit* *Graylimit*	*SLogicMat*

Subfunciones que componen *Postprocessing*.

Tabla B.7. 2 Subfunciones de la función Postprocessing

Nombre	Inputs	Outputs	Descripción
ReadMsh	-	*N* *E*	Lee la malla de la segmentación y devuelve la topología y numeración.
InPíxels	*index* *N* *E* *SLogicMat*	*SLogicMat*	Actualiza la matriz *SLogicMat* con los píxeles contenidos en el elemento *index*.

AroundElements	*index* *E* *elempicked* *aroundelem*	*elempicked* *aroundelem*	Guarda los elementos alrededor del elemento *index* en el vecto *aroundelem*.

Variables de la función:

Tabla B.7. 3 Variables de la función Postprocessing

Nombre	Descripción
i	Índice del bucle que recorre cada plano de corte.
j	Índice del bucle que recorre cada imagen de la matriz *ImgFigure* en el plano *i*.
k	Índice del bucle que recorre cada división de la imagen *j*.
PGMSizes	Matriz que contiene las dimensiones de cada división de la imagen.
SFigure	Matriz que contiene la información de la división *k* de la imagen *j*.
Alimit	Valor del porcentaje de área máxima, en tanto por 1, que deben tener los primeros elementos que se eligen en el postproceso.
alimit	Valor del porcentaje de área máxima, en tanto por 1, que deben tener los elementos que envuelven al primer elemento elegido.
Tetalimit	Valor que define el ángulo por debajo del cual los elementos se consideran estirados.
Graylimit	Valor que define el nivel de gris en la escala de grises 255 por debajo del cual los elementos no se consideran relevantes.
N	Nodos de la malla.
E	Elementos de la malla.
nelements	Número de elementos.
L	Matriz con la dimensión de los lados de cada elemento.
teta	Matriz con los ángulos de cada elemento. Posteriormente se convierte en un vector lógico, siendo 1 aquellos elementos con un ángulo máximo mayor que *Tetalimit*.
area	Vector con el área de cada elemento.
areamax	Valor del área máxima de la malla.

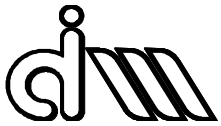
Gray	Vector con el nivel de gris de cada elemento. Posteriormente se convierte en un vector lógico, siendo 1 aquellos elementos con un valor de gris menor que *Graylimit*.
info	Vector que multiplica los vectores *area·teta·Gray*, de forma que contenga el área de los elementos válidos.
elempicked	Vector que indica que elementos han sido analizados.
nmax	Número máximo de iteraciones del bucle que recorre la malla.
index	Índice del elemento a analizar.
aroundelem	Vector con los elementos que envuelven al elemento *index*.
SLogicMat	Matriz lógica correpondiente a la división *k* de la imagen *j*.

Dentro de la función, primeramente, se lee la malla *Mesh_j_k*, generada en la segmentación con la función *ReadMsh*, que devuelve la numeración y topología de la malla con las variables *N* y *E*, siendo los nodos y elementos respectivamente. Esta información se guarda en la carpeta *Mesh* y en la subcarpeta 1, 2 o 3, siendo referentes a los planos XY, YZ o XZ respectivamente. La malla generada en la segmentación se borra para no tener ficheros que no se van a utilizar.

Se inicializa la matriz *SLogicMat* como una matriz lógica de 1 con las dimensiones que marca *PGMSizes* correspondiente a la división *k*.

Las mallas obtenidas de la segmentación están compuestas de triángulos lineales, para realizar el postproceso se necesita información de estos triángulos, como la longitud de los lados, los ángulos, el área y el nivel de gris de cada triángulo. Esta se consigue empleando relaciones trigonométricas sencillas, como el teorema de Pitágoras, el teorema del coseno o el baricentro de un triángulo.

- **Lados**: Conociendo la posición de los nodos del triángulo, los lados se obtienen mediante el teorema de Pitágoras. La información de los lados de todos los triángulos se guarda en la matriz *L*.
- **Ángulos**: Los ángulos del triángulo se calculan a partir de los lados con el teorema del coseno. La información se guarda en el vector lógico *teta*, donde se asigna 1 si el mínimo ángulo del triángulo es mayor que la variable *Tetalimit* o 0 si es menor. El valor de *Tetalimit* se define como el ángulo límite a partir del cual un elemento se considera muy estirado y, por tanto, este elemento define el borde.



- **Área**: El área del triángulo se calcula como la mitad de la base por la altura, donde la base será el lado 1, y la altura será el lado 3 multiplicado por el seno del ángulo 1. La información de las áreas se guarda en el vector *area*, además se guarda el área más grande en la variable *areamax*.
- **Nivel de gris**: El nivel de gris que tiene el triángulo se corresponde al píxel que contiene el baricentro del triángulo. En la figura 12 se puede ver el baricentro del triángulo marcado en rojo, y el píxel donde cae el baricentro marcado en azul. La información se guarda en el vector lógico *Gray*, donde se asigna 1 si el nivel de gris es menor que la variable *Graylimit* o 0 si es mayor. El valor de *Graylimit* se define como el nivel de gris por debajo del cual no se considera hueso en la escala de grises con la que se está trabajando en este trabajo.

Se crea la matriz *info* que contiene el valor de las áreas de aquellos elementos que cumplen las condiciones de valor de gris menor que *Graylimit* y ángulo mínimo mayor que *Tetalimit,* y el número del elemento correspondiente a estos valores, de esta forma se descartan aquellos elementos que sí que formarían parte del hueso. También se crea el vector lógico *elempicked* que guarda con 1 aquellos elementos que no han sido analizados y con 0 aquellos que sí.

La función entra dentro de un bucle *for* donde irá seleccionando elementos que no formen parte del hueso hasta encontrar un borde. De esta manera, aparecen regiones de elementos confinados entre bordes separadas entre sí, por ello, cada iteración del bucle busca encontrar estas regiones. Se define con la variable *nmax* un máximo de regiones inconexas, por defecto tiene un valor de 10.

Primeramente, el bucle selecciona el elemento con el área más grande que no haya sido descartado o analizado. Esto se consigue obteniendo el índice del máximo valor del producto entre *info* y *elempicked*. Se guarda el índice del elemento en la variable *index* y se asigna un 0 a la fila que se corresponde a este elemento en el vector *elempicked*, para anotar que este elemento ha sido analizado.

Se exige una condición *if* al elemento elegido, viendo si su área es mayor que un porcentaje del área máxima de toda la malla. Este porcentaje en tanto por 1 viene representado por la variable *Alimit*. De no existir esta condición, es posible que se seleccionasen elementos que forman parte del hueso, pero tienen un color de gris menor. Si no se cumple esta condición se vuelve al inicio del bucle.

En caso de cumplir la condición, se deseleccionan los píxeles que se encuentran dentro del elemento con la función *InPíxels*, y se guardan los elementos que comparten al menos un nodo con el elemento elegido en *aroundelem*, con la función *AroundElements*.

Se entra en un bucle *while* que se ejecutará mientras *aroundelem* contenga elementos. En este bucle se irán añadiendo los elementos que rodean a cada elemento de *aroundelem* dentro del propio *aroundelem*, hasta que se llegue al borde y se hayan analizado todos los elementos dentro de ese borde. Por lo tanto, este bucle se retroalimenta a sí mismo.

Para comprobar qué elementos de *aroundelem* se analizan, se exige una condición *if* que pregunta si el área de ese elemento es mayor que un porcentaje del área máxima de toda la malla, si el elemento no está estirado y si su nivel de gris es menor que *Graylimit*. La primera condición se consigue con la variable *alimit* que representa el porcentaje en tanto por 1. Es menos restrictiva que *Alimit* ya que es posible que haya elementos más pequeños. No obstante, es necesaria esta condición porque es posible que en ciertas partes del borde no haya elementos estirados, pero sí una concentración alta de elementos muy pequeños. La segunda condición busca si el valor del vector *teta* correspondiente al elemento es 1 o 0, para comprobar si está estirado o no.

En caso de cumplir ambas condiciones se llama a las funciones *InPíxels* para deseleccionar los píxeles que se encuentren dentro del elemento y a *AroundElements* para seleccionar los elementos que rodean al elemento. Por último, se borra el elemento de *aroundelem* para que el bucle *while* no entre en un bucle infinito.

Una vez finalizado el bucle *for* se obtiene la matriz lógica *SLogicMat*, no obstante, hay que invertir las filas de esta matriz, ya que para valores de *y* de la malla se corresponde a la parte más baja de la imagen, mientras que en la matriz los valores más bajos de *y* se dan en la parte más alta.

Tabla B.7. 4 Líneas donde hay mayor coste computacional

Línea	Código	% Tiempo	Descripción
62	[aroundelem,elempicked]=Around...	46.4%	Encontrar elementos alrededor
5	[N,E]=ReadMsh(['Mesh_' num2str...	35.7%	Leer mallas .msh

B.8. *InPíxels*

InPíxels es una subfunción dentro de la función *Postprocessing* que actualiza la matriz *SLogicMat* con los píxeles contenidos en el elemento elegido.

Tabla B.8. 1 Inputs y Outputs de la función InPíxels

Inputs	Outputs
index *N* *E* *SLogicMat*	*SLogicMat*

Subfunciones que componen *InPíxels*.

Tabla B.8. 2 Subfunciones de la función InPíxels

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

Tabla B.8. 3 Variables de la función InPíxels

Nombre	Descripción
index	Índice del elemento a analizar.
N	Nodos de la malla.
E	Elementos de la malla.
xv	Coordenadas en X de los nodos del elemento *index*.
yv	Coordenadas en Y de los nodos del elemento *index*.
BBx	Malla de píxeles en la coordenada X.
BBy	Malla de píxeles en la coordenada Y.
VP	Matriz con el producto vectorial de cada lado con la malla de píxeles.

in	Matriz lógica con los píxeles que se encuentran dentro del elemento *index*.
SLogicMat	Matriz lógica correpondiente a la división *k* de la imagen *j*.

Se guardan en *xv* e *yv*, la posición de los nodos del elemento en x e y respectivamente, y se crea un bounding box dentro de la matriz con una cuadrícula donde cabe el elemento, de esta manera no es necesario trabajar con toda la matriz, solo con el bounding box, disminuyendo el coste computacional.

Para todos los píxeles de la cuadrícula se realiza el producto vectorial entre ellos y los vectores que se corresponden a cada lado del triángulo. Aquellos píxeles cuyo producto vectorial con cada uno de los lados sea mayor o igual que 0, quiere decir que se encuentran dentro del triángulo, y se guardan en la matriz lógica *in*. Esto es debido a que la numeración de los nodos de los elementos de la malla que devuelve FreeFem están ordenados en sentido antihorario, en caso contrario, se debería buscar los píxeles cuyo producto vectorial no fuese positivo.

Por último, se realiza una operación booleana de intersección entre el boundig box de *SLogicMat* e *in,* para asignar un 0 a los elementos de la matriz *SLogicMat* correspondientes a los píxeles que se encuentren dentro del elemento.

B.9. *AroundElements*

AroundElements es una subfunción dentro de la función *Postprocessing* que guarda los elementos alrededor del elemento elegido en el vector *aroundelem*.

Tabla B.9. 1 Inputs y Outputs de la función AroundElements

Inputs	Outputs
index	
E	*elempicked*
elempicked	*aroundelem*
aroundelem	

Subfunciones que componen *AroundElements*.

Tabla B.9. 2 Subfunciones de la función AroundElements

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

Tabla B.9. 3 Variables de la función AroundElements

Nombre	Descripción
index	Índice del elemento a analizar.
E	Elementos de la malla.
elempicked	Vector que indica que elementos han sido analizados.
aroundelem	Vector con los elementos que envuelven al elemento *index*.
nodes	Nodos del elemento *index*.
i	Índice del bucle que recorre todos los nodos del elemento *index*.
a	Elementos que comparten al menos un nodo con el elemento *index*.
j	Índice del bucle que recorre todos los elementos de *a*.

La función devuelve *elempicked* actualizado con los elementos que se han analizado y *aroundelem* con los elementos que se han añadido.

La función busca, para todos los nodos del elemento *index*, aquellos elementos con los que al menos comparte un nodo, dentro de la topología de la malla y los guarda en la variable *a*. Esto se realiza con el comando *find*.

A cada elemento que envuelve al elemento seleccionado, se le exige la condición de saber si ha sido analizado previamente o no, observando el vector *elempicked*. En caso de no haber sido analizado, este elemento se añade a *aroundelem* y se deselecciona de *elempicked* para que no pueda volver a ser analizado.

En Figura B.9. 1 y Figura B.9. 2 se puede observar el resultado de la función *AroundElements*. La leyenda de colores es la siguiente:

- Elemento amarillo, se corresponde al elemento *index* que se pasa como argumento a la función.
- Elementos azules, son los elementos que al menos comparten un nodo con el elemento *index* y no han sido seleccionados previamente. Estos elementos se añaden al vector *aroundelem*.
- Elementos verdes, son los elementos que al menos comparten un nodo con el elemento *index* pero han sido seleccionados previamente. Estos elementos no se añaden al vector *aroundelem*.

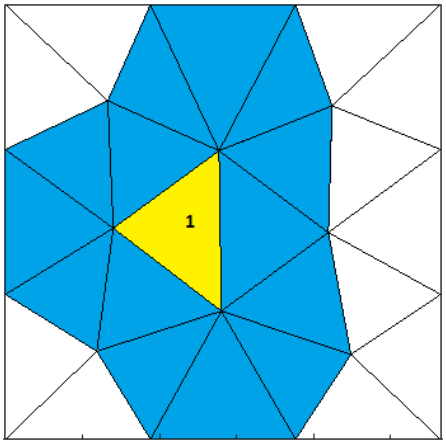


Figura B.9. 1 Elementos alrededor del elemento 1

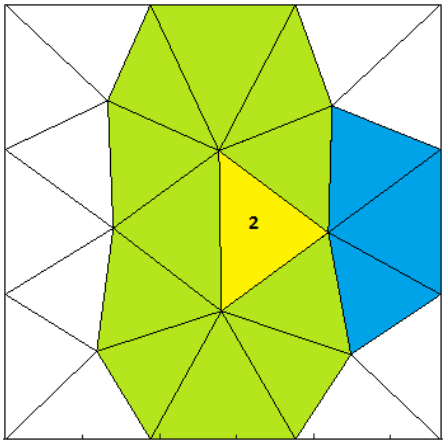


Figura B.9. 2 Elementos alrededor del elemento 2

Tabla B.9. 4 Líneas donde hay mayor coste computacional

Línea	Código	% Tiempo	Descripción
6	[a,~]=find(E==nodes(i));	96.0%	Encontrar elementos que comparten al menos un nodo

B.10. *SaveLogicMat*

SaveLogicMat es una subfunción dentro de la función *LogicalMatrix* que combina las diferentes matrices lógicas de cada plano de corte en la matriz *LogicMat*.

Tabla B.10. 1 Inputs y Outputs de la función SaveLogicMat

Inputs	Outputs

planes	*LogicMat*

Subfunciones que componen *ImMat*.

Tabla B.10. 2 Subfunciones de la función SaveLogicMat

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

Tabla B.10. 3 Variables de la función SaveLogicMat

Nombre	Descripción
planes	Vector que indica los planos de corte con los que se trabaja. 1 para el plano XY, 2 para el plano YZ y 3 para el plano XZ.
i	Índice del bucle para cada plano de corte.
LogicMatXY *LogicMatYZ* *LogicMatXZ*	Matrices lógicas de cada plano de corte, XY, YZ y XZ.
h	Tamaño del vector *planes*.
LogicMat	Matriz lógica combinación de las matrices de cada plano de corte.

Dentro de la función, primero se guardan en memoria las matrices lógicas de cada plano de corte que se habían guardado en el directorio *LogicMats*.

Según el número de planos de corte que se haya realizado en el proceso, se crean diferentes casos para obtener la matriz lógica *LogicMat*:

- 1 plano de corte, la matriz lógica se crea como la matriz lógica de este plano de corte.
- 2 planos de corte, la matriz lógica se crea como la combinación de estos dos planos de corte. En *LogicMat*, están los píxeles que al menos están contenidos en una de las dos matrices de cada plano.

- 3 planos de corte, la matriz lógica se crea como la combinación de estos dos planos de corte. En *LogicMat*, están los píxeles que al menos están contenidos en dos de las tres matrices de cada plano.

B.12. *DeleteDomains*

DeleteDomains es una subfunción dentro de la función *LogicalMatrix* que elimina aquellos dominios de elementos que no están conectados al dominio más grande.

Tabla B.11. 1 Inputs y Outputs de la función DeleteDomains

Inputs	Outputs
LogicMat	*LogicMat*

Subfunciones que componen *Security*.

Tabla B.11. 2 Subfunciones de la función DeleteDomains

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

Tabla B.11. 3 Variables de la función Security

Nombre	Descripción
LogicMat	Matriz lógica.
CC	Estructura que contiene información del número de dominios y del tamaño de cada uno.
Index	Índice de los dominios.
i	Índice del bucle.

Primeramente, se obtienen los diferentes dominios inconexos de píxeles de la matriz lógica *LogicMat* con el comando *bwconncomp*. Este comando crea una estructura que contiene 4 campos:

- *Connectivity*: Conectividad de los elementos que forman un dominio.
- *ImageSize*: Tamaño de la imagen.
- *NumObjects*: Número de dominios.
- *PíxelIdxList*: Vector de celdas con el tamaño de elementos de los diferentes dominios.

Se ordena de mayor a menor los diferentes dominios según el número de elementos que los componen y se guardan los índices en la variable *Index*.

Dentro de un bucle *for*, se borran todos los dominios a excepción del primero, que será el que mayor número de elementos tenga.

B.12. *Security*

Security es una subfunción dentro de la función *LogicalMatrix* que crea una capa de seguridad de píxeles alrededor de los bordes en la matriz lógica.

Tabla B.12. 4 Inputs y Outputs de la función Security

Inputs	Outputs
LogicMat *SecLayers*	*LogicMat*

Subfunciones que componen *Security*.

Tabla B.12. 5 Subfunciones de la función Security

Nombre	Inputs	Outputs	Descripción
-	-	-	-

Variables de la función:

Tabla B.12. 6 Variables de la función Security

Nombre	Descripción
SecLayers	Número de capas de seguridad de píxeles que envuelven a los bordes de la matriz lógica.
xsize *ysize* *zsize*	Tamaño de la matriz *LogicMat*.
LM	Matriz lógica *LogicMat*.
LM0	Matriz lógica *LogicMat* de la iteración anterior.
LogicMat	Matriz lógica con las capas de seguridad.

Básicamente, la función genera una capa de píxeles moviendo la matriz en las tres dimensiones, X, Y, Z, y en las diagonales XY, YZ, XZ y XYZ.

Para conseguirlo se crea la matriz lógica *LM0* que hace referencia a la matriz resultante de la iteración anterior (inicialmente tiene el valor de *LogicMat*), y la matriz resultante de la iteración *LM* (inicialmente también vale *LogicMat*). Ambas matrices se mueven en las direcciones descritas y se combinan en la matriz *LM* de forma que contenga cualquier píxel que al menos esté en una de las dos matrices.

La función realiza esta operación dentro de un bucle *for* tantas veces como capas de seguridad *SecLayers* se hayan definido.

B.13. *Code_j_k*

En este trabajo existen cinco archivos EDP:

- *Code_j_k*
- *make_metrica_freefem_new*
- *make_patch_freefem_new*
- *make_recovery*
- *sparse2full*

El primero es el código principal, mientras que el resto generan el mallado anisótropo adaptativo de acuerdo a la teoría desarrollada por S. Perotto y S. Micheletti.

Code_j_k es el código principal que realiza la segmentación en FreeFem, donde *j* hace referencia a la imagen y *k* a la división. El fichero se genera para cada división *k* de cada imagen *j* dentro de la función *Segmentation*.

En el código se deben especificar como argumentos:

- El nombre de la imagen PGM con la que va a trabajar, *figure_j_k.pgm*.
- El nombre de la malla de salida que devuelve, *Mesh_j_k.msh*.
- Las tolerancias *tol* y *t*, con las que trabaja el código.

Primeramente, se lee la imagen PGM y se crea una malla inicial *Th*. Esta malla es solo una base a partir de la cual se generarán las mallas del proceso adaptativo. Se debe tener en cuenta que FreeFem transpone las imágenes al leerlas.

Se inicializan las variables con las que va a trabajar el código y se llama a la rutina *make_metrica_freefem_new*. En esta rutina, y sus subrutinas *make_patch_freefem_new*, *make_recovery* y *sparse2full*, se genera la métrica que gobernará las mallas del proceso adaptativo.

Posteriormente se realiza el proceso adaptativo, hasta obtener la malla final *Th,* dentro de un bucle *while*, que se ejecutará mientras se cumplan dos condiciones:

- El error sea menor que una tolerancia *tol*. Este error se calcula como la diferencia de elementos entre la malla inicial y la malla obtenida. Por defecto la tolerancia *tol* tendrá un valor de 0.5.
- El número de iteraciones del bucle sea menor que un número de iteraciones máximo *nmax*, que por defecto será 100.

Dentro del bucle se crea la malla *Th* de acuerdo a la métrica generada en las rutinas. Para la creación de la métrica se debe especificar la tolerancia *t* que por defecto tiene un valor de 0.1. Esta tolerancia equivale a la tolerancia τ descrita en la teoría del mallado anisótropo adaptativo.

Por último, se guarda la malla *Th* en *Mesh_j_k.msh*, y se finaliza FreeFem.