Universidad Nacional Autónoma de México

Facultad de Ciencias

UN SISTEMA PARA LA GENERACIÓN NUMÉRICA DE MALLAS ESTRUCTURADAS EN REGIONES CON AGUJEROS

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
MATEMÁTICO
PRESENTA:

GUSTAVO ADOLFO GARCÍA CANO

DIRECTOR DE TESIS:
DR. PABLO BARRERA SÁNCHEZ

2011

Datos del Alumno:	
Autor	
Apellido Paterno:	García
Apellido Materno:	Cano
Nombre(s):	Gustavo Adolfo
Teléfono:	044 55 15 02 41 39
Universidad:	Universidad Nacional Autónoma de México
Facultad o Escuela:	Facultad de Ciencias
Carrera:	Matemáticas
No. de Cuenta:	40601311-4
Datos del Tutor:	
Grado:	Dr.
Apellido Paterno:	Barrera
Apellido Materno:	Sánchez
Nombre(s):	Pablo
Datos del Sinodal 1:	
Grado:	M. en C.
Apellido Paterno:	González
Apellido Materno:	Flores
Nombre(s):	Guilmer Ferdinand
Datos del Sinodal 2:	
Grado:	Dr.
Apellido Paterno:	Domínguez
Apellido Materno:	Mota
Nombre(s):	Francisco Javier
Datos del Sinodal 3:	
Grado:	M. en C.
Apellido Paterno:	Velázquez
Apellido Materno:	Guerrero
Nombre(s):	Luis Carlos
Datos del Sinodal 4:	
Grado:	Mat.
Apellido Paterno:	Vázquez
Apellido Materno:	Maisón
Nombre(s):	Luis Alberto
Datos del Trabajo Escrito:	
Título:	Un sistema para la generación numérica de mallas estructuradas en regiones con agujeros
Subtitulo:	
No. de Páginas:	161
Año:	2011

A mi madre y familia (consanguínea o no)
Por su apoyo, compañia y abrazos

Agradecimientos

Deseo agradecer a: Mi Madre por el esfuerzo invertido en mi a lo largo de la vida.

Dr. Pablo Barrera por el apoyo que me ha dado, por las quejas/críticas sobre mi trabajo sin las cuales no hubiera crecido.

Guilmer González por el gran apoyo para el avance de este trabajo, los comentarios, correcciones, y aguantar mis necedades.

Francisco Domínguez por el apoyo y su visión sobre la generación de mallas, los comentarios tan certeros los cuales considero aumentaron la calidad del contenido de este trabajo.

Luis Carlos Vázquez por los animos y observaciones en este trabajo.

Luis Alberto Vázquez por sus buenas observaciones y apoyo.

Aldo Guzmán por su gran ayuda y apoyo, ayudándome en la demostración del capitulo dos, y sus muy nutritivas platicas.

A mis compañeros del laboratorio de computo científico: Cezar Carreon, Jorge Zavaleta, Javier de Jesús, Leticia Ramirez, por aguantarme con ustedes.

A mis amigos (no pretendo mencionarlos son muchos y ya saben quienes son) por todo su apoyo...

A SEP-PROMEP "Aplicaciones de la optimización numérica a la solución de diversos problemas de cómputo científico"por el apoyo económico que fue bien usado.

Gracias

Prefacio

La discretización de regiones en el plano es fundamental para poder resolver cualquier clase de problema computacional donde la geometría de la región es complicada; una de las aplicaciones más directas es resolver ecuaciones diferenciales parciales. Actualmente la generación de mallas no estructuradas es muy popular ya que existen métodos desde hace tiempo que facilitan la creación de estas mallas sobre regiones complicadas, sin embargo las mallas estructuradas son ampliamente recomendables para algunos problemas ya que son fáciles de usar. El Grupo UNAMALLA dirigido por el Dr. Pablo Barrera a lo largo de más de 20 años ha investigado y desarrollado un método para la generación de mallas estructuradas para regiones simples haciendo uso de diversos funcionales. El cual nos permite generar mallas en regiones no convexas complicadas.

Desde el 2008 he trabajado en el grupo UNAMALLA en el desarrollo de la versión 4.0 del sistema UNAMalla, mi servicio social consistió en añadir al sistema la posibilidad de fijar puntos de la malla, algo de suma importancia para este trabajo. En 2010 el Dr. Pablo Barrera me propuso como trabajo de tesis el desarrollo de un método para generar mallas estructuradas sobre regiones con agujeros, lo que desemboco en el desarrollo de la versión 4.5 del sistema UNAMalla y este escrito.

La generación de mallas sobre regiones con agujeros es un problema que se ha abordado de diferentes formas, lo mas usual es dividir la región de tal forma que se obtengan subregiones en las cuales sea sencillo obtener una malla de cada subregión. Sin embargo, es difícil encontrar una división de forma automática ya que depende fuertemente de la distribución geométrica de la región. En este trabajo proponemos un método que permite trabajar con cualquier región con agujeros y en donde se obtiene una estructura muy sencilla de manejar, con el cual el usuario decide la partición que él requiera.

Antes de exponer nuestro algoritmo, es necesario dar el contexto y las bases del tema. La **introducción** expone de manera informal el concepto de malla y da un recorrido por algunos métodos para la generación de mallas y algunos conceptos básicos, con el objetivo de sentar el contexto del trabajo.

El **primer capítulo** formaliza algunos de los conceptos ya mencionados en la introducción y métodos que son necesarios, además de abordar el trabajo realizado por el Grupo UNAMALLA.

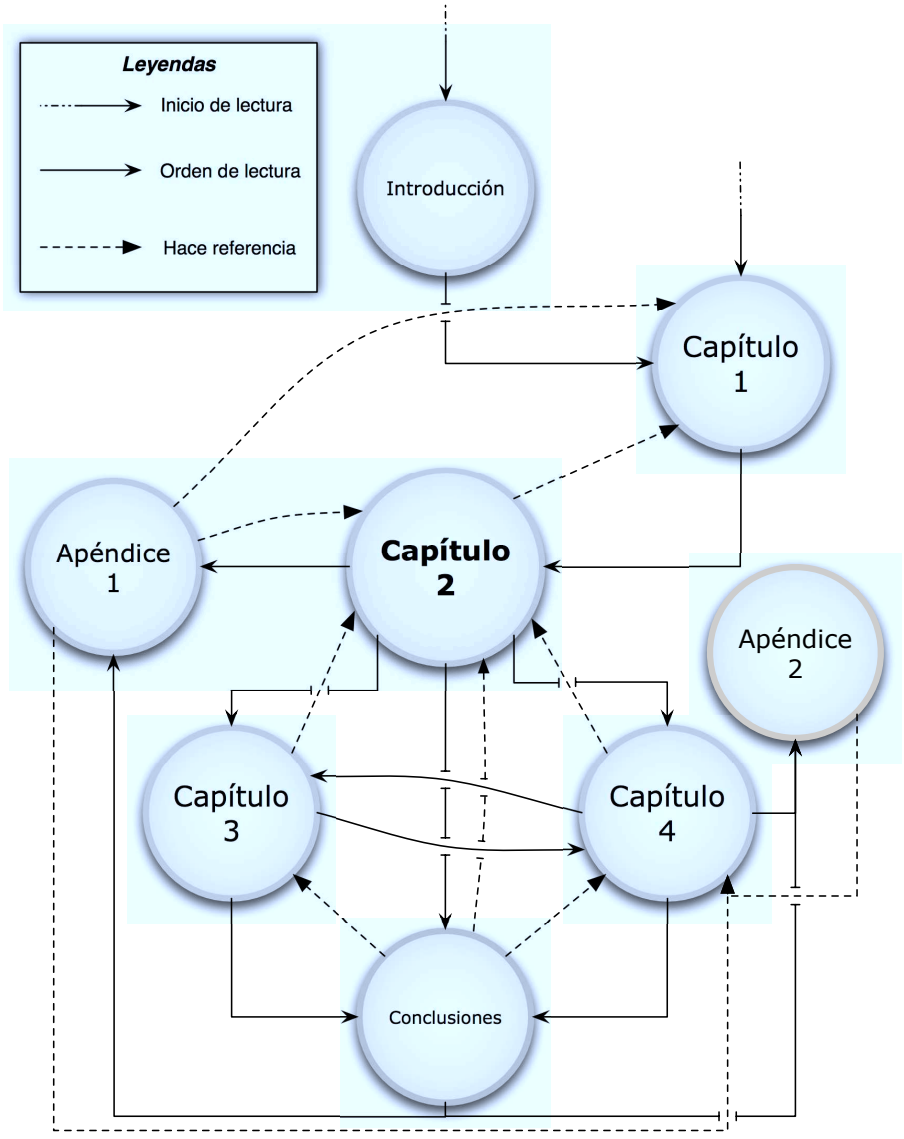
El **segundo capítulo** expone el método que proponemos para la generación de mallas estructuradas sobre regiones con agujeros, antes se presentan las propuestas ya existentes para resolver este problema y se formaliza el dominio de trabajo.

El **tercer y cuarto capítulo** dan una descripción del sistema UNAMalla 4.5 (que se encuentra dividido en dos programas), explica la organización del código y los algoritmos usados dentro del mismo.

El **quinto capítulo** da un resumen del trabajo, se comentan algunas notas finales, y se menciona sobre el trabajo a futuro.

Finalmente se anexa una galería de mallas generados con nuestro método en el sistema UNAMalla 4.5, y un segundo apéndice con el formato de los archivos que genera el sistema.

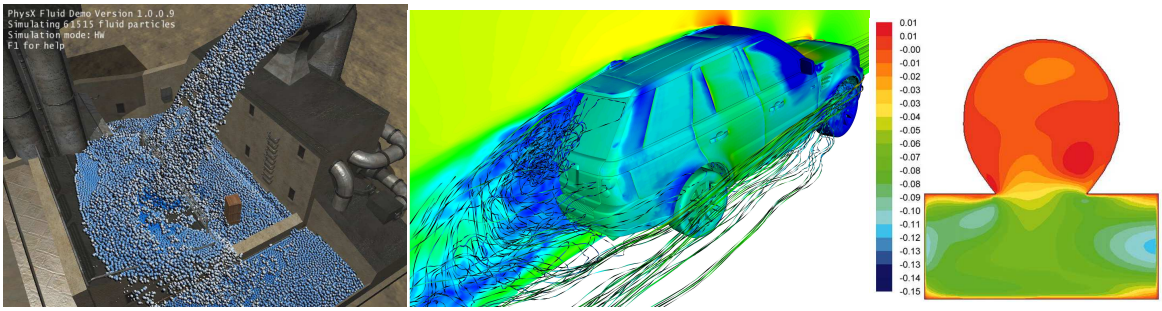
La figura siguiente muestra un mapa sobre los capítulos de la tesis, en donde se muestran las posibles formas de leer el trabajo y las referencias cruzadas entre las partes que lo conforman, con el objetivo de acortar la lectura de este en dependencia de los intereses del lector, ya que por ejemplo el capítulo dos se complementa con la galería de imágenes y los capítulos posteriores.



Índice general

Introducción

En la actualidad es muy común ver programas para la simulación de algunos fenómenos físicos en videojuegos, animaciones por computadora, diseño de automóviles, aviones, edificios; pero para resolver estas simulaciones se requiere de un modelo matemático que represente el problema de manera adecuada. En general dentro de la modelación matemática de varios problemas, la solución de ecuaciones diferenciales e integrales aparecen de manera natural; sin embargo, en muy pocos casos es posible de encontrar la solución analítica de dichas ecuaciones, lo que ha dado pie a la investigación para resolver estas ecuaciones de manera numérica, es decir, aproximar la solución por medio de métodos indirectos. Cuando tenemos las computadoras como herramienta esto cobra sentido, por lo que se han creado varios métodos computacionales para la solución de sistemas de ecuaciones de diversos tipos, como los métodos de elemento finito, diferencia finita, volumen finito, entre otros.



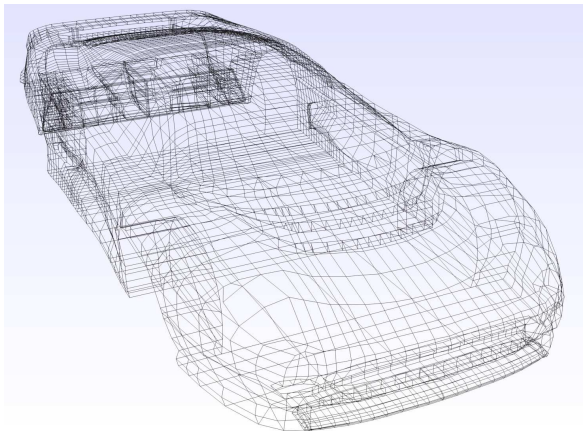
(a) Simulación [20] (b) Simulación de Aerodinámica [7] (c) Solución de una EDP [11]

Figura 1: Aplicaciones de las Mallas

0.1. Planteamiento del Problema

En muchos modelos matemáticos la representación de la región de trabajo (dominio de la ecuación) es importante (Fig. 1), una representación adecuada ayuda a obtener una buena aproximación a la solución de ecuaciones, estas regiones usualmente están modeladas en $\mathbb{R}^n$ con $n \in \{1, 2, 3, 4\}$. Cuando $n = 2$ se dice que el dominio es una región plana, a pesar de no ser el caso más usual en problemas relacionados con objetos de la vida cotidiana, no deja de ser importante; ya que en muchos casos estas regiones planas representan parte de una estructura en tres dimensiones, por ejemplo para analizar la dinámica de un fluido en un lago. El problema que se debe resolver es el dar una representación adecuada del dominio, de tal forma que esta representación pueda ser manejada fácilmente en la computadora, este en realidad son dos problemas: el primero, la representación como estructura de dato para la computadora y el segundo, la representación discreta de la región en $\mathbb{R}^2$, en otras palabras cómo se leen y manejan los datos y como se interpretan (Fig. 2).

(a) Automovil real



(b) Representación en mallas

Figura 2: Representación en mallas

0.2. Conceptos Preliminares

Se le llama malla de una región, a un conjunto de subregiones simples (en nuestro caso polígonos) tales que son ajenas entre si (salvo por la frontera), y la unión de todas ellas es una aproximación a la región (Fig. 3). También esto nos provee una estructura sobre la discretización basada en las conexiones de cada vértice con los demás, de esta manera se hace una clasificación sobre las mallas: a las mallas donde existan dos puntos interiores (contenidos en el interior de la región) con una cantidad de "vecinos" diferente se les llama **mallas no-estructuradas** y a las demás se les llama **mallas estructuradas**; es decir, son las mallas tales que todos los puntos interiores tienen la misma cantidad de vecinos (Fig. 4). Las mallas no-estructuradas son las más usadas ya que existen algoritmos muy conocidos para generarlas.

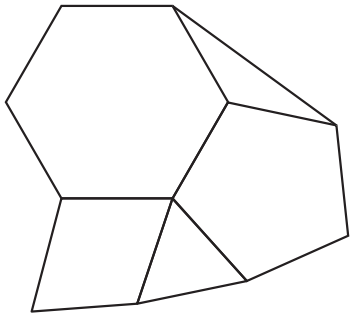
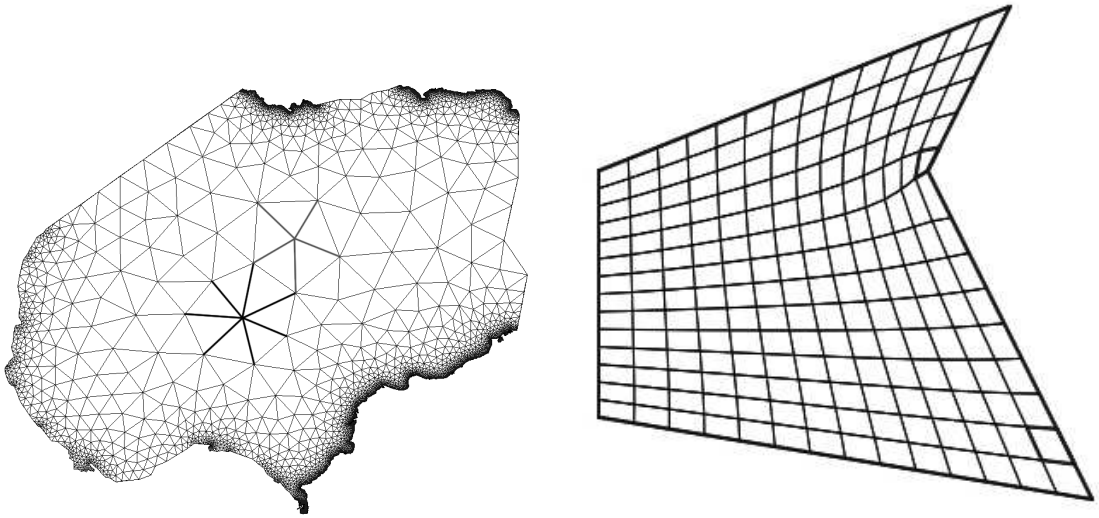


Figura 3: Ejemplo de Malla

Otro concepto importante es el de "celda". Las celdas son las caras (subregiones) que forman la malla, en caso de tener el mismo polígono regular en todas las celdas la malla es estructurada (triángulos, cuadrados, hexágonos), en mallas no estructuradas a veces tienen polígonos de diferente cantidad de lados, y es usual identificar las mallas por el tipo de celda; como triangulaciones, hexagonales, etc. (Fig. 5). Sin embargo, cuando tienen cuadriláteros se les llama simplemente mallas estructuradas, por ser las más usuales, se utilizan para modelos en superficies sumergidas en tres dimensiones, en animaciones, juegos, simulaciones, etc. esto se debe a su facilidad de uso y de creación. Las celdas nos inducen otra forma de pensar la discretización y verla ya no como una colección de puntos, sino como una colección de celdas, donde no sólo nos interesa que las celdas estén totalmente contenidas en la región, también que sean convexas.

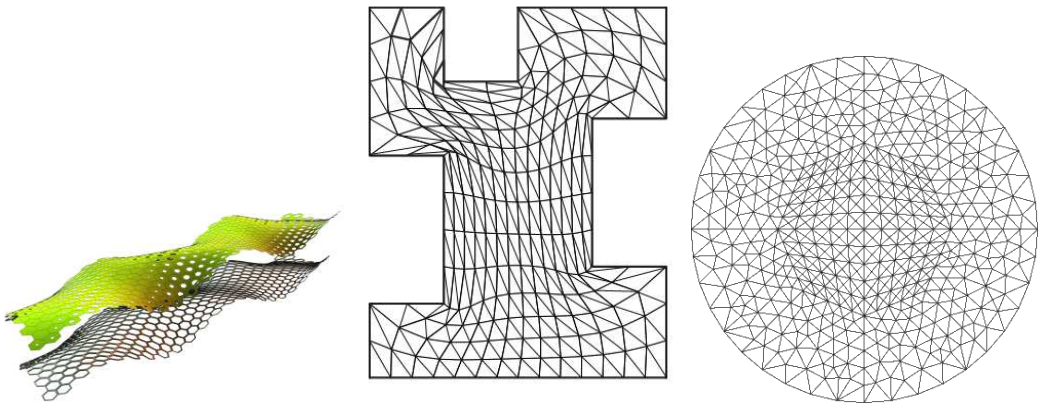
Pese a que las computadoras a lo largo de los años han aumentado su capacidad de cálculo numérico considerablemente, los modelos geométricos necesitan una gran capacidad de cómputo por lo que no deja de ser importante solucionar los problemas de manera eficiente y es conveniente que sea fácil de entender, por la estructura de datos sencilla de las mallas estructuradas se sugiere utilizarlas. [1, 2, 17].

Otro enfoque sobre la discretización de regiones plana consiste en una nube de puntos en el interior de la región con cierta distribución y otros puntos en la frontera; esto se utiliza mucho en problemas donde no es necesaria una estructura conectiva de los puntos; los llamados meshless [3]. Este enfoque también lo satisfacen las mallas.



(a) Mala No-estructurada (b) Malla Estructurada

Figura 4: Ejemplo de los tipos de mallas



(a) Mala Estructurada Hexagonal (b) Triangulación Estructurada (c) Triangulación No-Estructurada

Figura 5: Ejemplo de mallas

0.3. Métodos Usuales o Clásicos para la Generación de Mallas Estructuradas

Este no es un problema nuevo, ya existen diversos métodos para generar mallas estructuradas, muchos de ellos están restringidos a ciertas regiones, entre ellos existe una familia llamados los **métodos de mapeo**. Estos métodos están basados en hacer un homeomorfismo con dominio en alguna región y codominio la región de estudio (Fig. 6). La imagen directa del mapeo de una malla sobre el dominio nos induce una malla para la región de estudio. Existen varios enfoques para generar el mapeo para mallas estructuradas (con celdas cuadrilaterales). Es necesario hablar un poco de ellos ya que se pueden usar como parte del algoritmo que propone este trabajo. Algunos de estos métodos se pueden clasificar de la siguiente manera [22].

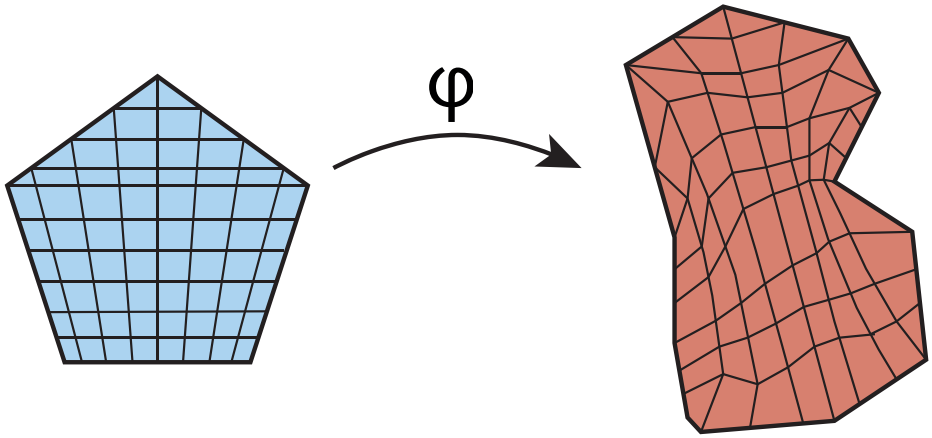


Figura 6: Método del Mapeo (malla estructurada (no todas sus celdas son cuadriláteros))

0.3.1. Algebraicos

Se le llaman así ya que su naturaleza es algebraica, estos métodos son rápidos de calcular aunque no garantizan mallas suaves (con ángulos entre los segmentos son lo menos agudos posibles) y en muchos casos resultan mallas dobladas (celdas no convexas), y en otros se salen del interior de la malla (lo que da una mala discretización de la región). Algunos de estos métodos son:

- Interpolación de Lagrange
- Interpolación de Hermite
- Splines
- Interpolación Transfinita

Esta última es de las más económicas computacionalmente pues solamente utiliza la información de las subfronteras, y es la que utilizamos en el sistema como parte del algoritmo global [15].

0.3.2. Elípticos

Estos métodos se les llama así ya que se basan en dar solución a una ecuación diferencial parcial elípticas condicionadas (los llamados términos de fuerza [forcing terms]), generalmente se formula con ecuaciones de Poisson y se resuelve con métodos iterativos, las mallas generadas por ellos son suaves, algo que generalmente se busca ya que la suavidad da ventajas en ciertos cálculos posteriores a la generación. Sin embargo las celdas tienden a salir del interior de la región cuando esta no es convexa; por lo que no se recomienda en regiones no convexas.

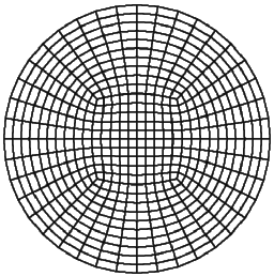


Figura 7: Malla generada con un método elíptico

0.3.3. Hiperbólicos

Se basan en la solución de ecuaciones diferenciales parciales hiperbólicas, estas ecuaciones se resuelven en el exterior de la región, buscan suavizar las líneas de las mallas y lograr cierta ortogonalidad; estos métodos son muy efectivos en problemas de flujos externos donde se tienen condiciones de frontera.(Fig 8)

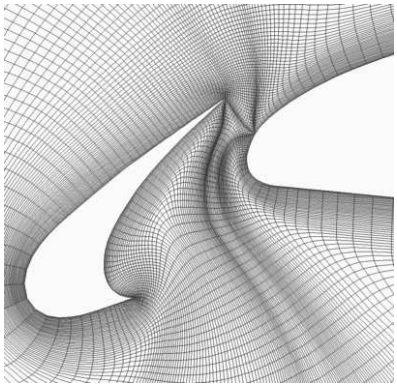


Figura 8: Malla generada con un método Hiperbólico

0.3.4. Ortogonales

Este tipo de métodos buscan construir mallas con celdas cercanas a rectángulares. A las mallas generadas por estos métodos también se les llama ortogonales y tienen muchas ventajas, entre ellas el error de truncamiento al resolver numéricamente una ecuación diferencial parcial por diferencias finitas se ven disminuidos, además de que al discretizar la ecuación se disminuyen los términos cruzados y las condiciones de frontera son más fáciles de representar; aunque estas mallas son muy difíciles de construir. Algunas de las formas de construir estas mallas incluye a los métodos elípticos e hiperbólicos.

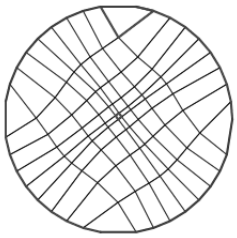


Figura 9: Malla generada con un método ortogonal

0.3.5. Variacionales

Estos métodos buscan el mapeo homeomorfo por medio del mínimo de una familia de funcionales ; generalmente la familia de funcionales esta asociada a alguna propiedad geométrica. Se dividen en dos tipos: los continuos y los discretos.

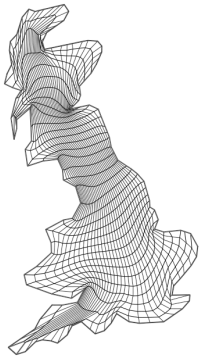


Figura 10: Malla generada con un método variacional discreto

Variacionales Continuos

Este tipo de métodos es introducido por Winslow en 1967. Brackbill y Saltzman en 1982 dan una versión adaptativa y tratan de obtener mallas suaves, Steinberg y Roache son los que dan una generalización mas interesante [21]. La idea de este método consiste en dar funcionales los cuales miden alguna propiedad geométrica de cada celda que nos interese esto se logar a través de buscar una función que minimice esta medida (área, ortogonalidad, longitud, etc.), para encontrar este mínimo se deben resolver las ecuaciones de Euler-Lagrange (hay que recordar que estas son ecuaciones diferenciales parciales).

$$\text{mín} \int_{\Omega} F(x_\xi, x_\eta) d\xi d\eta$$

Variacionales Discretos

El enfoque discreto replantea el problema de encontrar el mapeo homemorfo, como el problema de encontrar coordenadas adecuadas para cada punto dentro de la malla. En un principio se buscaba por medio de un Laplaceano discreto, o auxiliandose de la posición de los nodos vecinos. En los años 80's Steinberg le propuso a Castillo formular los funcionales del propio Steinberg de forma discreta; Castillo en su tesis doctoral propone encontrar un mapeo homemorfo por medio de la solución de un problema de optimización a gran escala y con restricciones de un funcional discreto (empezando por una formulación del funcional de área), posteriormente Barrera propone plantearlo como un problema de optimización sin restricciones y basado en el trabajo de Ivanenko, con ello empieza una linea de investigación, en esta linea Tinoco en 1997 propone una familia de funcionales (los llamados "K-Funcionales") que son parecidos en características a los funcionales armónicos. Barrera y Domínguez Mota en 2005 dan una caracterización para los funcionales con los cuales se pueden obtener mallas convexas; los llamados "Funcionales Convexos" los cuales conservan ciertas características de los "K-Funcionales". En el siguiente capitulo se vera a detalle estos funcionales ya que son los utilizados en el sistema.

Funcionales continuos de Área, Ortogonalidad, Longitud y Combinaciones

Estos funcionales son algunos de los propuestos para dar una buena caracterización de las mallas generadas por los métodos variacionales, se llaman funcionales pues son funciones de funciones, la incógnita es una función; de manera general los funcionales que para este trabajo son importantes tienen la forma:

$$I(\overline{x}) := \int_{\Omega} F(\overline{x}_\xi, \overline{x}_\eta) \quad d\xi d\eta$$

Donde el funcional F da una caracterización de alguna propiedad (La cuál mide el funcional I), de esta manera se han creado varios funcionales.

Funcional de Área Este funcional mide el área de cada celda calcula el área de cada triángulo de la siguiente manera:

$$I_A(\overline{x}) = \frac{1}{2} \int_{\Omega} (x_\xi y_\eta - x_\eta y_\xi)^2 \quad d\xi d\eta$$

Funcional de Longitud Mide la longitud de las curvas que conforman la malla:

$$I_l(\overline{x}) = \int_{\Omega} \|\overline{x}_\xi\|^2 + \|\overline{x}_\eta\|^2 \quad d\xi d\eta = \int_{R} x_\xi^2 + x_\eta^2 + y_\xi^2 + y_\eta^2 \quad d\xi d\eta$$

Donde $\overline{x}_\xi$ y $\overline{x}_\eta$ son las tangentes asociadas a la malla en el punto $\overline{x}$.

Funcional de Ortogonalidad Para medir la ortogonalidad se utiliza el producto punto de los vectores tangentes a las curvas que conforman la malla; donde mientras más cercano sea el producto punto (en norma) a cero más ortogonales son las curvas entre si.

$$I_o(\overline{x}) = \frac{1}{2} \int_{R} (\overline{x_\xi} \cdot \overline{x_\eta})^2 \quad d\xi d\eta = \frac{1}{2} \int_{R} x_\xi x_\eta + y_\xi y_\eta \quad d\xi d\eta$$

Capítulo 1

Teoría moderna para Generación Numérica de Mallas Estructuradas

El profesor Pablo Barrera y su grupo UNAMALLA han desarrollado desde finales de los años 80's una línea de investigación sobre la generación de mallas estructuradas por medio de métodos variacionales discretos. El presente trabajo utiliza y continua con esta línea de investigación, por lo que en este capítulo se expondrán formalmente los conceptos y métodos desarrollados por el Grupo UNAMALLA en los que se basa el algoritmo que proponemos.

1.1. Preliminares y Definiciones

1.1.1. Mallas Estructuradas

Ya se ha descrito de manera informal lo que es una malla estructurada propiamente; sin embargo, en todo este trabajo se va a manejar la siguiente definición:

Definición 1.1. *Malla Estructurada*
Sean $m, n \in \mathbb{N}$ *tales que* $m, n > 2$, $\Omega \subset \mathbb{R}^2$ *definida por una región*[1] *poligonal* ρ *con vértices* $V = \{v_1, v_2, v_3, v_4, \ldots, v_{2m+2n-4}\}$, *cerrada, simple y con orientación positiva. (Fig. 1.1) Una malla estructurada, es una colección de puntos* G *tales que:*

- $G = \{P_{i,j} \in \Omega | i \in \{1, 2, \ldots, m\}; j \in \{1, 2, \ldots, n\}\}$.
- $L_1(G) = \{P_{i,1} \in G | i \in \{1, 2, \ldots, m\}\}$
- $L_2(G) = \{P_{m,j} \in G | j \in \{1, 2, \ldots, n\}\}$
- $L_3(G) = \{P_{i,n} \in G | i \in \{1, 2, \ldots, m\}\}$
- $L_4(G) = \{P_{1,j} \in G | j \in \{1, 2, \ldots, n\}\}$

[1] Con región nos referimos a un conjunto conexo con interior no vacío

Donde $L_1(G) \cup L_2(G) \cup L_3(G) \cup L_4(G)$ *es la frontera de la región de estudio.*

Decimos que G *es una malla estructurada, admisible y discreta si:*

$$\bigcup_{i=1}^{4} L_i(G) \subset \partial\Omega \quad y \quad V \subset \bigcup_{i=1}^{4} L_i(G) \quad y \quad G - (\bigcup_{i=1}^{4} L_i(G)) \subset \Omega^{\circ}$$

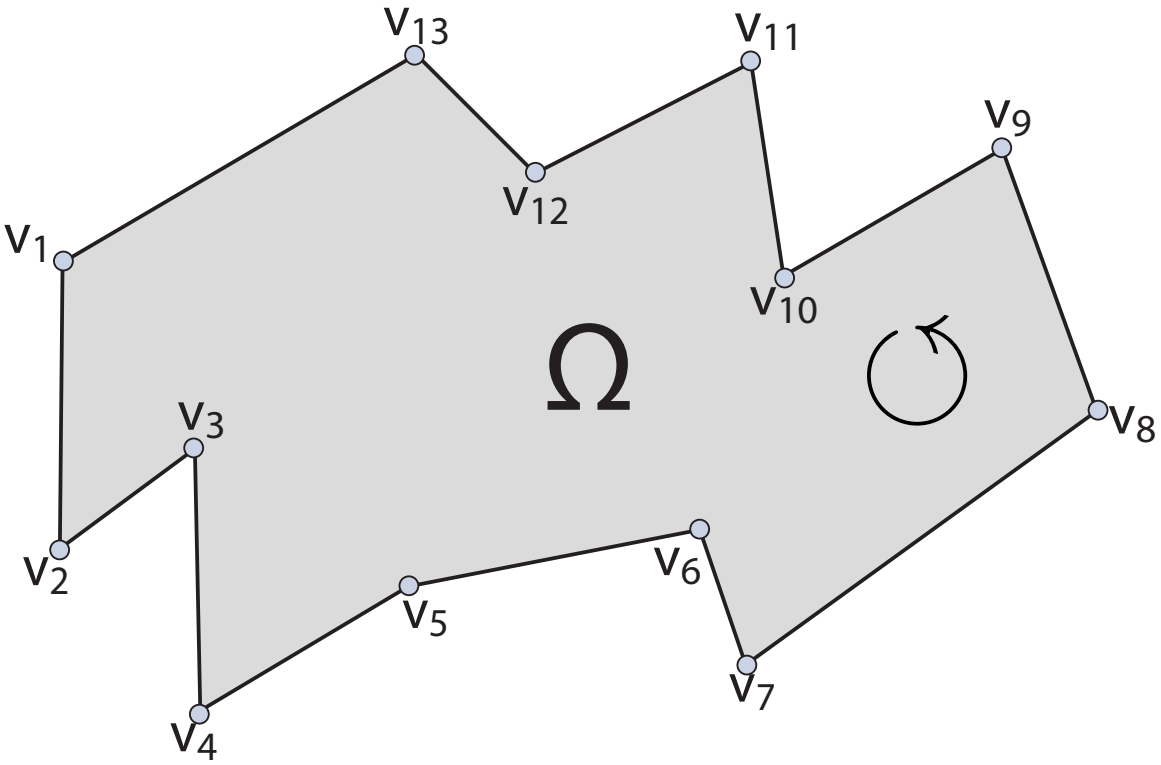


Figura 1.1: Región Ω

Como dije antes, al hablar de mallas, estamos pensando la discretización de la región como una colección de celdas que cubren dicha región, para que el cubrimiento sea una buena discretización es necesario que las celdas sean convexas, lo cual nos induce la definición de una malla convexa.

Definición 1.2. *Malla Convexa*
Decimos que una malla es convexa si todas sus celdas son convexas

Además se tiene el siguiente teorema:

Teorema 1.1. *Celdas Convexas*
Sea $c_{i,j} = \{P_{i,j}, P_{i+1,j}, P_{i+1,j+1}, P_{i,j+1}\}$ *una celda. Si los triángulos orientados de manera positiva:*

$$\triangle P_{i,j}P_{i+1,j}P_{i+1,j+1}$$
$$\triangle P_{i+1,j+1}P_{i,j+1}P_{i,j}$$
$$\triangle P_{i,j+1}P_{i,j}P_{i+1,j}$$
$$\triangle P_{i+1,j}P_{i+1,j+1}P_{i,j+1}$$

tienen área positiva (Fig. 1.2). Entonces $c_{i,j}$ *es convexa.*

Este teorema nos da una caracterización sobre las celdas convexas muy importante, la cuál es suficiente para garantizar la convexidad en una celda(aunque no necesaria). Es fácil ver que la positividad del área de una celda (orientada de manera positiva) no es una condición suficiente para caracterizar su convexidad (Fig. 1.3).

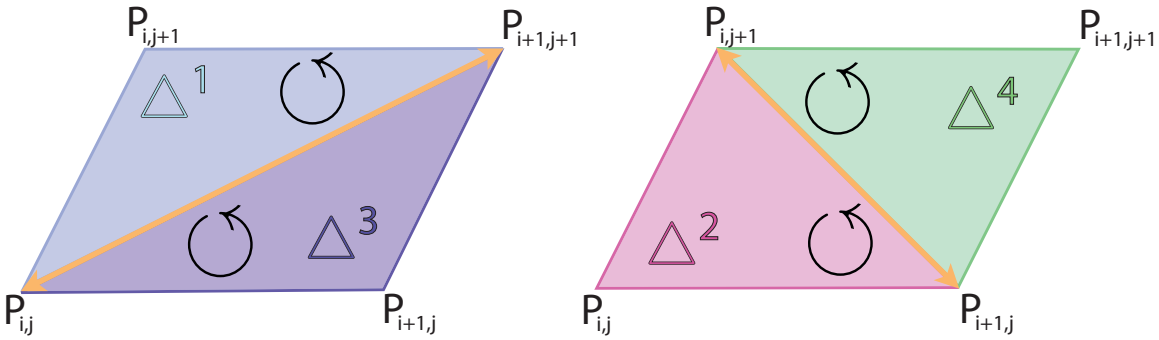


Figura 1.2: Triángulos que induce una celda

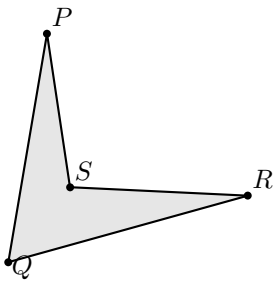


Figura 1.3: Celda no convexa y con área positiva

Se puede plantear el problema de generar una malla estructurada de forma continua de la siguiente manera: Sea $\Omega \in \mathbb{R}^2$ una región cerrada simplemente conexa y $R := [0,1] \times [0,1]$ (al cuál llamaremos "cuadrado unitario"), una función $\chi : R \to \Omega$ homeomorfa y diferenciable (Figura 1.4), donde la malla estructurada de la región Ω es la imagen directa del conjunto de puntos que resultan de hacer una malla estructurada sobre R; por lo tanto para generar la malla hay que buscar una función χ adecuada; es decir, se quiere que las líneas que forman la malla no se "entrelacen".

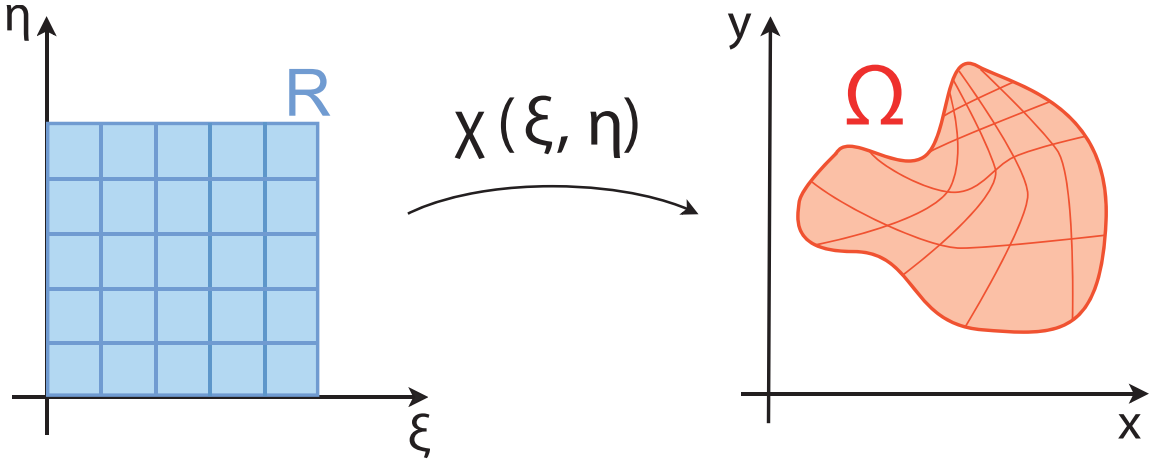


Figura 1.4: Homeomorfismo χ

1.2. Generación Variacional

En el planteamiento anterior para la generación de mallas el funcional χ puede verse como la solución a un problema variacional; a continuación se va a hablar sobre estos métodos de manera formal y se introducirán algunos conceptos necesarios para este.

1.2.1. Formulación Continua

Empezaremos por definir lo que es un funcional.

Definición 1.3. *Funcional*
Un funcional $I(\overline{x})$ es una función con dominio en un subconjunto de un espacio de funciones

Como ya habíamos dicho en la introducción; el enfoque variacional se desarrolla buscando el mínimo de un funcional, donde este mida propiedades geométricas y que se aproximen al homeomorfismo deseado, es decir, el dominio del funcional el cuál vamos a minimizar son las funciones de la forma $x : R \to \Omega$ diferenciable, con inversa continua, de tal manera que la frontera de R vaya a la frontera de Ω. Esto lo hacemos utilizando el siguiente tipo de funcional:

$$I(\overline{x}) := \int_R F(\overline{x}_\xi, \overline{x}_\eta)\, \mathrm{d}(\xi, \eta)$$

donde $F(x)$ es un funcional que da una caracterización geométrica sobre toda la región Ω.

Decimos que cada uno de estos mapeos x es una *malla continua* para Ω. Estos funcionales usualmente miden área, ortogonalidad y longitud. Para obtener información detallada sobre la generación variacional continua véase [9, 10, 15, 22] .

1.2.2. Formulación Variacional Discreta

El enfoque discreto se basa en discretizar directamente los funcionales y verlo como un problema de optimización no lineal de dimensión finita; sin embargo, antes de ver como se hace esto vamos a enunciar un teorema y algunos resultados que justifican los métodos de discretización que utilizamos.

Teorema 1.2.
Sea

$$B := \{B_{i,j} | i = 1, \ldots, m-1, j = 1, \ldots, n-1\} \tag{1.1}$$

una malla del rectángulo unitario R, donde cada $B_{i,j}$ es una celda de la malla.

$$\chi : R \to \Omega \tag{1.2}$$

una función continua con:

$$\chi_{i,j} := \chi \mid_{B_{i,j}} \tag{1.3}$$

la restricción de χ sobre la celda $B_{i,j}$.
Si $\forall \chi_{i,j} \in C^1$ ***y el Jacobiano*** $J_{i,j} > 0$ ***entonces:***
χ ***es un homeomorfismo.***

Esto nos da una caracterización del homeomorfismo que nos servirá para discretizar los funcionales, ya que una forma de hacer las restricciones $\chi_{i,j}$ es usando un mapeo bilinear, esto en la parte discreta es muy conveniente pues celdas rectangulares de R las queremos mandar a celdas cuadrilaterales en Ω.

1.2.3. Mapeo Bilineal

Vamos a usar del mapeo bilinear más sencillo que manda el cuadrado unitario R a cualquier cuadrilátero $c_{i,j}$ con vértices
$P_{i,j}, P_{i+1,j}, P_{i,j+1}, P_{i+1,j+1} \in \mathbb{R}^2$ (Fig. 1.2), el cuál se puede expresar de la siguiente forma:

$$\overline{r}(s,t) := \overline{a}s + \overline{b}st + \overline{c}t + \overline{d} \tag{1.4}$$

Para determinar los coeficientes basta con dar las condiciones de los puntos a los que queremos que vaya:

$$\begin{aligned} \overline{r}(0,0) &\to P_{i,j} \\ \overline{r}(1,0) &\to P_{i+1,j} \\ \overline{r}(1,1) &\to P_{i+1,j+1} \\ \overline{r}(0,1) &\to P_{i,j+1} \end{aligned} \tag{1.5}$$

De lo anterior resulta lo siguiente:

$$\begin{aligned} \overline{d} &= P_{i,j} \\ \overline{a} &= P_{i+1,j} - P_{i,j} \\ \overline{c} &= P_{i,j+1} - P_{i,j} \\ \overline{b} &= P_{i+1,j+1} - P_{i+1,j} - P_{i,j+1} + P_{i,j} \end{aligned} \tag{1.6}$$

De las ecuaciones (1.5), (1.6) reescribimos la ecuación (1.4):

$$\overline{r}(s,t) = (P_{i+1,j} - P_{i,j})s + (P_{i+1,j+1} + P_{i,j} - [P_{i,j+1} + P_{i+1,j}])st + (P_{i,j+1} - P_{i,j})t + P_{i,j} \tag{1.7}$$

Podemos calcular el Jacobiano de dicho mapeo de la siguiente manera:

$$J(s,t) = \overline{r}_s^t J_2^t \overline{r}_t \tag{1.8}$$

donde $J_2 := \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$

$$\begin{aligned} \overline{r}_s &= P_{i+1,j} - P_{i,j} + (P_{i+1,j+1} - P_{i+1,j} + P_{i,j} - P_{i,j+1})t \\ \overline{r}_t &= P_{i,j+1} - P_{i,j} + (P_{i+1,j+1} - P_{i+1,j} + P_{i,j} - P_{i,j+1})s \end{aligned} \tag{1.9}$$

De las ecuaciones (1.8) y (1.9) tenemos que:

$$\begin{aligned} J(s,t) =& (P_{i+1,j} - P_{i,j})^T J_2 (P_{i,j+1} - P_{i,j}) + \\ &+ (P_{i+1,j+1} - P_{i+1,j})^T J_2 (P_{i,j+1} - P_{i,j})s + \\ &+ (P_{i+1,j} - P_{i,j})^T J_2 (P_{i+1,j+1} - P_{i,j+1})t \end{aligned} \tag{1.10}$$

Algo importante de notar es que el Jacobiano en los vértices de R involucra al área de los triángulos:

$$\begin{aligned} J(0.0) &= (P_{i+1,j} - P_{i,j})^T J_2 (P_{i,j+1} - P_{i,j}) = 2\text{área}(\triangle P_{i,j+1} P_{i,j} P_{i+1,j}) \\ J(1,0) &= (P_{i+1,j} - P_{i,j})^T J_2 (P_{i,j+1} - P_{i,j}) + (P_{i+1,j+1} - P_{i+1,j})^T J_2 (P_{i,j+1} - P_{i,j}) \\ &= 2\text{área}(\triangle P_{i,j} P_{i+1,j} P_{i+1,j+1}) \\ J(0,1) &= (P_{i+1,j} - P_{i,j})^T J_2 (P_{i,j+1} - P_{i,j}) + (P_{i+1,j} - P_{i,j})^T J_2 (P_{i,j+1} - P_{i,j}) \\ &= 2\text{área}(\triangle P_{i+1,j} P_{i+1,j+1} P_{i,j+1}) \\ J(1,1) &= (P_{i+1,j} - P_{i,j})^T J_2 (P_{i,j+1} - P_{i,j}) + (P_{i+1,j+1} - \\ &- P_{i+1,j})^T J_2 (P_{i,j+1} - P_{i,j}) + (P_{i+1,j} - P_{i,j})^T J_2 (P_{i,j+1} - P_{i,j}) \\ &= 2\text{área}(\triangle P_{i+1,j+1} P_{i,j+1} P_{i,j}) \end{aligned}$$

El Jacobiano es positivo si los triángulos tienen área positiva.

La definición 1.2 complementada con el teorema 1.1 nos da una condición para la convexidad de la malla que se satisface al trabajar con funcionales que tengan un jacobiano positivo (que cumple el mapeo bilineal), lo que significa que la convexidad no es una restricción sobre los puntos de la discretización si no que viene de la mano con el planteamiento del funcional;*esto permite escribir el problema de optimización de gran escala para generar mallas convexas como un problema sin restricciones.*

1.2.4. Discretización de Funcionales

Definición 1.4. *Funcional Discreto*
Un funcional discreto I sobre una malla discreta G es una función de la siguiente forma:

$$I(G) = \sum_{i.j} f(c_{i,j})$$

donde $c_{i,j}$ es la celda i, j de G y f es función de sus vértices.

Esto hace del método variacional discreto un problema de optimización de gran escala. Se pueden crear los funcionales discretos a partir de los funcionales de la forma continua, consideremos el funcional:

$$I(\overline{x}) = \iint_R f(\overline{x}_\xi, \overline{x}_\eta)\, \mathrm{d}\xi\mathrm{d}\eta \tag{1.11}$$

y una malla uniforme de tamaño $m \times n$ sobre R y llamemos $B_{i,j}$ a la celda i, j de la malla, de tal manera sucede $R = \cup_{i,j} B_{i,j}$, y podemos ver la ecuación (1.11) de la siguiente forma:

$$I(\overline{x}) = \sum_{i=1}^{m}\sum_{j=1}^{n} \iint_{B_{i,j}} f(\overline{x}_\xi, \overline{x}_\eta)\, \mathrm{d}\xi\mathrm{d}\eta. \tag{1.12}$$

Para discretizar el funcional I discretizaremos por cachos, es decir, en cada integral de la ecuación (1.12) tenemos la función $\overline{x}$ restringida a $B_{i,j}$, esta función va de una celda en la malla de R a una celda en la malla de Ω, y lo aproximamos con un mapeo bilineal, y de esta forma inducimos una discretización de la función $\overline{x}$ por cachos, de esta aproximación vamos a tomar una aproximación a la integral y con esto se obtiene el funcional discreto. a continuación vamos a desarrollarlo:

$$I_{i,j} = \iint_{B_{i,j}} f(\overline{x}_\xi, \overline{x}_\eta)\, \mathrm{d}\xi\mathrm{d}\eta \approx \iint_{B_{i,j}} f(\overline{r}_{ij\xi}, \overline{r}_{ij\eta})\, \mathrm{d}\xi\mathrm{d}\eta. \tag{1.13}$$

Esta integral $I_{i,j}$ la podemos aproximar por una *Regla de cuadratura de cuatro puntos*, usando el promedio de los valores en los cuatro vértices (P, Q, R, S) del dominio $(B_{i,j})$:

$$\begin{aligned} I_{i,j} \approx & \frac{1}{4}\left[f(\overline{r}_{ij\xi}(P), \overline{r}_{ij\eta}(P)) + f(\overline{r}_{ij\xi}(Q), \overline{r}_{ij\eta}(Q)) + \right. \\ & \left. + f(\overline{r}_{ij\xi}(R), \overline{r}_{ij\eta}(R)) + f(\overline{r}_{ij\xi}(S), \overline{r}_{ij\eta}(S))\right] \\ & = \widehat{I}_{i,j}. \end{aligned} \tag{1.14}$$

De las ecuaciones (1.12), (1.13), (1.14) y renombrando a P, Q, R, S los vértices de $B_{i,j}$ como $P_{i,j}, P_{i+1,j}, P_{i+1,j+1}, P_{i,j+1}$, obtenemos la siguiente discretización:

$$\begin{aligned} I(\overline{x}) &\approx \sum_{i=1}^{m}\sum_{j=1}^{n} \widehat{I}_{i,j} \\ &= \frac{1}{4}\sum_{i=1}^{m}\sum_{j=1}^{n}[f(\overline{r}_{ij\xi}(P_{i,j}), \overline{r}_{ij\eta}(P_{i,j}) + f(\overline{r}_{ij\xi}(P_{i+1,j}), \overline{r}_{ij\eta}(P_{i+1,j}) + \\ &+ f(\overline{r}_{ij\xi}(P_{i+1,j+1}), \overline{r}_{ij\eta}(P_{i+1,j+1}) + f(\overline{r}_{ij\xi}(P_{i,j+1}), \overline{r}_{ij\eta}(P_{i,j+1})] \\ &= \widehat{I}(G). \end{aligned} \tag{1.15}$$

De esta manera es como serán aproximados los funcionales continuos. Lo que permite discretizar muchos de los funcionales ya existentes en la teoría de los métodos variacionales continuos.

Para mezclar las características que ofrecen los funcionales en una malla, se puede considerar una combinación lineal convexa de ellos; esto fue originalmente discutido por Knuup y Steinberg [15], aunque Roache y Steinberg introdujeron los funcionales del siguiente tipo [21] :

$$I(\overline{x}) = \sigma_1 I_1(\overline{x}) + \sigma_2 I_2(\overline{x}) + \sigma_3 I_3(\overline{x})$$

con $\sigma_1, \sigma_2, \sigma_3 \geq 0$ y $\sigma_1 + \sigma_2 + \sigma_3 = 1$.

En el sistema UNAMalla se utilizan combinaciones de funcionales de la forma:

$$I(x) = \sigma I_1(x) + (1 - \sigma) I_2(x)$$

La elección de los "pesos" σ y los funcionales f depende de las propiedades que se buscan, en algunos reportes se dan valores que en la práctica se consideran adecuados, sin embargo hasta ahora parece no haber una regla para determinar el valor de σ, una combinación habitual que se da es el funcional de área con el de ortogonalidad, aunque existen otras combinaciones.

Hasta aquí se ha visto una forma de dar funcionales discretos basados en funcionales continuos, sin embargo existen funcionales que sólo son posibles de plantear de forma discreta, más adelante se hablará a fondo del que se utiliza en el sistema UNAMalla.

1.2.5. Sobre la ϵ-Convexidad

Debido a la importancia del área para la convexidad de una celda (Def. 1.2) en la generación de las mallas y su caracterización, es usual definir las siguientes medidas.

Definición 1.5. *Dada una malla discreta G de orden $m \times n$, sobre una región Ω, definimos área promedio, mínima, y máxima ($\overline{\alpha}, \alpha_-, \alpha_+$ respectivamente) como:*

$$\overline{\alpha}(G) := \frac{1}{N} \sum_{q=1}^{N} \alpha(\triangle_q)$$

$$\alpha_-(G) := \text{mín}\{\alpha(\triangle_{i,j}^q) | 1 \leq i \leq m \, ; 1 \leq j \leq n \, ; 1 \leq q \leq 4\}$$

$$\alpha_+(G) := \text{máx}\{\alpha(\triangle_{i,j}^q) | 1 \leq i \leq m \, ; 1 \leq j \leq n \, ; 1 \leq q \leq 4\}$$

Donde $\alpha(\triangle_q)$ es el área del triángulo $\triangle_q$, N es la cantidad de triángulos inducidos por las celdas de toda la malla (4 por cada celda) y $\triangle_{i,j}^q$ es el triángulo q-esimo de los 4 triángulos que induce cada celda (Fig. 1.2). De esta forma para garantizar una malla convexa se utiliza como condición que $\alpha_- > 0$.

Esta condición ($\alpha_- > 0$) tiene un problema asociado al error de redondeo numérico. Una primera idea para corregir el problema sería dar una $\epsilon > 0$ y reformular la condición como sigue:

Una malla es ϵ-convexa si y sólo si:

$$\alpha(\triangle_q) > \epsilon \qquad \forall \triangle_q \in G$$

Sin embargo, el problema de este primer intento es que la elección de la ϵ es dependiente de la escala, de la dimensión de la malla y de la región Ω, lo cuál no es conveniente; es necesario contar con

una solución independiente de escala. Para solucionar a esto, vamos primero a notar la siguiente propiedad:

$$\begin{aligned}\overline{\alpha}(G) &= \frac{1}{N}\sum_{q=1}^{N}\alpha(\triangle_q)\\ &= \frac{1}{4(m-1)(n-1)}\sum_{q=1}^{N}\alpha(\triangle_q)\\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{2\alpha(c_{i,j})}{4(m-1)(n-1)}\\ &= \frac{4\text{área}(\Omega)}{4(m-1)(n-1)}\\ &= \frac{\text{área}(\Omega)}{(m-1)(n-1)}\end{aligned} \tag{1.16}$$

Esto significa que $\overline{\alpha}(G)$ es *independiente de* G, sólo depende de m, n, y del área de Ω, sin embargo m, n son fijas, por lo que generalmente se dice que sólo depende de Ω.

La siguiente propiedad nos va a dar la respuesta al problema planteado:

Teorema 1.3. *ϵ-convexidad*
Sea $A : \mathbb{R}^2 \longrightarrow \mathbb{R}^2$ una transformación lineal, no singular.
Si $\hat{G} = A(G) \Longrightarrow$

$$\frac{\alpha(\widehat{\triangle}_q)}{\overline{\alpha}(\hat{G})} = \frac{\alpha(\triangle_q)}{\overline{\alpha}(G)}$$

Donde $\widehat{\triangle}_q = A(\triangle_q)$ y $\triangle_q$ es uno de los triángulos inducidos por las celdas de G

Este teorema lo que nos dice es cuál es la medida de área de los triángulos(en consecuencia de la convexidad de la malla) que es independiente de la escala.

Demostración. Como $\widehat{\triangle}_q = A(\triangle_q) \Longrightarrow$

$$\alpha(\widehat{\triangle}_q) = det(A)\alpha(\triangle_q) \tag{1.17}$$

Por lo tanto:

$$\begin{aligned}\alpha(\hat{G}) &= \sum_{q=1}^{N}\frac{1}{2}\alpha(\widehat{\triangle}_q))\\ &= \sum_{q=1}^{N}\frac{1}{2}\det(A)\alpha(\triangle_q))\\ &= \det(A)\sum_{q=1}^{N}\frac{1}{2}\alpha(\triangle_q))\\ &= \det(A)\alpha(G)\end{aligned} \tag{1.18}$$

por este se deduce:

$$\begin{aligned}\overline{\alpha}(\hat{G}) &= \frac{1}{N}\sum_{q=1}^{N}\alpha(\triangle_q)\\ &= \frac{2}{N}\alpha(\hat{G})\\ &= \frac{2}{N}\det(A)\alpha(G)\\ &= \frac{\det(A)\alpha(G)}{2(m-1)(n-1)}\\ &= \det(A)\overline{\alpha}(G)\end{aligned} \tag{1.19}$$

De las ecuaciones (1.17) y (1.19), obtenemos lo que queríamos:

$$\frac{\alpha(\widehat{\triangle}_q)}{\overline{\alpha}(\hat{G})} = \frac{\det(A)\alpha(\triangle_q)}{\det(A)\overline{\alpha}(G)} = \frac{\alpha(\triangle_q)}{\overline{\alpha}(G)}$$

□

Ahora podemos decir lo que significa que una malla sea ϵ-convexa:

Definición 1.6. *Una malla G de Ω es ϵ-convexa* $\Longleftrightarrow$

$$\frac{\alpha(\triangle_q)}{\overline{\alpha}(G)} > \epsilon \qquad \forall \triangle_q \in G$$

De las definición 1.5, obtenemos la siguiente relación:

$$\alpha_-(G) \leq \overline{\alpha}(G) \leq \alpha_+(G) \tag{1.20}$$

Es fácil ver que la definición 1.6 y la ec. (1.20) tiene la siguiente equivalencia:

Teorema 1.4. *ϵ-convexidad Sea G una malla de Ω es ϵ-convexa* $\Longleftrightarrow$

$$\frac{\alpha_-(G)}{\overline{\alpha}(\Omega)} > \epsilon$$

y que:

$$\frac{\alpha_-(G)}{\overline{\alpha}(\Omega)} \leq 1$$

Entonces $\epsilon \leq 1$ ya que en caso contrario la definición no tiene sentido. Recordemos que este concepto fue introducido a raíz de que si $\alpha_-(G) = 0$ tenemos un problema con el error de redondeo; eso significa que cuando ϵ es muy cercano a cero seguimos conservando el problema, por esta razón se crea el siguiente concepto:

Definición 1.7. *Número de condición de malla*

$$k(G) := \frac{\overline{\alpha}(\Omega)}{\max\{\alpha_-(G) | G \text{ es malla de } \Omega\}}$$

a este número se le llama "Número de condición de malla", y nos da una medida sobre la dificultad del problema de oprimización sobre dicha región; si este número es muy cercano a cero es muy difícil numéricamente y si es muy cercano a uno significa que las celdas son muy uniformes por lo que encontrar una malla adecuada es sencillo.

1.3. Funcionales Cuasi-Armónicos

Los funcionales armónicos son históricamente importantes, este tipo de mapeos se iniciaron por medio del estúdio del operador de Laplace. A las funciones que satisfacen las ecuaciones de Laplace se les llama funciones armónicas, definimos a los mapeos armónicos de la siguiente forma:

Definición 1.8. *Mapeo Armónico Un mapeo armónico es una transformación $\chi : R \longrightarrow \Omega$ inyectiva, tal que:*

- $$\chi(\partial\Omega) = \partial R$$
- *La parametrización que induce χ sobre la frontera de Ω es continua.*
- *Las funciones componentes de χ son funciones armónicas*

Este tipo de funcionales se les llama también *Funcionales de Winslow.*

Al intentar buscar este tipo de mapeos por medio de la solución de un problema de optimización; resulta que se requiere de una malla convexa en cada paso del optimizador, es decir, la malla inicial debe ser convexa, sin embargo construir una malla convexa inicial es el problema que se pretende resolver, pero una de las propiedades más importantes de los mapeos armónicos: *En el óptimo de un mapeo armónico se obtiene una malla tal que los triángulos que inducen sus celdas son lo más cercano a un triángulo isósceles*, esto hace una malla convexa y con una buena distribución uniforme (o lo más uniforme posible) de los puntos y del área de cada celda. Por esta razón es importante encontrar alternativas que no requirieran una malla inicial convexa y que su óptimo tenga esta misma propiedad(la propiedad de los triángulos equiláteros), de ahí que se les nombre *Funcionales Cuasi-Armónicos.*

1.3.1. Funcional Cuasi-Armónico

El primer funcional de este tipo fue resultado de la tesis de Tinoco [4, 5] donde se presenta la familia de funcionales llamada *k-funcionales* y demuestra que son una alternativa a los funcionales armónicos, estos funcionales tienen la siguiente forma:

$$H_k(G) = \sum_{q=1}^{N} \frac{l(\triangle_q) - 2\alpha(\triangle_q)}{k + \alpha(\triangle_q)}$$

no requieren de una malla inicial convexa, y en su óptimo se tiene una malla muy parecida a la que se busca con los funcionales armónicos; sin embargo, los k-funcionales tienen un polo; lo que presenta problemas de estabilidad. En un artículo de Domínguez-Mota [6], encontraron otras familias de funcionales, con el siguiente resultado:

Teorema 1.5. *Sea $f : \mathbb{R} \to \mathbb{R}; f \in C^2$ estrictamente decreciente, no negativa, una malla G de una región $\Omega \in \mathbb{R}^2$ definida por un polígono y $F : \mathbb{R}^N \to \mathbb{R}$ definida como:*

$$F(G) := \sum_{q=1}^{N} f(\alpha_q)$$

Si existe una malla G_0 convexa de Ω entonces es posible encontrar $t \in \mathbb{R}, t \geq 1$ tal que el problema de optimización:

$$\text{mín}\{F(G)|G \in M(\tau_t(\Omega))\}$$

donde $\tau_t(\overline{x}) := t\overline{x}$ y $M(\Omega)$ es el conjunto de las mallas ϵ-convexas de Ω,
tiene solución.

En este espíritu se han creado dos funcionales, uno de ellos también es un funcional cuasi-armónico y el otro no, el primero se le conoce como H_ω o *funcional de suavidad* y se define de la siguiente forma:

$$H_\omega(G) := \sum_{q=1}^{N} \lambda(\triangle_q)\varphi_\omega(\alpha(\triangle_q))$$

Donde $\lambda : \triangle \to \mathbb{R}$ y $\varphi_\omega : \mathbb{R} \to \mathbb{R}$ como sigue:

$$\lambda(\triangle(A, B, C)) = \|A - B\|^2 + \|C - B\|^2 \tag{1.21}$$

$$\varphi_\omega(x) := \begin{cases} 2(\omega - x)/\omega^2 & \text{si } x < \omega \\ 1/x & \text{si } x \geq \omega \end{cases} \tag{1.22}$$

Este funcional fue creado con el objetivo de ser más eficiente que los k-funcionales y es el funcional usado por default en el sistema UNAMalla 4.

1.3.2. Funcional de Área

Otro funcional que se ha desarrollado a partir de las propiedades que se deben satisfacer según el teorema 1.5 se llama $S_{\omega,\epsilon}$ y se define de la siguiente manera:

$$S_{\omega,\epsilon} := \sum_{q=1}^{N} f(\omega(\alpha(\triangle_q) - \epsilon))$$

Donde $\alpha(\triangle_q)$ es el de la definición 1.5 y $f : \mathbb{R} \to \mathbb{R}$ como sigue:

$$f(x) := \begin{cases} (x-1)(x-2) + 1 & \text{si } x < 1 \\ 1/x & \text{si } x \geq 1 \end{cases} \tag{1.23}$$

Este funcional combinado con área-ortogonalidad es usado por default en el sistema UNAMalla 4.5 por mejorar el tiempo de computo en comparación con el H_ω combinado con área.

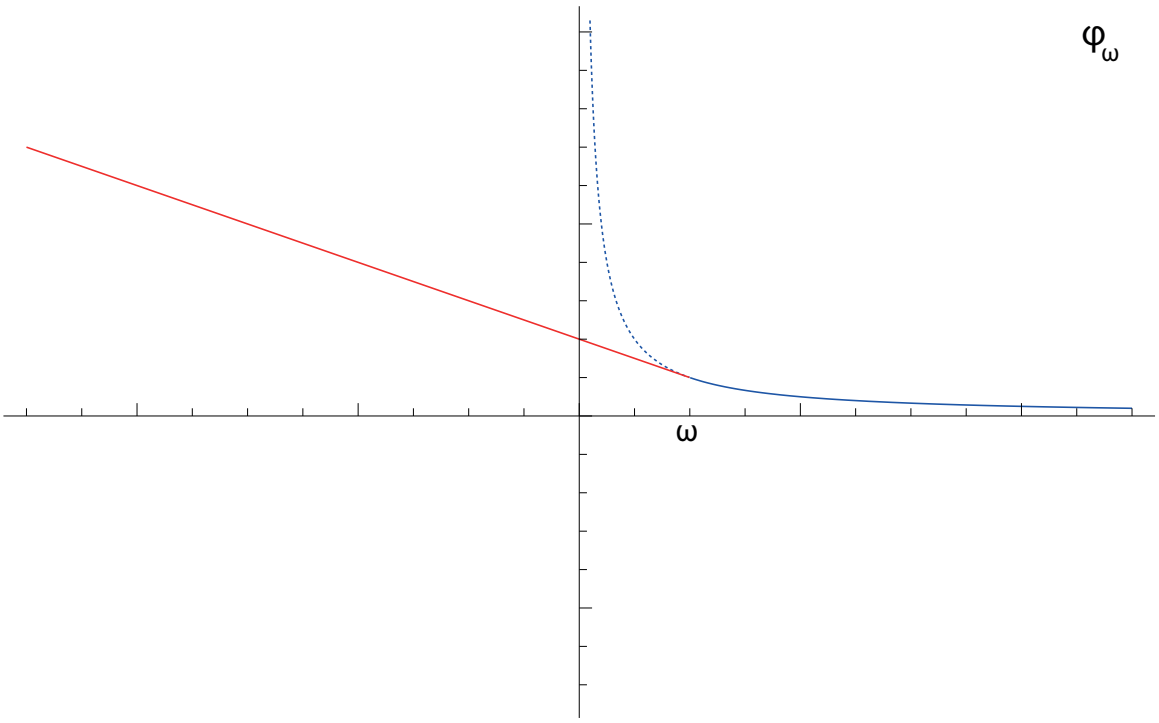


Figura 1.5: función φ_ω

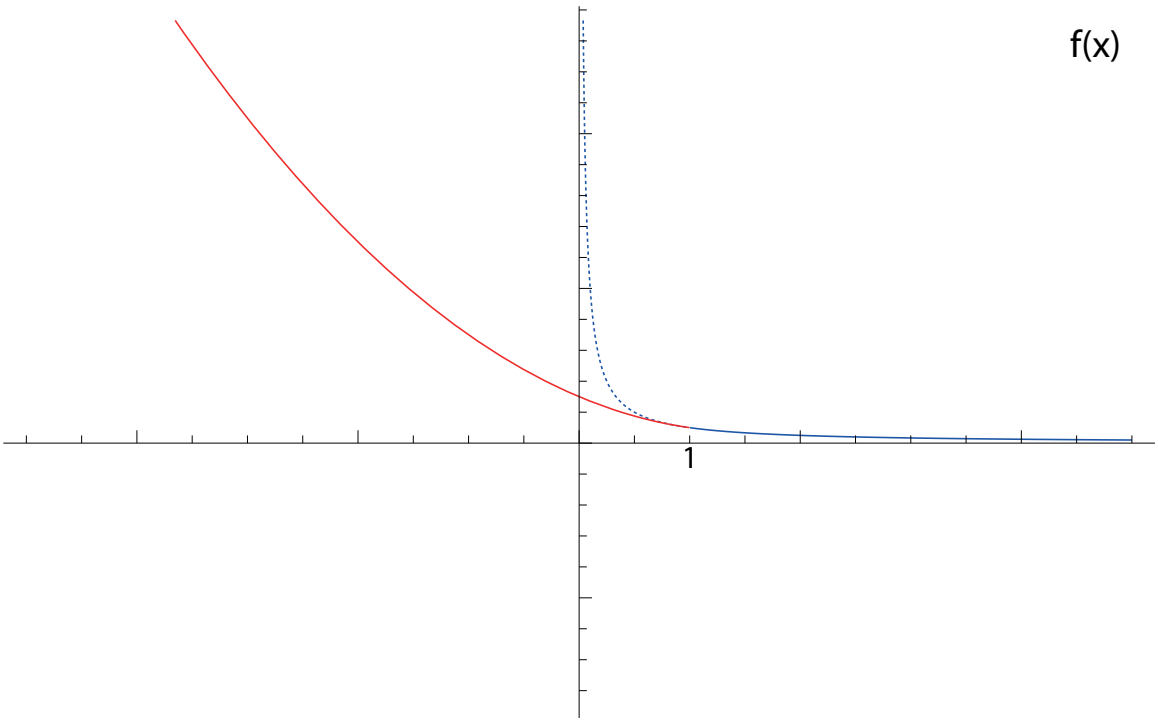


Figura 1.6: función f

1.3.3. Algoritmos

El algoritmo para resolver el problema de optimización con los funcionales de área y suavidad es muy semejante por lo que sólo se explicará para el caso en que el funcional es H_ω y se hará mención de los cambios que se deben hacer para $S_{\omega,\epsilon}$. Algo importante de mencionar es que la solución de estos funcionales crea de manera explícita una homotopía discreta entre la malla inicial y la final con lo que paso a paso se van calculando mallas que forman esta homotopía.

Definir $tolf, tolg$, $\omega, \tau \geq 1$, $\epsilon > 0$todos tipo flotante, $i, ITERMAX$ enteros $i = 0$;
Generar la malla inicial G;
while *G no es ϵ-convexa* $\wedge$ $i < ITERMAX$ **do**
    Resolver el problema: $G = \arg\min_{G \in M(G)} H_\omega(G)$;
    $\omega = \tau\omega$;
end

Capítulo 2

Generación de Mallas ϵ-convexas en regiones con agujeros

Hasta el momento hemos hablado de como se realiza la generación de mallas convexas para regiones simplemente conexas; sin embargo, el objetivo de este trabajo es presentar una extensión del dominio de regiones de estudio añadiendo las regiones con agujeros, la idea de este tratamiento sobre las regiones es hacerlo de una forma sencilla y eficiente, ya que este problema se puede atacar de diferentes maneras en dependencia de la cantidad de agujeros o de las propiedades que se quieren sobre la malla, en términos generales muchos de los algoritmos se basan en la misma idea, separar la región inicial en regiones sin agujeros que pueda ser resueltas; sin embargo, son pocos los casos en que se tiene un orden sencillo de la región para trabajar, que finalmente se deberán usar para resolver ecuaciones, por lo que una estructura simple es lo que se propone en este trabajo.

2.1. Antecedentes

El problema de manejar regiones con agujeros ya ha sido tratado desde diferentes maneras [16], antes de dar el que se utiliza en este trabajo, haré una mención de algunos enfoques.

Un enfoque muy sencillo es el que se muestra en la figura 2.1 donde existen puntos repetidos en el contorno a modo de relación de equivalencia; estos puntos repetidos son generalmente de fronteras opuestas aunque pueden pertenecer a la misma frontera , la matriz que se maneja en este caso debe ser manejada con la relación de equivalencia que le corresponda, sin embargo en regiones con muchos agujeros la relación de equivalencia se puede volver complicada, y la distribución de la malla no siempre es adecuada ya que se debe hacer con cuidado.

Existen varios métodos que tienen por objetivo dividir geométricamente la región en algunas subregiones contiguas a las cuales se les llama bloques; donde cada bloque se puede generar una malla con algún método ya conocido. A la malla resultante de la unión de todos los bloques se le llama **malla multi-bloque** (Fig. 2.2); sin embargo, esta forma de dividir tiene muchas vertientes.

También se puede dividir en diferentes bloques, en donde cada bloque representa una región sin agujeros y son unidos de tal forma que se posible identificar a que región corresponde (Fig. 2.2b). Esta técnica es realmente complicada de manejar, ya que depende de la región la configuración geomética de los bloques que la dividen. Uno de estos consiste en dividir al contorno en dos partes

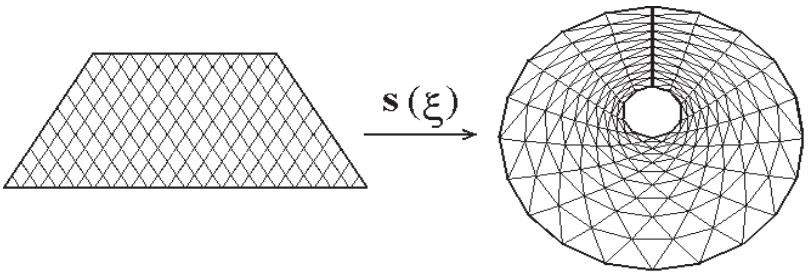


Figura 2.1

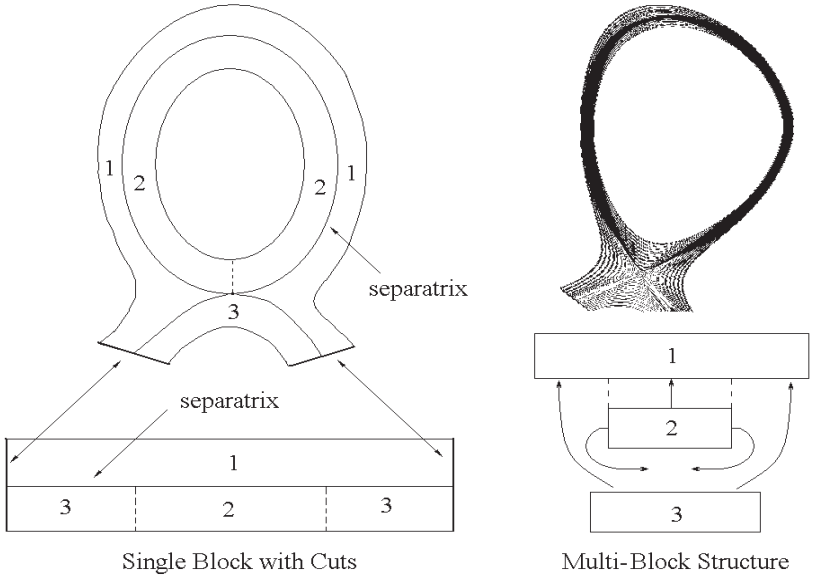


Figura 2.2: Tipos de Mallas por Bloques

de tal forma que cada parte sea un contorno sin agujero, para luego generar la malla a cada parte y finalmente unirla, esta estructura en la parte de la unión tiene puntos en común (la frontera con la que se dividió) y puntos que no; lo que ocasiona una separación dentro de la malla (Fig. 2.3); donde se debe trabajar con cuidado de preservar la relación de equivalencia en los puntos correctos.

Otra manera de dividir la región es auxiliándose de una curva llamada "Eje Medial" y con esta inducir una partición dentro de la región(Fig. 2.4); sin embargo encontrar el "Eje Medial" puede ser complicado para regiones irregulares, por lo que estos métodos cubren regiones con fronteras muy simples [13].

Las llamadas Mallas de Bloque-Estructuradas permiten usar una submalla para representar los agujeros que existan en la región (Fig. 2.5), estas se manejan de una manera muy semejante a las mallas estructuradas y son compatibles de una forma eficiente con diferencias finitas, volumen finito, elemento finito, o elemento finito espectral [16], también este tipo de método es usado para fijar subregiones [14], lo que muestra una relación intima entre fijar regiones y los agujeros. Para

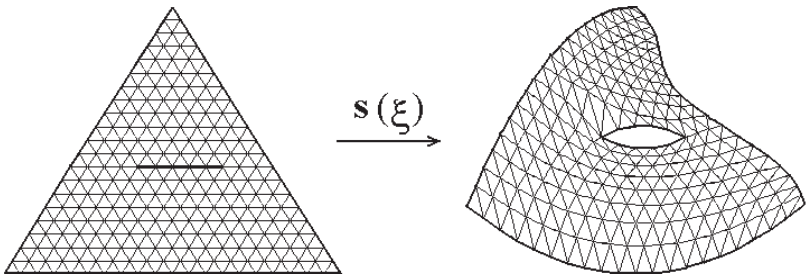


Figura 2.3

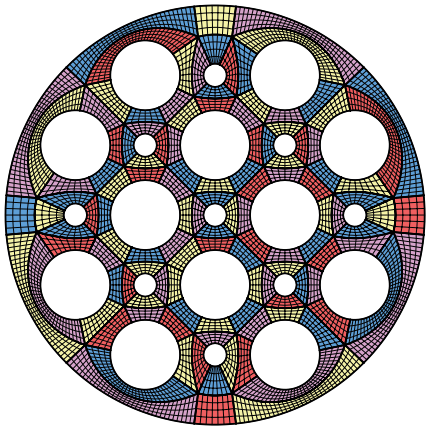


Figura 2.4: Malla MultiBloque

determinar la submalla que se asocia al agujero usualmente se divide la región de tal suerte que cada bloque vaya a una submalla. Existen muchas formas de realizar esta división y hasta la fecha no hay una forma automática de hacerla. Una manera de hacerlo es distribuyendo las submallas que van a los agujeros lo más semejante posible a la distribución de la región (Fig. 2.5), otra manera sería distribuir las submallas de forma alineada (Fig. 2.6) esto permite tener una distribución de los bloques más sencilla.

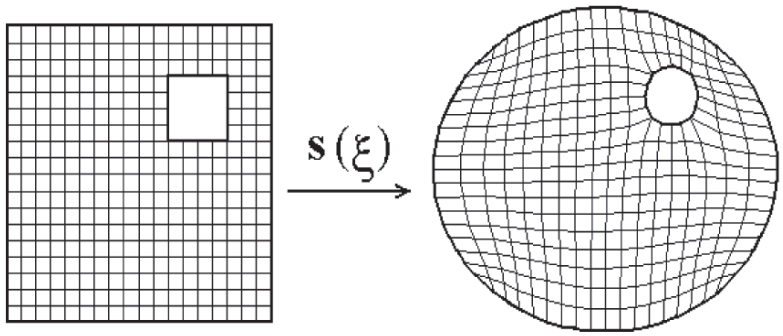


Figura 2.5: Malla Bloque-Estructurada

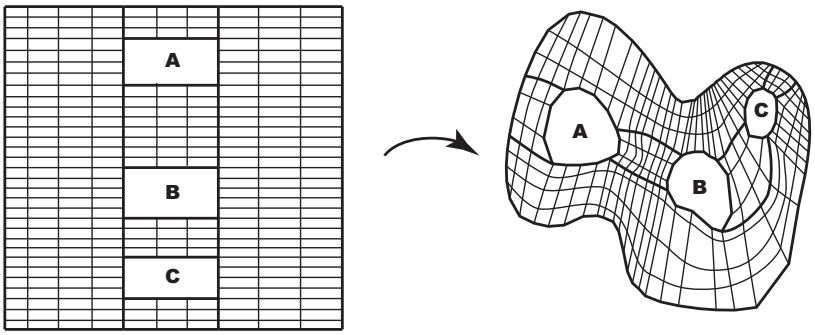


Figura 2.6: Malla Bloque-Estructurada Alineada

2.2. Método para generación de mallas ϵ-convexas

2.2.1. Planteamiento del Problema

Definición 2.1. *Regiones Admisibles*
Llamaremos "Región plana con ρ-agujeros admisible", a una región[1] $\Omega \subset \mathbb{R}^2$ *que cumple lo siguiente:*

- *Existe una región poligonal simple $\Omega_{ext} \subset \mathbb{R}^2$, la cuál llamaremos "Contorno Exterior".*
- *Existe un conjunto con cardinalidad ρ, compuesto por regiones poligonales simplemente conexas Ω_i con $i \in \{1, \ldots, \rho\}$ a las cuales llamaremos agujeros.*

tales que

- *Los agujeros son ajenos:*
  $\Omega_i \cap \Omega_j = \emptyset \quad \forall i \neq j$
- *Los agujeros están contenidos en el contorno exterior:*
  $\Omega_i \subset \Omega_{ext} \quad i \in \{1, \ldots, \rho\}$
- $\Omega = \Omega_{ext} - (\cup_{i=1}^{h} \Omega_i^{\circ})$

Como se puede ver en la figura 2.18.

Definición 2.2. *Cuadrado con ρ agujeros*
Llamaremos "Cuadrado con ρ agujeros", denotado como $R_{\rho H}$, a una región plana que cumple con lo siguiente:

- *Existe un conjunto de cardinalidad ρ, formado por regiones plana R_i con las siguientes propiedades:*
  - $R_i = [a_i, b_i] \times [c_i, d_i]$
    donde $a_i, b_i, c_i, d_i \in (0, 1)$
  - *Los R_i son ajenos:*
    $R_i \cap R_j = \emptyset \quad \forall i \neq j$
- $$R_{\rho H} = R - (\cup_{i=1}^{\rho} R_i^{\circ})$$

  donde R es el cuadrado unitario.

Como se ve en la figura 2.7.

[1] Con región nos referimos a un conjunto conexo con interior no vacío

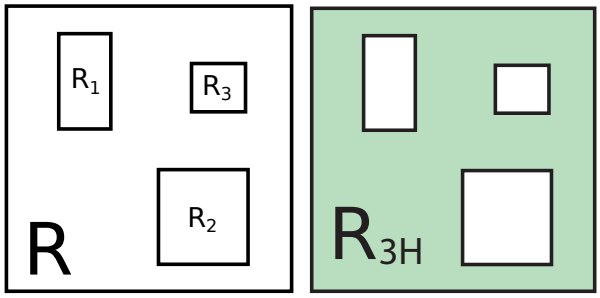


Figura 2.7: Ejemplo de un Cuadrado con 3 agujeros

Definición 2.3. *Malla Estructurada*
Dada una región plana con ρ-agujeros admisible Ω, le llamaremos "Malla estructurada sobre Ω" a un homeomorfismo $\varphi : R_{\rho H} \to \Omega$

Hasta la fecha no conocemos algún métodos general para construir mallas estructuradas sobre regiones con agujeros. Una posibilidad sería partir de los mapeos:

$$\varphi_i : R_i \to \Omega_i \quad \forall i \in \{1, \ldots, \rho\}$$

tales que

$$\varphi_i(\partial R_i) = \partial \Omega_i$$

Y buscar un mapeo homeomorfo $\varphi : R \to \Omega_{ext}$ tal que

$$\varphi|_{R_i} = \varphi_i \quad \forall i \in \{1, \ldots, n\}$$
$$\varphi|_{\partial R} = \partial \Omega_{ext}$$

Esta formulación puede no tener solución. Por lo que proponemos una solución de manera análoga a los métodos para generar mallas estructuradas en regiones poligonales simples; abordaremos el problema de generar mallas estructuradas sobre regiones planas con n-agujeros admisible de la siguiente manera:

Dados los mapeos homeomorfos

$$\tilde{\varphi}_i : \partial R_i \to \partial \Omega_i$$
$$\tilde{\varphi} : \partial R \to \partial \Omega_{ext}$$

Construiremos un homemorfismo $\varphi : R_{\rho H} \to \Omega$ (Fig. 2.8) tal que:

$$\varphi|_{\partial R_i} = \tilde{\varphi}_i \quad \forall i \in \{1, \ldots, \rho\}$$
$$\varphi|_{\partial R} = \tilde{\varphi}$$

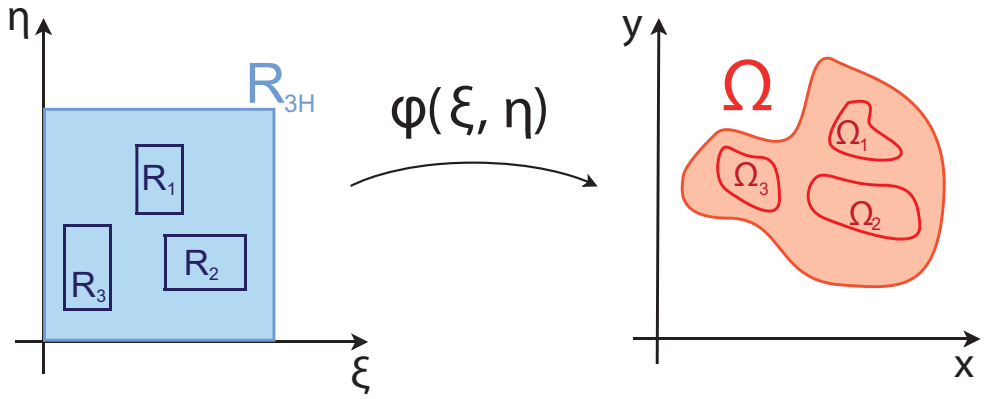


Figura 2.8: Mapeo φ

2.2.2. Esbozo del método propuesto

La idea del método es partir la región con agujeros en dos regiones Θ_1 y Θ_2 cada una sin agujeros por medio de unas poligonales auxiliares, para generar mallas estructuradas sobre Θ_1 y Θ_2 que sea compatibles para con ellas construir una malla estructurada sobre Ω. Tomando Θ_1 y Θ_2 regiones poligonales simplemente conexas, para construir las mallas sobre estas necesitamos especificar la distribución de los puntos sobre las fronteras donde se encuentran los agujeros para conservar la forma. Una vez generadas las mallas en cada región se puede optar por hacer convexas cada parte y unirlas, o primero unirlas y posteriormente hacer la malla convexa; nosotros hemos optado por esta última, ya que sólo es necesario un proceso de optimización.

Se presentará este nuevo método a detalle, en dos ejemplos: con un agujero y con dos agujeros. Con el fin de presentar las ideas de manera sencilla [2] y posteriormente explicar el método general.

Ejemplo: Un Agujero

Consideremos la región Ω con un agujero Ω_1, por ejemplo la que aparece en la figura 2.10. Como el agujero es una región simplemente conexa entonces existe un homeomorfismo $\varphi_1 : R_1 \to \Omega_1$ tal que $\varphi_1(\partial R_1) = \partial\Omega_1$, donde R_1 es cualquier rectángulo, nuestra idea es elegir a $R_1 \subset R$ (R es el cuadrado unitario) y buscar un mapeo $\varphi : R \to \Omega \cup \Omega_1$ homeomorfismo una extensión de φ_1 tal que:

$$\varphi(\partial R) = \partial\Omega$$
$$\varphi(\partial R_1) = \partial\Omega_1$$

En general puede ser posible que no existe el homeomorfimos φ, pero se presentará un método que nos permitirá construir un mapeo homeomorfo $\varphi : R - R_1^\circ \to \Omega - \Omega_1^\circ$, no intentaremos extender a un homeomorfimo que incluya los agujeros.

[2] El esbozo pretende dar la idea del método más que representar la implementación del mismo, los detalles de la implementación se discutirán más adelante

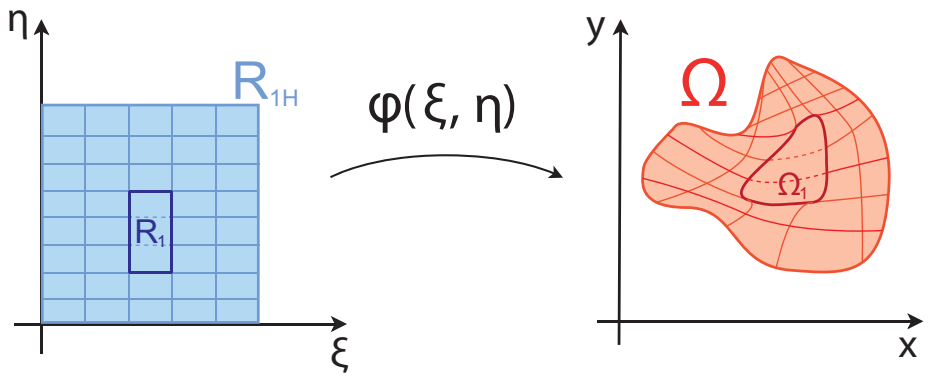


Figura 2.9: Homeomorfismo

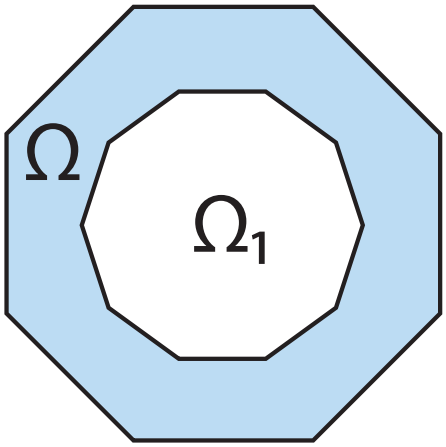


Figura 2.10: Región con un agujero

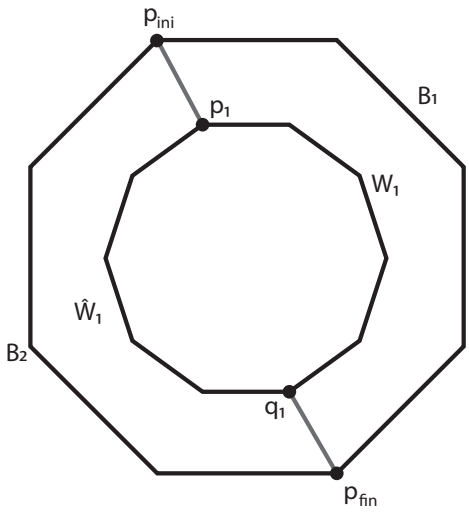


Figura 2.11: Región con un agujero divida

Paso 1:
Elijase dos puntos p_{ini}, p_{fin} sobre el contorno exterior ($\Omega_{ext} = \Omega \cup \Omega_1$) con los cuales la frontera del contorno exterior queda dividida en dos curvas B_1, B_2 donde:

$$B_1 \cup B_2 = \partial\Omega_{ext}$$
$$B_1 \cap B_2 = \{p_{ini}, p_{fin}\}$$

Elijanse dos puntos p_1, q_1 sobre Ω_1, dichos puntos dividen la frontera en dos curvas $\hat{W}_1, W_1$ tales que:

$$\hat{W}_1 \cup W_1 = \partial\Omega_1$$
$$\hat{W}_1 \cap W_1 = \{p_1, q_1\}$$

Como se muestra en la figura 2.11, consideremos la curva $\Gamma := \overline{p_{ini}p_1} \cup \overline{q_1p_2} \cup \overline{q_3p_{fin}}$, y nombraremos las curvas:

$$\Gamma_1 := \overline{p_{ini}p_1} \cup W_1 \cup \overline{q_1p_{fin}}$$
$$\Gamma_2 := \overline{p_{ini}p_1} \cup \hat{W}_1 \cup \overline{q_1p_{fin}}$$

Estas dividen a la región Ω en dos subregiones Θ_1, Θ_2 simplemente conexas que tienen a Γ en común, es decir:

$$\Theta_1 \cup \Theta_2 = \Omega$$
$$\Theta_1 \cap \Theta_2 = \Gamma$$
$$\partial\Theta_1 = \Gamma_1 \cup B_1$$
$$\partial\Theta_2 = \Gamma_2 \cup B_2$$

Paso 2:
Se elijen dos puntos d_1, d_2 sobre B_1 y dos puntos e_1, e_2 sobre B_2; los puntos $d_1, p_{ini}, p_{fin}, d_2$ determinan las cuatro fronteras para construir una malla sobre Θ_1, de la misma forma $e_1, e_2, p_{fin}, p_{ini}$ para Θ_2. (Fig. 2.12)

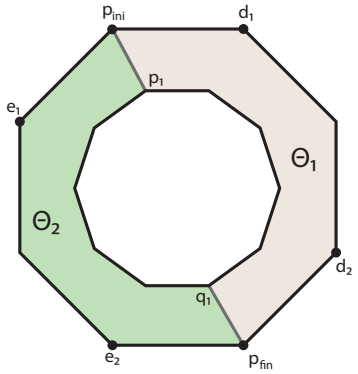


Figura 2.12: Región con un agujero dividida

Paso 3:
Elijanse las dimensiones de las mallas; es decir, elija el número de puntos sobre cada curva. Hay que recordar que necesitamos especificar la distribución de los puntos para no perder la forma y sean compatibles las mallas que generemos) Sean $\eta_A, \eta_B, \eta_1, \eta_2, \mu_1 \in \mathbb{N}$ mayores que tres, asignaremos la cantidad de puntos en cada curva de la siguiente manera:

- $e_1 p_{ini} \to \eta_A$
- $e_2 p_{fin} \to \eta_A$
- $p_{ini} d_1 \to \eta_B$
- $p_{fin} d_2 \to \eta_B$
- $\overline{p_{ini} p_1} \to \eta_1$
- $W_1 \to \mu_1$
- $\hat{W}_1 \to \mu_1$
- $\overline{q_1 p_{fin}} \to \eta_2$

Paso 4:

Se genera una malla estructurada[3] G_1 sobre Θ_1 de dimensiones:

$$\eta_B \times (\eta_1 + \mu_1 + \eta_2)$$

[3]Se puede generar con diversos métodos, véase [12] y `http://www.matematicas.unam.mx/unamalla/`

Se genera una malla estructurada G_2 sobre Θ_2 de dimensiones:

$$\eta_A \times (\eta_1 + \mu_1 + \eta_2) \tag{2.1}$$

Con G_1 y G_2 se construye una malla G sobre Ω de dimensión:

$$(\eta_A + \eta_B - 1) \times (\eta_1 + \mu_1 + \eta_2) \tag{2.2}$$

Paso 5:
Con la malla G es posible construir una malla suave y convexa utilizando los funcionales ya conocidos.
La figura 2.13 muestra un mismo contorno con dos configuraciones diferentes

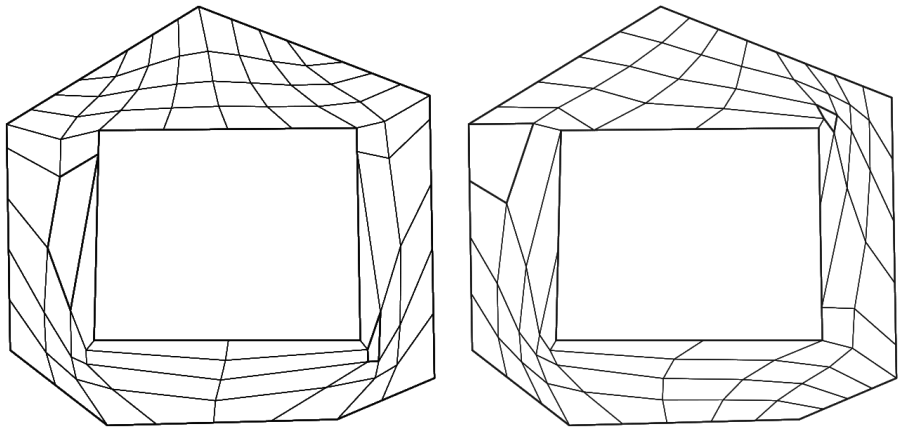


Figura 2.13: Mallas con un agujero

Ejemplo: Dos Agujero

Consideremos ahora una región Ω con dos agujeros (Ω_1 y Ω_2), como se ve en la figura 2.15. De manera similar, existen homeomorfismos como sigue:

$$\varphi_i : R_i \to \Omega_i \; i \in \{1, 2\}$$
$$\varphi_i(\partial R_i) = \partial \Omega_i$$

Donde R_1, R_2 son rectángulos. La idea es elegir R_1 y R_2 como subrectángulos del cuadrado unitario R y extender cada homeomorfismo a un homeomorfismo $\varphi : R \to \Omega \cup \Omega_1 \cup \Omega_2$ con las siguientes propiedades:

$$\varphi(\partial R) = \partial(\Omega \cup \Omega_1 \cup \Omega_2)$$
$$\varphi(R_i) = \varphi_i \; i \in \{1, 2\}$$

Se presentará un método que nos permitirá construir un mapeo $\varphi : R - R_1^\circ - R_2^\circ \to \Omega - \Omega_1^\circ - \Omega_2^\circ$ homeomorfismo,tal que:

$$\varphi(\partial R_1) = \partial\Omega_1$$
$$\varphi(\partial R_2) = \partial\Omega_2$$
$$\varphi(\partial R) = \partial(\Omega \cup \Omega_1 \cup \Omega_2)$$

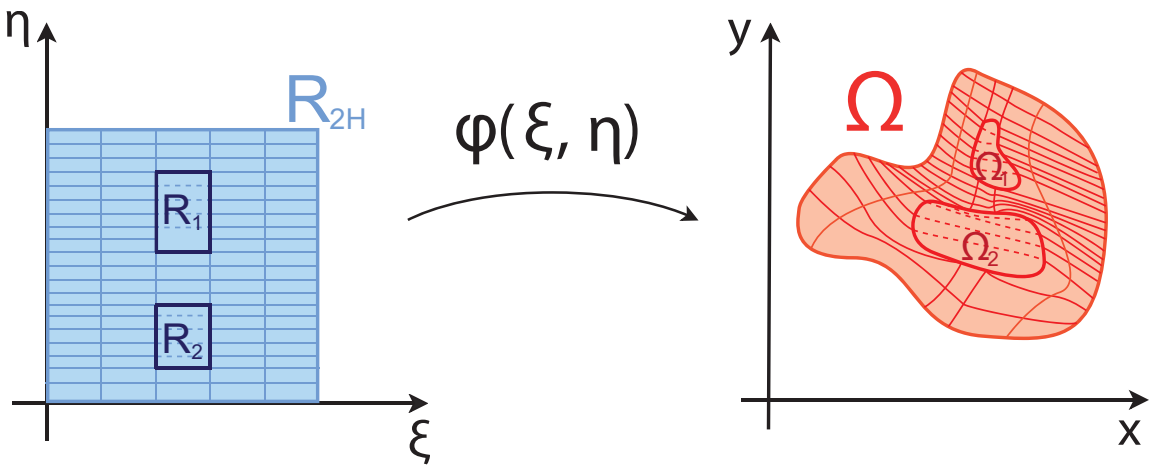


Figura 2.14: Mapeo φ

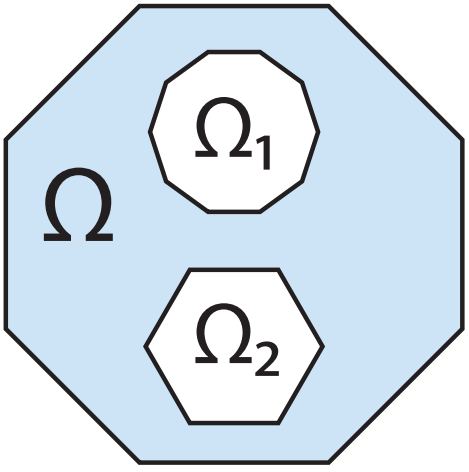


Figura 2.15: Región con dos agujeros

Paso 1:
dividida la región Ω en dos regiones Θ_1 y Θ_2 tales que $\Omega = \Theta_1 \cup \Theta_2$, por medio de la curva Γ formada por una elección de puntos $p_{ini}, p_{fin} \in \partial\Omega$, $p_1, q_1 \in \Omega_1$, $p_2, q_2 \in \Omega_2$ donde $\Gamma := \overline{p_{ini}p_1} \cup \overline{q_1p_2} \cup \overline{q_3p_{fin}}$, y que índicen una partición en la frontera de cada frontera de la siguiente forma:

$$B_1 \cup B_2 = \partial\Omega$$
$$B_1 \cap B_1 = \{p_{ini}, p_{fin}\}$$
$$\hat{W}_1 \cup W_1 = \partial\Omega_1$$
$$\hat{W}_1 \cap W_1 = \{p_1, q_1\}$$
$$\hat{W}_2 \cup W_2 = \partial\Omega_2$$
$$\hat{W}_2 \cap W_2 = \{p_2, q_2\}$$

Tomamos las siguientes curvas auxiliares:

$$\Gamma_1 := \overline{p_{ini}p_1} \cup W_1 \cup \overline{q_1p_2} \cup W_2 \cup \overline{q_2p_{fin}}$$
$$\Gamma_2 := \overline{p_{ini}p_1} \cup \hat{W}_1 \cup \overline{q_1p_2} \cup \hat{W}_2 \cup \overline{q_2p_{fin}}$$

Donde $\partial\Theta_1 := \Gamma_1 \cup B_1$ y $\partial\Theta_2 := \Gamma_2 \cup B_2$ por lo tanto $\Theta_1 \cap \Theta_2 = \Gamma$. (Fig. 2.16)

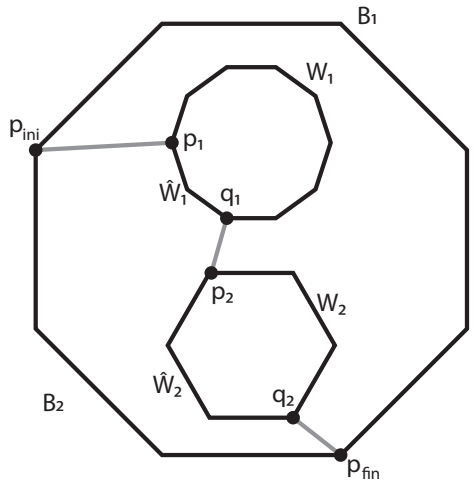


Figura 2.16: Región con dos agujeros dividida

Paso 2:
Se elijen dos puntos d_1, d_2 sobre B_1 y dos puntos e_1, e_2 sobre B_2; los puntos $d_1, p_{ini}, p_{fin}, d_2$ determinan las cuatro fronteras para construir una malla sobre Θ_1, de la misma forma $e_1, e_2, p_{fin}, p_{ini}$ para Θ_2. Como se puede observar en la figura 2.17.

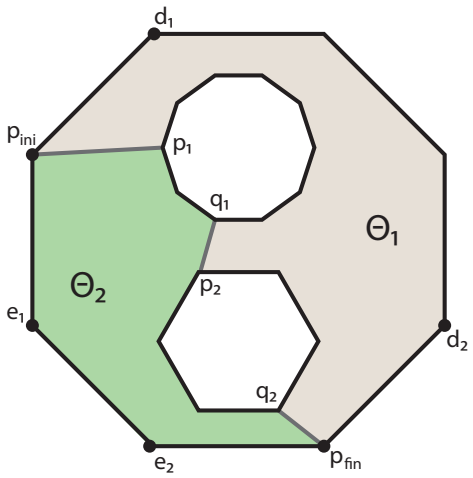


Figura 2.17: Región con dos agujeros separa en dos regiones

Paso 3:
Elijanse las dimensiones de las mallas; es decir, elija el número de puntos sobre las curvas. Sean $\eta_A, \eta_B, \eta_1, \eta_2, \eta_3, \mu_1, \mu_2 \in \mathbb{N}$ mayores que tres, asignaremos la cantidad de puntos en cada curva de la siguiente manera:

- $e_1 p_{ini} \to \eta_A$
- $e_2 p_{fin} \to \eta_A$
- $p_{ini} d_1 \to \eta_B$
- $p_{fin} d_2 \to \eta_B$
- $\overline{p_{ini} p_1} \to \eta_1$
- $W_1 \to \mu_1$
- $\hat{W}_1 \to \mu_1$
- $\overline{q_1 p_2} \to \eta_2$
- $W_2 \to \mu_2$
- $\hat{W}_2 \to \mu_2$
- $\overline{q_2 p_{fin}} \to \eta_3$

Paso 4:

Se genera una malla estructurada G_1 sobre Θ_1 de dimensiones:

$$\eta_B \times (\eta_1 + \mu_1 + \eta_2 + \mu_2 + \eta_3)$$

Se genera una malla estructurada G_2 sobre Θ_2 de dimensiones:

$$\eta_A \times (\eta_1 + \mu_1 + \eta_2 + \mu_2 + \eta_3) \tag{2.3}$$

Con G_1 y G_2 se construye una malla G sobre Ω de dimensión:

$$(\eta_A + \eta_B - 1) \times (\eta_1 + \mu_1 + \eta_2 + \mu_2 + \eta_3) \tag{2.4}$$

Paso 5:
Con la malla G es posible construir una malla suave y convexa utilizando los funcionales ya conocidos. Las figuras que se encuentran en el anexo exponen diversos ejemplos que muestran diversas configuraciones para una misma región.

Caso General (ρ-agujeros)

En el caso general la figura sería parecida (en estructura) a la figura 2.18. La región Ω tiene ρ agujeros $(\Omega_1, \Omega_2, \ldots, \Omega_\rho)$.

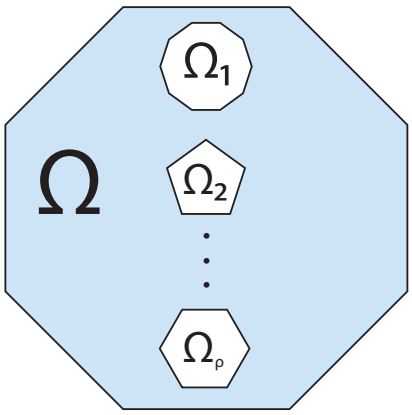


Figura 2.18: Región con n-agujeros

Paso 1:
dividida la región Ω en dos regiones Θ_1 y Θ_2 tales que $\Omega = \Theta_1 \cup \Theta_2$, auxiliandonos de la curva Γ formada por una elección de puntos $p_{ini}, p_{fin} \in \partial\Omega$, $p_1, q_1 \in \Omega_1$, $p_2, q_2 \in \Omega_2$, ... , $p_\rho, q_\rho \in \Omega_\rho$ donde $\Gamma := \overline{p_{ini}p_1} \cup (\cup_{i=1}^{n-1} \overline{q_i p_{i+1}}) \cup \overline{q_n p_{fin}}$, y que índicen una partición en la frontera de cada frontera de

la siguiente forma:

$$B_1 \cup B_2 = \partial\Omega$$
$$B_1 \cap B_1 = \{p_{ini}, p_{fin}\}$$
$$\hat{W}_i \cup W_i = \partial\Omega_i$$
$$\hat{W}_i \cap W_i = \{p_i, q_i\}$$
$$\forall i \in \{1, 2, \ldots, \rho\}$$

Tomamos las siguientes curvas auxiliares:

$$\Gamma_1 := \Gamma \cup (\cup_{i=1}^{\rho} W_i)$$
$$\Gamma_2 := \Gamma \cup (\cup_{i=1}^{\rho} \hat{W}_i)$$

Donde $\partial\Theta_1 := \Gamma_1 \cup B_1$ y $\partial\Theta_2 := \Gamma_2 \cup B_2$ por lo tanto $\Theta_1 \cap \Theta_2 = \Gamma$. Como se ve en la figura 2.19

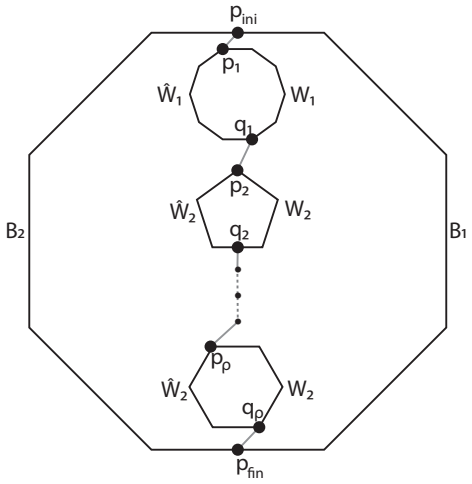


Figura 2.19: Región con n agujeros dividida

Paso 2:
Se elijen dos puntos d_1, d_2 sobre B_1 y dos puntos e_1, e_2 sobre B_2; los puntos $d_1, p_{ini}, p_{fin}, d_2$ determinan las cuatro fronteras para construir una malla sobre Θ_1, de la misma forma $e_1, e_2, p_{fin}, p_{ini}$ para Θ_2. (Fig. 2.20)

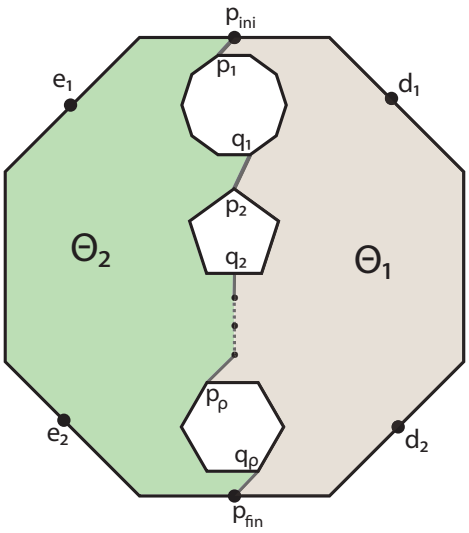


Figura 2.20: Región con n agujeros separa en dos regiones

Paso 3:
Elijanse las dimensiones de las mallas; es decir, elija el número de puntos sobre las curvas. Sean $\eta_A, \eta_B, \eta_i, \mu_j \in \mathbb{N}\ \forall i \in \{1, 2, \ldots, \rho, \rho+1\}\ j \in \{1, 2, \ldots, \rho\}$ mayores que tres, asignaremos la cantidad de puntos en cada curva de la siguiente manera:

Donde se deben asignar los puntos sobre cada curva como sigue:

- $e_1 p_{ini} \to \eta_A$
- $e_2 p_{fin} \to \eta_A$
- $p_{ini} d_1 \to \eta_B$
- $p_{fin} d_2 \to \eta_B$
- $\overline{p_{ini} p_1} \to \eta_1$
- $W_1 \to \mu_1$
- $\hat{W}_1 \to \mu_1$
- $\overline{q_i p_{i+1}} \to \eta_{i+1}\ \forall i \in \{1, \ldots, \rho - 1\}$
- $W_i \to \mu_i\ \forall i \in \{1, \ldots, \rho\}$
- $\hat{W}_i \to \mu_i\ \forall i \in \{1, \ldots, \rho\}$
- $\overline{q_\rho p_{fin}} \to \eta_{\rho+1}$

Paso 4:

Se genera una malla estructurada G_1 sobre Θ_1 de dimensiones:

$$\eta_B \times (\sum_{i=1}^{\rho+1} \eta_i + \sum_{i=1}^{\rho} \mu_i)$$

Se genera una malla estructurada G_2 sobre Θ_2 de dimensiones:

$$\eta_A \times (\sum_{i=1}^{\rho+1} \eta_i + \sum_{i=1}^{\rho} \mu_i) \tag{2.5}$$

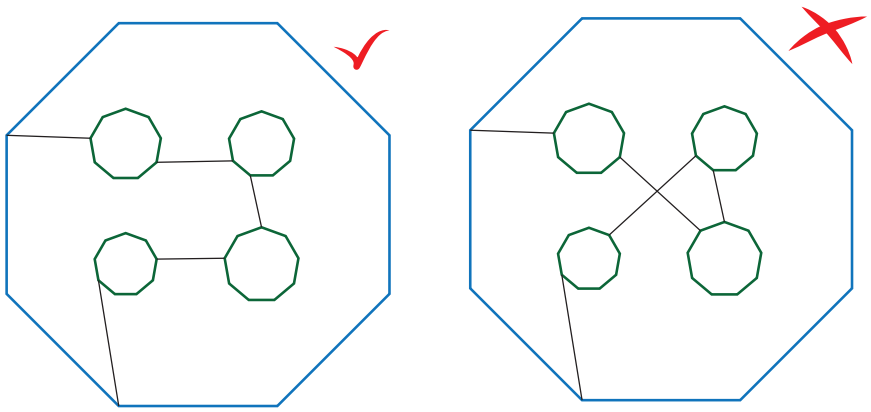
Con G_1 y G_2 se construye una malla G sobre Ω de dimensión:

$$(\eta_A + \eta_B - 1) \times (\sum_{i=1}^{\rho+1} \eta_i + \sum_{i=1}^{\rho} \mu_i) \tag{2.6}$$

Paso 5:
Con la malla G es posible construir una malla suave y convexa utilizando los funcionales ya conocidos.

2.2.3. Algoritmo

Como ya se vio antes, la idea del algoritmo en general es dividir la región Ω en dos regiones conexas y sin agujeros (Θ_1 y Θ_2), por medio de dos curvas (Γ_1 y Γ_2); de esta forma obtenemos dos regiones ajenas conexas sin agujeros (Fig. 2.17) y a las cuales les podemos aplicar los métodos ya conocidos para generar una malla inicial (para la malla inicial no es necesario que la región sea simplemente conexa, al menos no para el método TFI), para luego unirlas y entonces utilizar la optimización usual; haciendo fijos los puntos de la frontera de los agujeros; por lo que generar una malla inicial convexa no es importante. Como los puntos fijos son sólo los puntos sobre las fronteras de cada subcontorno, no es importante el segmento $\overline{p_{ini}p_1}$ intersecte o no la región Ω_1 (Fig. 2.21a),análogamente que $\overline{q_n p_{fin}}$ intercepte a Ω_ρ ni los segmentos $\overline{q_i p_{i+1}}$ tenga intersección con $\Omega_{i-1} \cup \Omega_i$ donde $i \in \{1, \ldots, \rho-1\}$, aunque es necesario que no haya intersección entre los segmentos: $\overline{p_{ini}p_1} \cap \overline{q_\rho p_{fin}} \cap (\cup_{i=1}^{\rho-1} \overline{q_i p_{i+1}}) = \emptyset$(Fig. 2.21b). Esto nos quita la necesidad de buscar un camino que dividida a Ω en dos regiones simplemente conexas, para luego generar las mallas iniciales en cada región (Θ_1 y Θ_2), fijar los puntos de las fronteras internas, y buscando que todas las celdas sean convexas se encuentra un camino simple de forma implícita dentro de la optimización.



(a) Recomendado (b) No recomendado

Figura 2.21: Lineas auxiliares

También se mencionó que $\partial\Theta_1 \cap \partial\Theta_2 = \Gamma$ por lo que las mallas generadas para Θ_1 y Θ_2 comparten puntos, lo que complica la forma de manejar la malla completa sobre Ω; esto se resuelve tomando un desplazamiento uniforme de los puntos $p_{ini}, p_{fin}, p_i, q_i$ sobre la frontera donde se encuentra cada punto para generar una nueva Γ_1 y Γ_2 nuevas fronteras para Θ_1 y Θ_2 (Una de estas fronteras es igual a la definida antes), que permite tomar a Θ_1 ajeno de Θ_2 (Fig. 2.22), para poder unir G_1 y G_2 de manera trivial concatenándolas (Fig. 2.23). El algoritmo decide en que dirección tomar el desplazamiento tomando el lado más largo (entre $e_1 p_{ini} \cup e_2 p_{fin}$ y $p_{ini} d_1 \cup p_{fin} d_2$).

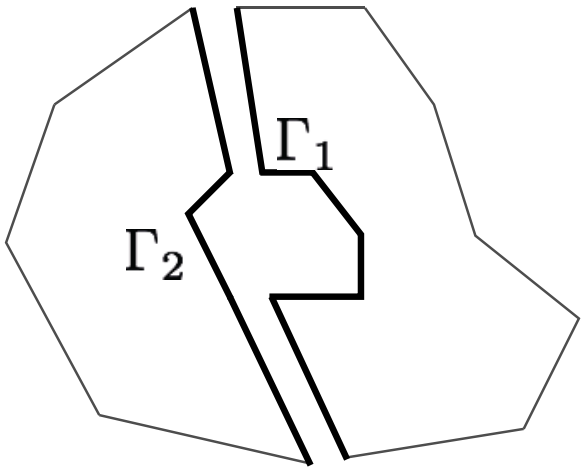
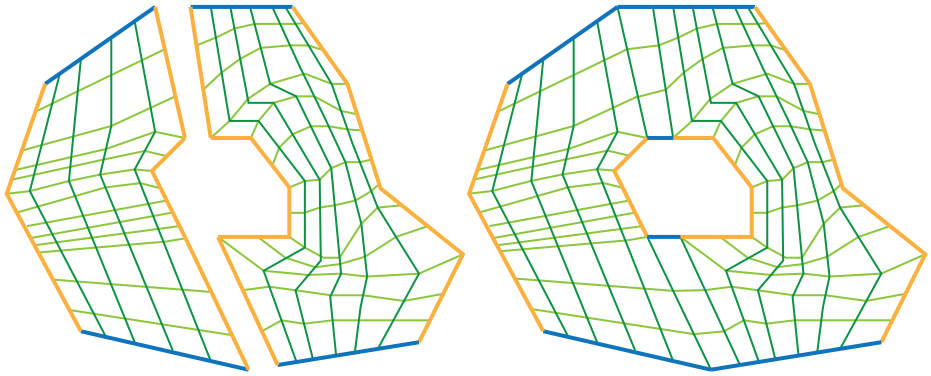


Figura 2.22: Región dividida por dos caminos



(a) Mallas G_1 y G_2 (b) Malla unida

Figura 2.23: Forma de unir las mallas G_1 y G_2

Insertar esta nueva curva Γ_1 (ó Γ_2) cambia la dimensión de la malla resultante pues al haber una nueva columna (o fila) de celdas tenemos que la dimensión de la malla resultante es:

$$(\eta_A + \eta_B) \times (\sum_{i=1}^{\rho+1} \eta_i + \sum_{i=1}^{\rho} \mu_i)$$

Este método da lugar a una nueva definición:

Definición 2.4. *Malla Estructurada Discreta Dada* Ω *una región plana con* ρ*-agujeros admisible,* $m_1, m_2 \in \mathbb{N}$ *mayores que* 3, $\eta_i \in \mathbb{N}$ *elementos de una colección de números de cardinalidad* $\rho + 1$ *tales que* $\eta_i > 3 \quad \forall i \in \{1, \ldots, \rho+1\}$, $\mu_i \in \mathbb{N}$ *elementos de una colección de naturales de cardinalidad* h *tales que todos son mayores que tres.*

Sean

$$m = m_1 + m_2$$
$$n = (\sum_{i=1}^{\rho+1} \eta_i) + (\sum_{i=1}^{\rho} \mu_i)$$

Sea V *el conjunto de vértices que definen a* Ω, $H_i : \mathbb{N} \to \mathbb{N}$ *una familia de funciones definidas como sigue:*

$$H_i(a) = (\sum_{j=1}^{i} \eta_j) + (\sum_{j=1}^{i-1} \mu_j) + (a - 1)$$

Una malla estructurada discreta admisible sobre Ω *es un conjunto de puntos* G *tales que:*

- $G = \{P_{i,j} \in \Omega | i \in \{1, 2, \ldots, m\}; j \in \{1, 2, \ldots, n\}\}$.
- $L_1(G) = \{P_{i,1} \in G | i \in \{1, 2, \ldots, m\}\}$
- $L_2(G) = \{P_{m,j} \in G | j \in \{1, 2, \ldots, n\}\}$

- $L_3(G) = \{P_{i,n} \in G | i \in \{1, 2, \ldots, m\}\}$
- $L_4(G) = \{P_{1,j} \in G | j \in \{1, 2, \ldots, n\}\}$
- $L_{iL}(G) = \{P_{m_1,j} \in G | j \in \{H_i(1), H_i(2), \ldots, H_i(\mu_i)\}\}$
- $L_{iR}(G) = \{P_{m_1+1,j} \in G | j \in \{H_i(1), H_i(2), \ldots, H_i(\mu_i)\}\}$

Los cuales cumplen:

$$\bigcup_{i=1}^{4} L_i(G) \subset \partial\Omega_{ext} \ y \ (L_{iL} \cup L_{iR}) \subset \partial\Omega_i \ \forall i \in \{1, \ldots, \rho\} \ y$$

$$V \subset (\bigcup_{i=1}^{4} L_i(G)) \cup (\bigcup_{i=1}^{\rho} L_{iL} \cup L_{iR}) \ y \ G - (\bigcup_{i=1}^{4} L_i(G)) \cup (\bigcup_{i=1}^{\rho} L_{iL} \cup L_{iR}) \subset \Omega^{\circ}$$

Es importante observar que esta definición restringe la distribución de los rectángulos R_i sobre el cuadrado unitario que definen el dominio del mapeo φ; ya que pide que dichos R_i estén alineados de forma vertical. Esta no es la única distribución pero es sencilla de describir a comparación del caso general.

2.2.4. Existencia de la curva Γ simple

En este método utiliza la existencia de un camino Γ que debe tener las siguientes características:

- Empezar y terminar en un punto dado (distinto) sobre el polígono Ω_{ext}
- Debe pasar por dos puntos dados de manera consecutiva sobre cada poligonal $\Omega_i \ i \in \{1, \ldots, \rho\}$
- No debe autointersectarse

Pero la existencia de esta poligonal no es inmediata, por lo que requiere de demostración. Para esta demostración vamos a hacer una pequeña abstracción; los dos puntos sobre cada polígono interno ($\Omega_i \ i \in \{1, \ldots, \rho\}$) los desconocemos, por lo que se va a colapsar todo el polígono a un punto, pues lo que nos interesa es que el camino una a cada polígono interno más no en donde lo une (eso significa que vamos a ver como un sólo punto a los dos puntos del camino que pasan por la frontera de los agujeros); primero creamos la poligonal que pasa por los agujeros, para finalmente unir esa poligonal a la frontera.

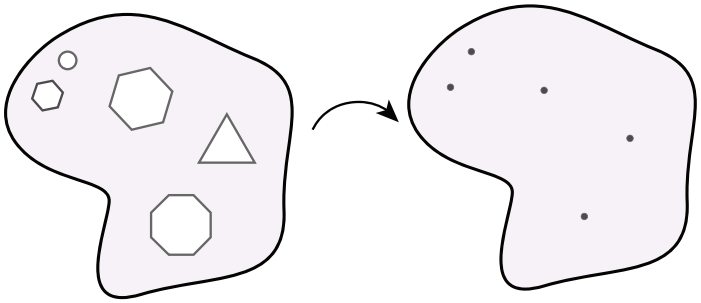


Figura 2.24: Colapso de agujeros a puntos

Lema 2.1. *Poligonal simple*
Dados k puntos en el plano, existe una poligonal que no se autointersecta y con los puntos dados como sus vértices.

Demostración. Sea $P := \{p_i | i \in \{1, \ldots, k\}\}$ el conjunto de los puntos dados.
Sea $Pol := (o_i | i \in \{1, \ldots, k\})$ un conjunto ordenado donde:
$\forall i \in \{1, \ldots, k\}$ $o_i = p_j$ con $j \in \{1, \ldots, k\}$
tal que $o_i < o_{i+1}$ $\forall i \in \{1, \ldots, k-1\}$
Usando el orden lexicográfico $(p_i < p_j \Leftrightarrow x_i < x_j \vee (x_i = x_j \wedge y_i < y_j))$ con $p_i = (x_i, y_i)$ y $p_j = (x_j, y_j)$.

Aseguro que Pol visto como poligonal es la que buscamos.

Esta poligonal no se autointersecta pues si nos paramos en un vértice o_i, o_{i+1} le pasa una de dos: esta a la derecha $(x_i < x_{i+1})$ ó esta arriba $(x_i = x_{i+1} \wedge y_i < y_{i+1})$, eso significa que avanza siempre a la derecha o arriba; si se autointersectará tendría que "regresar", cosa que no hace. □

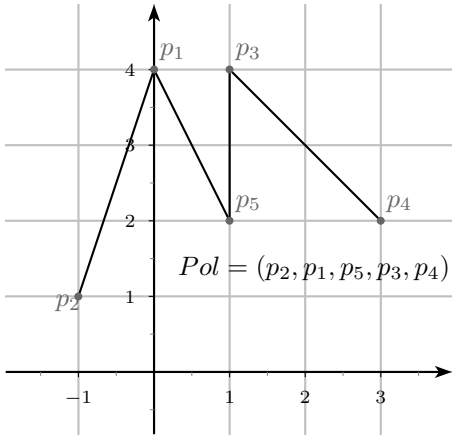


Figura 2.25: Poligonal Pol

Usando es lema anterior garantizamos la existencia de una poligonal que una los puntos que representan a los agujeros, aunque nos da restricciones de como tomar los puntos de la frontera exterior (esta restricción esta íntimamente ligada con el orden en que se elijen); siendo más preciso, observemos que existe un rectángulo *Rect* de area mínima tal que contiene la poligonal Pol, si A se une con o_1 debe estar dentro del área definida por el lado izquierdo de *Rect*, que denominaremos I (como se ve en la figura 2.26) y análogamente si B se une con o_k debe estar en la zona definida por el lado derecho de *Rect*, que nombraré D, para poder garantizar que la poligonal no se autointersecta. Sin embargo me basta con demostrar que existe una configuración de puntos que lo hace más no darla explicita, por lo que se termina la demostración si tomamos a $A \in \Omega_{ext} \cap I$ (que no es vacía pues $Pol \subset \Omega^{\circ}_{ext}$) y $B \in \Omega_{ext} \cap D$ (no es vacía por lo mismo que la anterior).

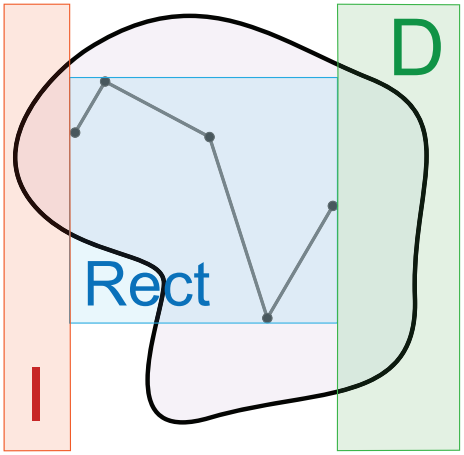


Figura 2.26: Zonas donde elegir los extremos del polígono que divide a Ω_{ext} en dos

Algo que es importante notar es que a pesar de que la malla inicial puede estar autointersectada; esta autointersección va a estar sobre celdas interiores y no sobre la frontera (ni de la región Ω_{ext} ni la de cada agujero); pero el optimizador va a buscar tener celdas con área ϵ-convexas por lo que las parte que se autointersectan se expandirán para ser convexas; este es una ventaja de este método pues nos evita buscar una curva que dividida la región y que no intersecte a las fronteras de la región misma(esto ya es un problema complicado en si mismo).

Finalmente cabe decir que la elección del camino puede ser cualquier curva, para este trabajo se eligió una poligonal por sencillez pero se podría definir una curva en el interior de Ω y hasta podrían darse condiciones sobre ellas para representar fallas o alguna región que sea de interés dentro del problema donde se utilizará la malla.

2.2.5. Implementación

La implementación de este algoritmo se hizo en lenguaje C++, utilizando las bibliotecas de Qt 4.7, dentro del sistema UNAMalla, por lo que el formato de los contornos y las mallas solo han sido adaptaciones de los formatos ya usados en el sistema(**CON** para contornos y **RED** para mallas). La creación del camino que divide la región se hace por medio del programa ContourCreator, para esto sólo se definen los vértices de la poligonal que divide la región de manera manual; a estos se les

llama “puntos especiales” dentro del sistema.

Para la generación de la malla inicial se realizó el algoritmo explicado en el diagrama de flujo 2.1 el algoritmo escrito se encuentra en *genmalla_dlg.cpp* en la función *createTwoMesh()*.

Los tratamientos de contorno se adaptaron para los contornos con agujeros, esto se hizo simplemente pensando que cada agujero es un contorno (sin agujero) con dos fronteras (en vez de cuatro), para dejar fijos a los puntos especiales (y no perderlos en el tratamiento).

Los detalles del sistema se tratan en los capítulos 3 y 4.

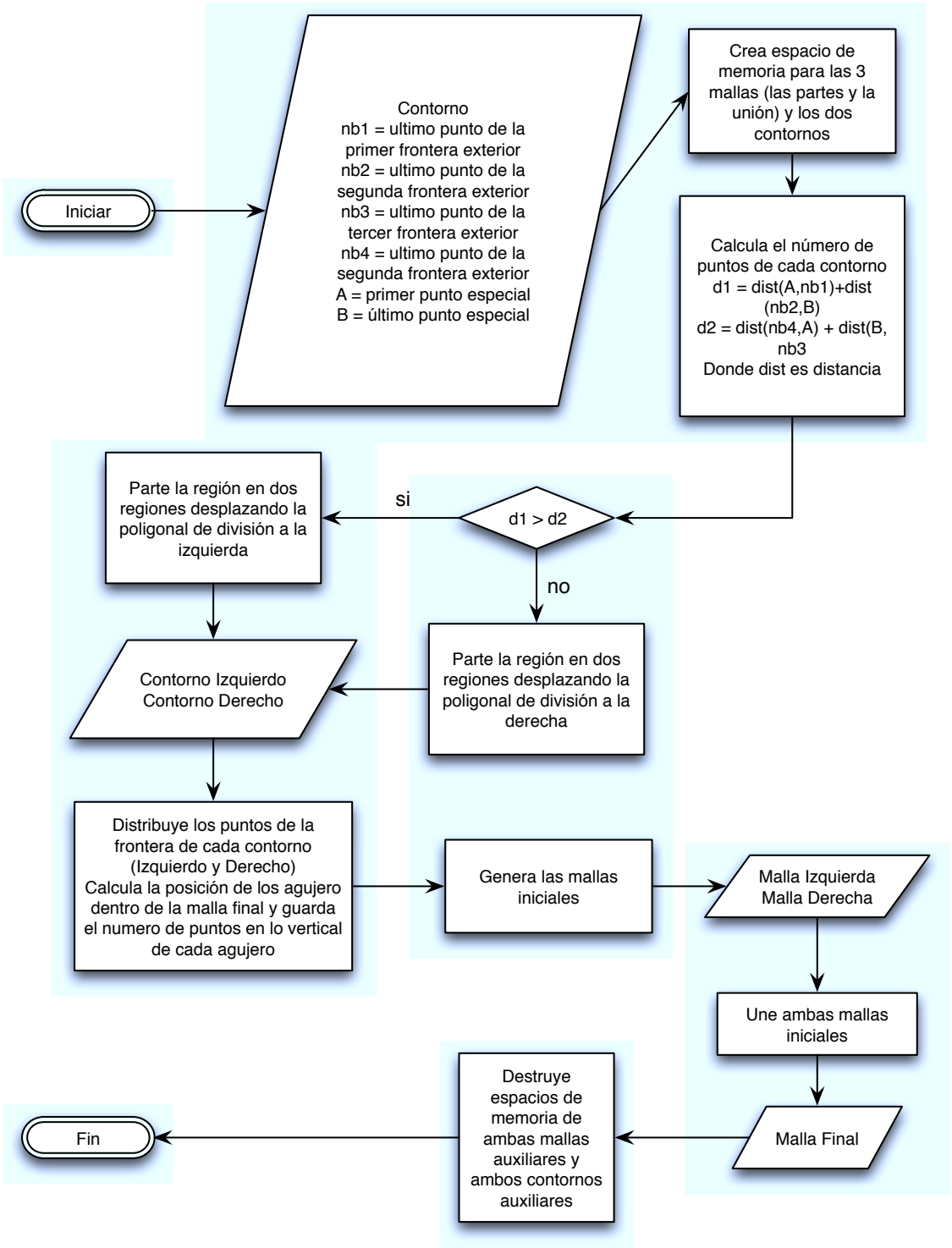


Diagrama de Flujo 2.1: Generación de malla inicial con agujeros

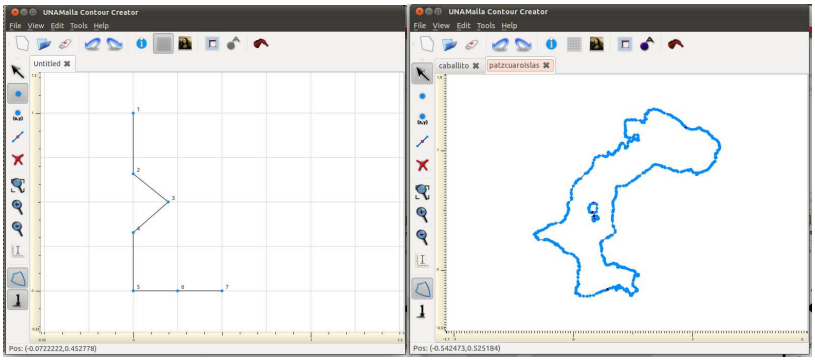
Capítulo 3

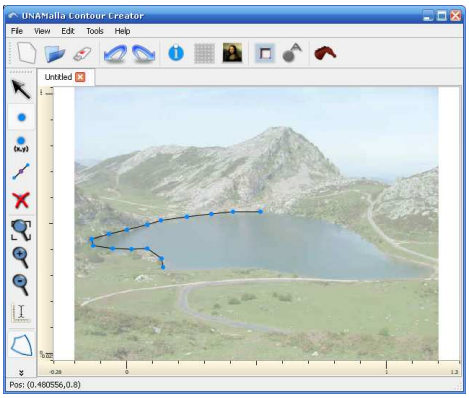
ContourCreator 1.5

ContourCreator es un editor de contornos gráfico multiplataforma con el fin de complementar el programa UNAMalla; este programa provee de herramientas para facilitar la creación de diversos tipos de contornos, incluyendo abrir varios contornos al mismo tiempo y ahora en la versión 1.5 contornos con agujeros.
En este capitulo se hablará sobre las funciones y el uso del programa de una manera breve y un bosquejo general de los algoritmos y la organización del código fuente utilizados dentro del sistema, dando una idea general de la programación del mismo.

3.1. Un editor de Contornos

El propósito de ContourCreator es ofrecer herramientas que hagan muy fácil crear contornos planos complicados; dichas herramientas deben ser manejadas de forma mas o menos intuitiva para el usuario, algo importante de mencionar es que el sistema UNAMalla le llama **"Puntos Especiales"** a los vértices que determinan la poligonal que dividirá la región para la generación de la malla, estos puntos están ordenados por medio de letras (Icosakaihexadecimal[26 unidades]) para diferenciar de la numeración natural del polígono, de esta manera nos da una lista ordenada que nos induce la poligonal, a continuación se muestra pantallas del sistema mostrando su uso:





La pantalla principal esta dividida en cuatro zonas principalmente (Fig. 3.1):

- **Barra de Menús y funciones básicas:** En esta parte se muestra los menús y las funciones básicas (excepto en el caso de Mac OS X, donde sólo se muestran las funciones básicas); como abrir, guardar, nuevo, etc.
- **Barra de Herramientas:** En esta zona se encuentran herramientas básicas de edición y visualización del contorno activo.
- **Área de Trabajo:** También llamado lienzo.En esta área es donde se visualiza el contorno e interactuar directamente sobre los vértices del contorno.
- **Barra de Estado:** Despliega mensajes informativos discretos.

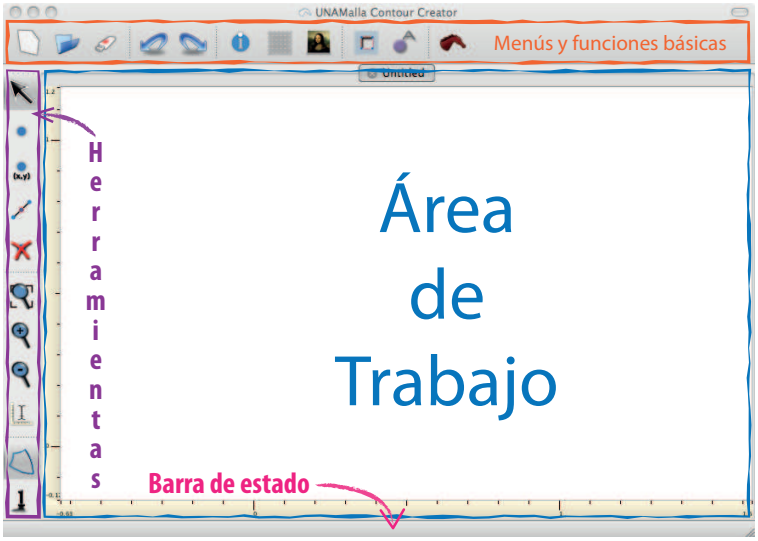


Figura 3.1: Interfaz ContourCreator 1.5

En la siguiente sección se dará una descripción de las funciones que ofrece ContourCreator 1.5.

3.1.1. Herramientas de creación

Agregar Puntos: La definición del contorno esta dada por los vértices que lo componen. La función de agregar puntos especifica dichos vértices, de manera automática conforme se van agregando dichos puntos se añade un segmento que une el nuevo con el anterior. Se ofrece dos formas de agregar estos puntos; una es directamente en la pantalla haciendo clic en la zona del dibujo y otra es por medio del teclado asignando las coordenadas del punto (Fig. 3.2). Para cerrar un contorno basta con hacer clic con el botón derecho de ratón.

Mover Puntos: Esta función permite seleccionar punto directamente en pantalla y arrastrarlo a una nueva posición dentro de la zona de dibujo. También dentro del menú "Herramientas" se encuentra la función para mover un punto seleccionado por coordenadas (Fig. 3.2).

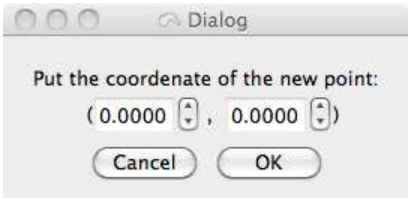


Figura 3.2: Ventana para introducir coordenadas

Añadir agujeros: Dentro del programa la función llamada "Hole Mode" (Modo Agujero Fig. 3.3) permite crear un nuevo contorno una vez que se haya cerrado el anterior, es decir que el modo agujero sirve como "candado" para permitir o no que se sigan creando contornos, pero se necesita usar la función de "agregar punto" para formar punto a punto el nuevo contorno.

Figura 3.3: Botón de "Hole Mode"

3.1.2. Herramientas de edición

Agregar punto medio: Esto permite refinar el contorno en algún segmento que se desee, esto se hace haciendo clic en pantalla sobre el segmento al que se desee agregarle un punto en medio.

Repoblar el contorno: Esta es una generalización del anterior que permite agregar una cantidad constante de puntos en cada segmento del contorno, para hacer esto se debe especificar primero si se desea repoblar un sólo contorno o todos los contornos que haya.

Quitar Puntos: En caso de haber cometido un error al añadir algún punto de más, el programa permite su eliminación. Esto se realiza seleccionando el punto que se pretenda eliminar y activar

la función, o bien primero activando la función y posteriormente hacer clic sobre los puntos directamente en pantalla que se desean eliminar.

Definición de los puntos especiales: Esta función permite especificar que puntos de los contornos construyen la poligonal que divide la región en dos, es importante destacar que para que se puedan especificar estos puntos debe haber al menos un agujero en el contorno. Estos puntos se ordenan alfabeticamente, la idea es ir marcando los puntos de manera consecutiva, para realizar esto, primero se debe activar el botón de puntos especiales e ir seleccionando en pantalla los puntos uno a uno. Se puede cambiar la configuración seleccionando el punto y utilizando el cuadro que esta al costado del botón de puntos especiales (Fig. 3.4). Para verificar que este hecho de manera correcta dentro del menú herramientas se encuentra una función llamada "Verify", o puedes utilizar la función de "Información del contorno". Recuerde que la poligonal que divide la región en dos debe empezar y terminar en el contorno externo, debe tener dos puntos de cada agujero y ser estos consecutivos, además de no inducir un camino que se autointersecte (Fig. 2.21).

Figura 3.4: Botón de puntos especiales

3.1.3. Herramientas Generales

Cuadrícula: Para algunos contornos conviene tener una cuadrícula de fondo para asegurar la alineación de diversos puntos. Esta cuadrícula puede ser configurada para cambiar la finura de la misma y para alinear los puntos a dicha cuadrícula.

Desplegar una imagen de fondo: En algunas aplicaciones ya se tiene una forma definida, como un lago o un río, y es conveniente poder basarse en una imagen del objeto que se quiere representar por medio del contorno, en este sentido se provee esta función, esta imagen puede tener diversos formatos, usualmente PNG, BMP, etc. También el programa permite cambiar la posición de la imagen, escalarla o cambiar su opacidad.

Definir el área de trabajo: El programa permite cambiar las coordenadas donde se va a trabajar para obtener contornos en la región del plano deseada. También se puede definir el tamaño del lienzo en pixeles (que es independiente del sistema coordenado) lo que permite poner imágenes de fondo con mayor resolución.

Comunicación directa con el programa UNAMalla: Esto permite enviar el contorno de manera directa (sin siquiera guardar el contorno) al programa UNAMalla para definir fronteras, aplicar algún tratamiento de contorno o crear la malla estructurada de dicho contorno.

Ver información del contorno: Esto nos permite saber algunas propiedades del contorno; por ejemplo, el área, si tiene subfronteras, y muestra advertencias sobre algún problema que pueda presentar el contorno (tal cuál está), acompañado generalmente de una sugerencia para solucionarlo o evitarlo.

Deshacer / Rehacer: Este es una de las funciones más usuales en muchos programas actualmente y su función es permitir el error del usuario y deshacer / rehacer acciones inmediatas que haya realizado.

Zoom: Esta función en realidad son tres funciones, la primera permite agrandar la imagen, otra para achicar la imagen y la última se encarga de ajustar el tamaño del contorno a la pantalla.

3.2. Implementación

El programa fue escrito en C++ y utilizando las bibliotecas de Qt 4.7 de Nokia [19], y se utilizan algunas rutinas escritas en fortran 90. La figura 3.5 describe el esquema de las clases que se utilizan dentro del sistema (parecido a un diagrama de clases).

Para el programa se han creado varias clases, por lo que es necesario explicar la función de cada una dentro del sistema, algunas son más sencillas de describir que otras, por lo que dedicaré una sección a explicar las que no son sencillas y a continuación explicaré las sencillas. Daremos por sentado el conocimiento del lector sobre la biblioteca Qt. También esperamos un conocimiento fluido de en lenguaje C++ para entender el código del programa y manejo del concepto de la llamada "Programación Orientada a Objetos" (POO).

Como se ve en la figura 3.5 hay diversos tipos de clases; las clases QGraphicsItem, QGraphicsView y QGraphicsScene (esta última no aparece en el diagrama de clases, pero si es usada) son las partes que conforman el esquema gráfico de Qt llamado "Graphic View Framework" un sistema de visualización muy completo que ofrece herramientas para facilitar el trabajo con elementos gráficos en 2D, algo que se utiliza mucho en ContourCreator es la función para detectar a que item se le ha hecho clic y dar facilidades para seleccionar uno o más items (objetos en pantalla), por lo que este esquema es idóneo para el programa; los QGraphicsItem son los elementos gráficos que aparecen en pantalla y a los que se podrá manejar de manera fácil. QGraphicsScene es la clase que se encarga de administrarlos en la pantalla, mientras que QGraphicsView se encarga de pintarlo en pantalla.

CSeg: Esta clase modela a una arista del polígono que representa al contorno, guarda su color, grosor, los vértices en los que incide, además de calcular el centro del mismo. Los CSeg al estar seleccionados se pintan con un grosor mayor.

CPoint: Esta clase modela a un vértice del polígono que representa al contorno, guarda el color y la forma de dibujarse en pantalla además de que también sabe el índice del contorno al que pertenece, el índice que tiene respecto al contorno, el índice de número especial (en caso de no serlo tiene el valor de -1), las coordenadas anteriores que tuvo en pantalla, si pertenece o no a una frontera y a que frontera pertenece, la cuadrícula que tiene, los segmentos que inciden en él, y la etiqueta que muestra su numeración. Los CPoint calculan la posición que van a tener en pantalla al momento de que se suelta el botón izquierdo del mouse después de moverlos, en dependencia de si está o no activada la función de pegarse a la cuadrícula, además mientras es movido indica a los segmentos que inciden en él que deben cambiar su posición.

CAbout: Es una ventana que despliega la información del programa, los autores y la versión del mismo.

UAbout: Al ser parte del paquete UNAMalla también se incluye la ventana que muestra la información del Grupo UNAMALLA.

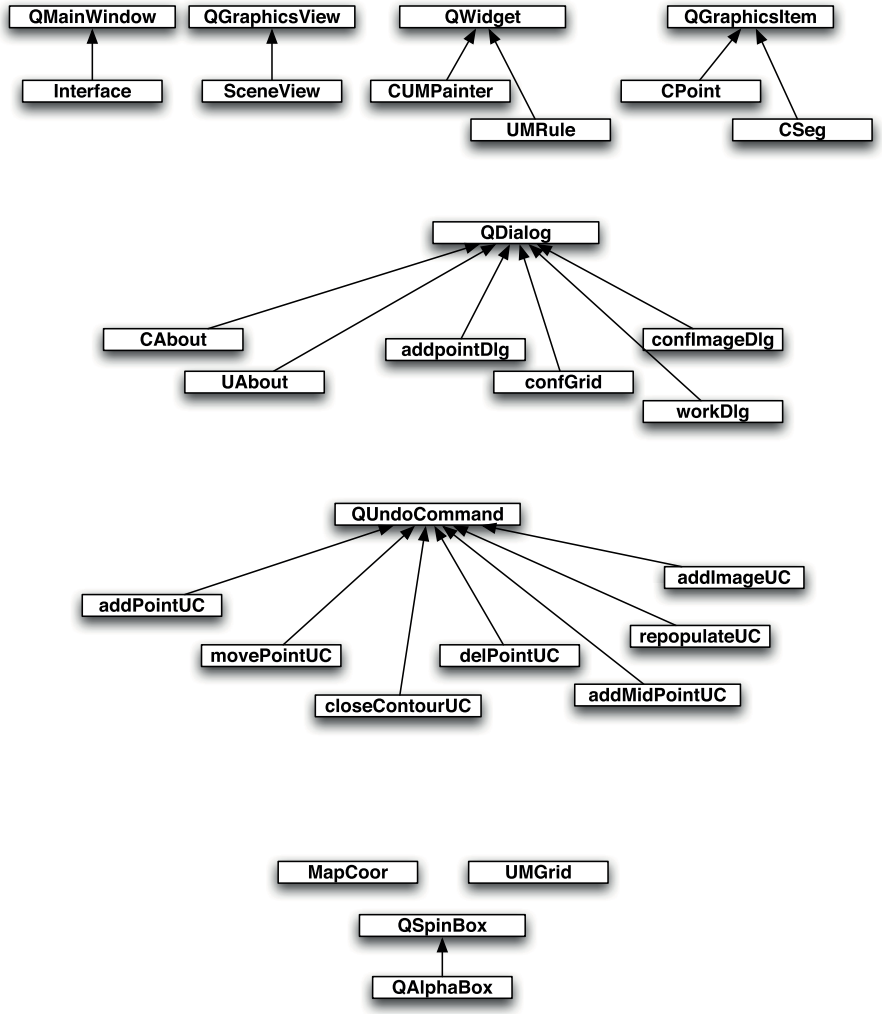


Figura 3.5: Clases del programa ContourCreator 1.5

addpointDlg: Es una ventana que pide las coordenadas de un punto. Se utiliza para agregar un punto por medio de coordenadas y para mover un punto ya creado a alguna otra coordenada (o simplemente ver sus coordenadas).

confGrid: Es la ventana donde se configuran parámetros de la cuadrícula. Para definir la cuadrícula hay dos parámetros, el tamaño de un cuadrado principal y las subdivisiones del mismo, esto con el fin de refinar la cuadrícula.

confImageDlg: Ventana que permite configurar la imagen de fondo. Contiene dos cuadros de texto para introducir las coordenadas de traslación, dos botones para aumentar o disminuir el tamaño

de la imagen, y una barra para cambiar la opacidad de la imagen.

workDlg: La ventana para configurar el área de trabajo, te permite definir las coordenadas del lienzo, además de cambiar el tamaño en pixeles del mismo (lo que permite ver con mayor definición las imágenes de fondo (siempre que estas tengan la misma o mayor de definición que la del lienzo), se recomienda su uso antes de empezar a hacer el contorno, pero se puede hacer en cualquier momento.

QAlphaBox: Es un cuadro de entrada de enteros, esta modificado para desplegar números en Icosakaihexadecimal por medio de la letras del alfabeto (como las etiquetas de las columnas en el programa MS Excel[1]) agregando otro símbolo más: "–", este representa el vacío (para indicar que no pertenece a esta numeración).

3.2.1. Interfaz

La interfaz del sistema está controlada por la clase **Interface** que al ser del tipo QMainWindow ya viene provista con muchas facilidades para el manejo de los menús, barras de herramientas, y de la subventana principal, la configuración gráfica por default es la que se muestra en la figura 3.6. La Zona de Menú es donde se encuentra el menú y las funciones básicas del programa, además de ser una barra de herramienta fija. La Zona de Herramientas es donde se encuentran las funciones principales para la creación y edición del contorno, y de la visualización. La Zona de Trabajo es parte donde se crea y visualiza el contorno y las coordenadas. La Barra de estado es un espacio dedicado para mostrar mensajes sobre información del sistema.

Figura 3.6: Distribución gráfica de la interfaz

[1] Microsoft Excel es propiedad de Microsoft Corporation.

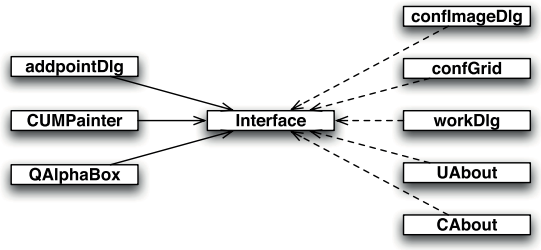


Figura 3.7: Clases relacionadas con Interface

La clase Interface no sólo es una interfaz para el usuario (GUI), además es interfaz para comunicar todas las ventanas entre si, mostrar avisos importantes al usuario, verificar la coherencia de los datos de entrada y salida, dentro de la clase Interfaz se usan otras clases como se muestra en la figura 3.7; donde las flechas punteadas son clases que se utilizan de manera auxiliar dentro del código fuente (generalmente para llamar algunas ventanas) y las normales son elementos (cuando son clases que tienen un uso más frecuente) de la clase Interface. En la Zona de Trabajo se encuentra el Widget principal, este Widget es un QTabWidget que nos permite tener pestañas para cada archivo abierto, a su vez dentro de cada pestaña se encuentra un CUMPainter. Interface se encarga de administrar estas pestañas y asegurar que se pueda mover el usuario sobre cualquiera de las pestañas sin afectar el rendimiento del sistemas ni haber inconsistencias. La mayoría del código es simple, por lo que no me adentraré demasiado en el código, si bien uno de los algoritmos más complicados es el que verifica la coherencia de los puntos especiales, este algoritmo lo describo en el diagrama de flujo 3.1, otro algoritmo que podría ser dificil de entender es el que viene en la función *finishArea* que es para hacer el ajuste del área de trabajo después de definir los parámetros para cambiar, esto se describe en el diagrama de flujo 3.2 , que están más adelante.

3.2.2. Manejo Gráfico

La clase CUMPainter se encarga de distribuir las reglas, visualizar la cuadrícula y el dibujo; además de administrar el contorno al cuál contiene, es interfaz entre la clase Interface y SceneView, UMRule y SceneView, UMGrid y SceneView. La distribución visual que tiene se puede ver en la figura 3.8 y las clases que están relacionadas con esta se puede ver en la figura 3.9.

El sistema de coordenadas que maneja el sistema es administrado por la clase **MapCoor**, esta clase es muy simple; el lienzo (Sceneview) donde se dibujan las figuras tiene un tamaño fijo (tamaño en memoria) este tamaño esta medido por pixeles que además tiene ejes distintos a los que nosotros estamos acostumbrados, sin embargo el usuario necesita un sistema de coordenadas usual, para esto hace falta un cambio de coordenadas. La idea es muy simple, el cambio de coordenadas sólo es informativo, es decir, sirve para dar la posición en pantalla y para la entrada y salida de datos, por lo que sólo cuando entran y salen datos hacia el usuario se hace la conversión; en todo momento se usan las coordenadas del lienzo, dentro del sistema se llaman coordenadas virtuales a las del lienzo y coordenadas reales a las que el usuario quiere, algo importante que notar es que Qt permite manejar coordenadas flotantes(a pesar de ser coordenadas enteras las del lienzo). Los datos de entrada definen la caja que queremos ver en pantalla, introduciendo las coordenadas del vértice inferior izquierdo: $startx$ y $starty$, la longitud de cada lado $lenx$ y $leny$, por último se utiliza el ancho y largo del lienzo,

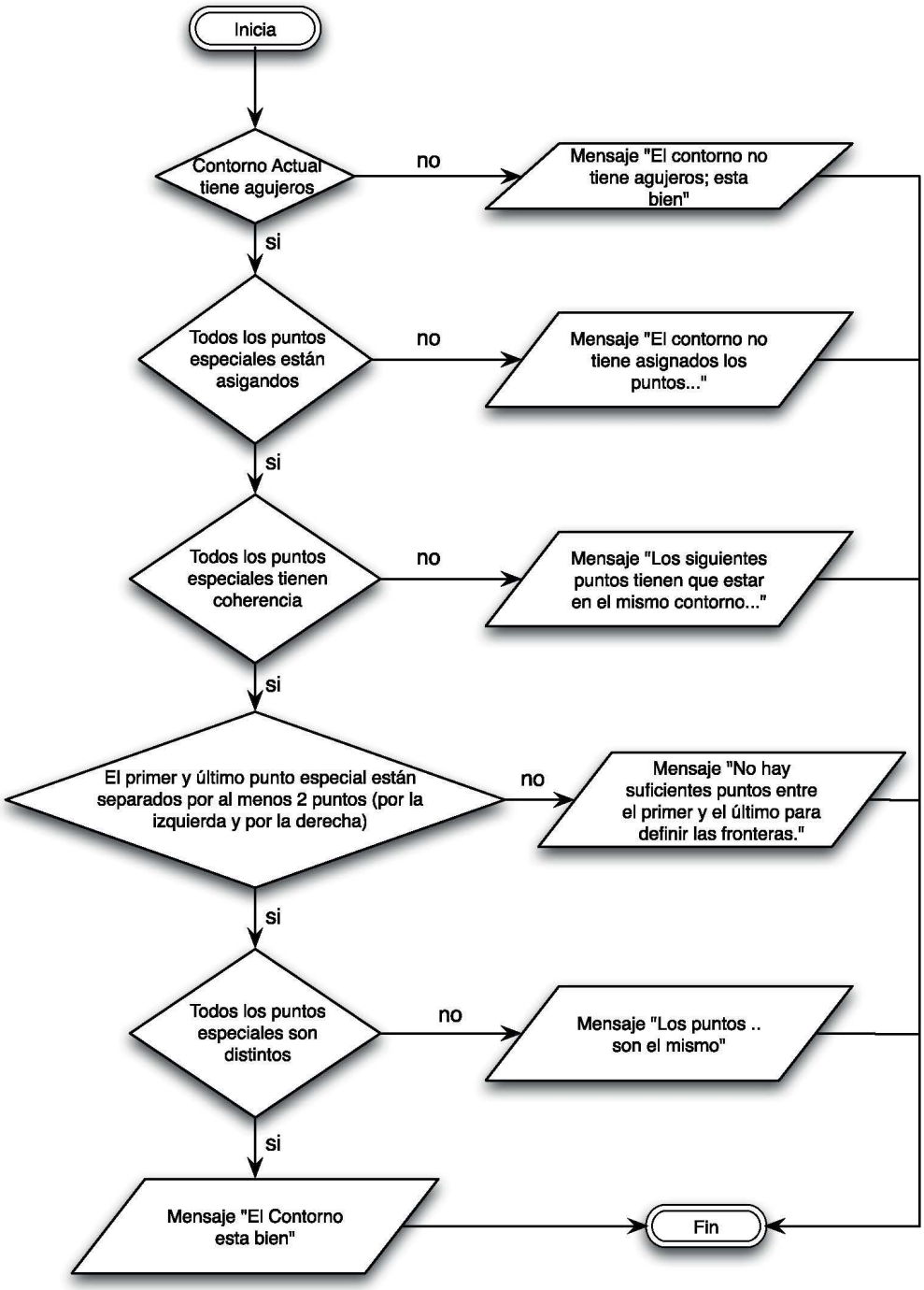


Diagrama de Flujo 3.1: Función *verifyCont*

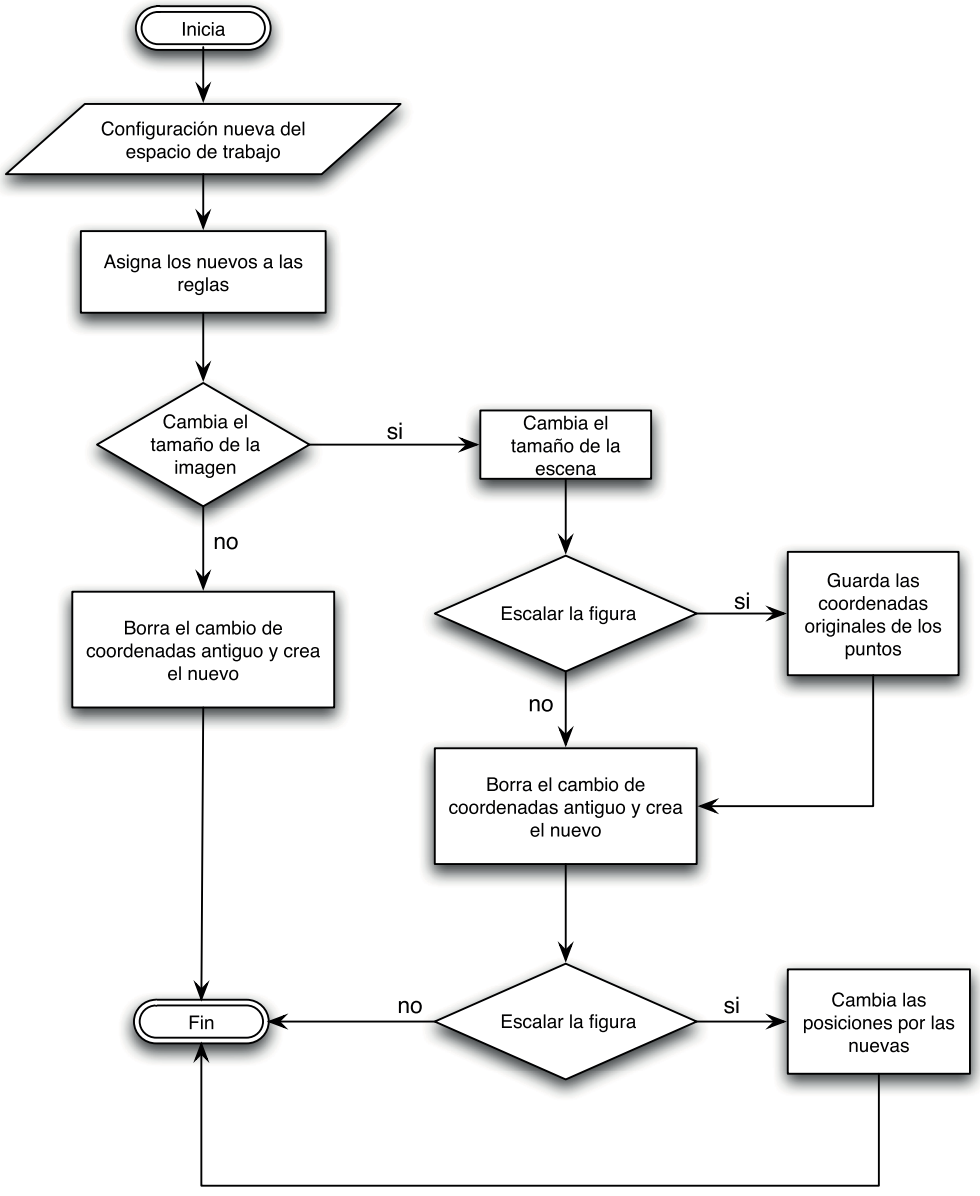


Diagrama de Flujo 3.2: Función *finishArea*

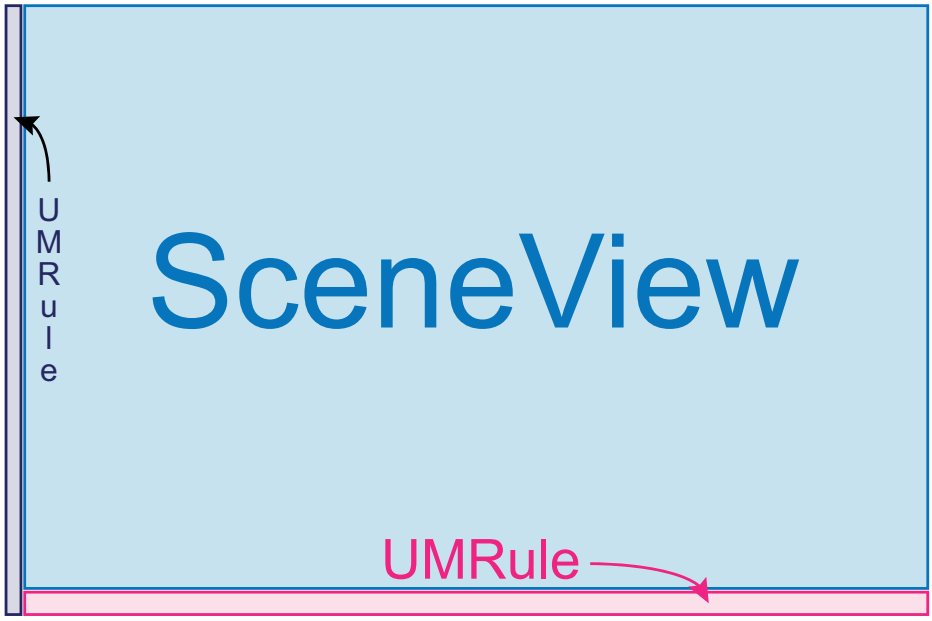


Figura 3.8: Distribución gráfica de CUMPainter

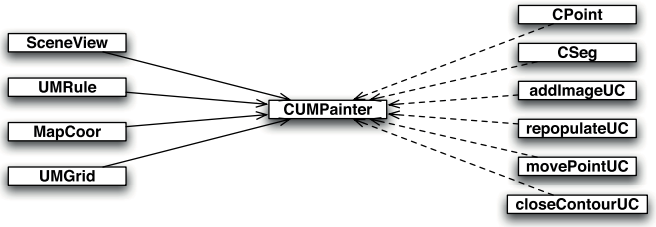


Figura 3.9: Clases relacionadas con CUMPainter

para calcular el sistema de coordenadas adecuado.

UMRule: Utiliza esta misma idea, aunque lo hace para calcular sólo una proyección, o bien un lado de la caja que queremos ver, la intención es dibujar una regla graduada; hay que especificarle si la regla es vertical u horizontal pues su tamaño depende de eso, además de que esta hecho a la medida para estar sincronizada con las coordenadas del lienzo (hay que recordar que esta en otra clase).

UMGrid: También esta basado en la misma idea de las coordenadas, pero con estas lo que hace es dibujar una cuadrícula, para el UMGrid la cuadrícula devuelve una QGraphicsPathItem que permite una buena calidad de las líneas, estas servirán de fondo, pero también dado un punto lo ajusta a la cuadrícula.

SceneView: Es el lienzo donde se pintan los objetos, además de que se encarga de avisar cuando se ha movido algún punto, así como cuando se hace Zoom, o se ha movido la hoja; y sirve de interfaz entre los objetos del dibujo y otras clases, administra cuando se puede crear o no un agujero, si esta o no cerrado el contorno actual, pintar o no las etiquetas de cada punto, además de que es el que aplica directamente sobre los objetos las funciones (como borrar punto, agregar el punto medio, etc.) ya que es la clase que sabe que objetos están seleccionados, las clases que están relacionadas con SceneView son las que se ven en la figura 3.10.

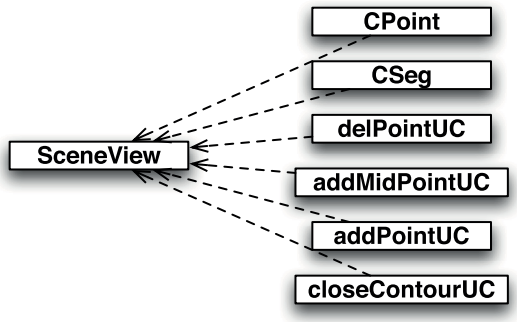


Figura 3.10: Clases relacionadas con SceneView

El sistema UNAMalla maneja de manera nativa una estructura para los contornos y las mallas; sin embargo, esta estructura no es adecuada para la creación de los contornos ya que no permite con facilidad la dinámica necesaria para crear y quitar puntos en distintas posiciones de la estructura, por lo que el programa maneja otra estructura, una estructura de lista de CPoint, y a los agujeros como una lista de listas de CPoint, pero al leer o guardar un archivo se debe cambiar a la estructura usual del sistema, para esto la clase CUMPainter tiene dos funciones una que llamada *setContour* para convertir de la estructura llamada *contour* a la lista de CPoint, y otra llamada *Contour* para el proceso inverso. Sin embargo, la función *setContour* no sólo debe convertir los tipo, si no que además busca el sistema de coordenadas correcto para dicho contorno. El contorno tiene una restricción, los puntos adyacentes deben estar separados por al menos 6 pixeles, ya que el sistema que detecta el clic depende de un cuadrado, este cuadrado tiene de lado 6, si se empalman los cuadros no funciona de manera correcta la función para seleccionar los puntos. Este posiblemente es el problema más grave del programa, para intentar solventarlo se escala el tamaño del lienzo para que quepa teniendo el tamaño 6 de separación, para esto el algoritmo sigue el diagrama de flujo 3.3 para abrir el contorno y el diagrama de flujo 3.4 para guardarlo.

3.3. Comandos

Una de las funciones más importantes del programa es la función "Deshacer / Rehacer" muy común en muchos programas actuales, esto permite cancelar de manera inmediata cualquier acción e ir regresando en las acciones hasta encontrar algo deseado o volver a hacer las cosas. En el diseño de un contorno esto es imprescindible pues el error humano es un factor muy importante, esto hace un poco más complicado el tratamiento del programa pero no es tan difícil de explicar ya que esto se basa en una lista de comandos, esta es la clase QUndoStack, donde se guardan todas las acciones que hace el usuario, o por lo menos las que el programador indica. Para que esto funcione la lista se recorre conforme lo pide (atrás / delante) pero para ir hacia atrás se debe ejecutar antes una función llamada *undo* y para ir hacia delante se debe hacer otra llamada *redo*, esas funciones vienen dentro de la clase QUndoCommand, por lo que todas las clases heredadas de QUndoCommand son las funciones propias del programa. Cuando se crea un QUndoCommand se ejecuta el constructor, y cuando este es insertado en la lista QUndoStack se ejecuta la función redo, por lo que el redo es una función que se encarga de crear la acción, mientras que undo se encarga de deshacerla, por ejemplo, la función redo se **addPointUC** agrega un punto en la escena en ciertas coordenadas y también en la lista de puntos, mientras que undo se encarga de quitarlo de la escena y de la estructura de puntos. En la mayoría de las clases QUndoCommand, dentro de los constructores es donde se crea el espacio de memoria y configura los nuevos elementos, en el redo se encarga de ponerlos en donde sea necesario, y el undo de quitarlos de las estructuras, pero no se borra ni crea memoria dentro del undo o el redo, para no comprometer la estabilidad del sistema, esto deja en memoria cada objeto creado a lo largo del uso del programa. En general las funciones son simples, voy a describir en pseudocódigo los simples y con un diagrama de flujo los que considero más complicados.

addPointUC:
Constructor:
Input: posición, SceneView, QUndoStack
Guarda la posición, en espacio el SceneView;
Crea el espacio en memoria para el nuevo punto;
Crea la etiqueta de su número y lo asigna al punto;
if *hay más de un punto* **then**
  Crea el segmento que unirá el nuevo con el anterior;
end

Redo:
Agrega el punto nuevo a la lista de puntos;
Agrega su etiqueta a la lista de etiquetas;
Agrega a la escena el punto y la etiqueta;
Asigna como posición y antigua posición la inicial;
if *hay más de un punto* **then**
  Asigna al segmento nuevo como puntos de partida y final al penúltimo punto y al nuevo;
  Asigna el segmento como incidente en el penúltimo punto y el nuevo;
  Agrega el segmento a la escena;
end

Undo:

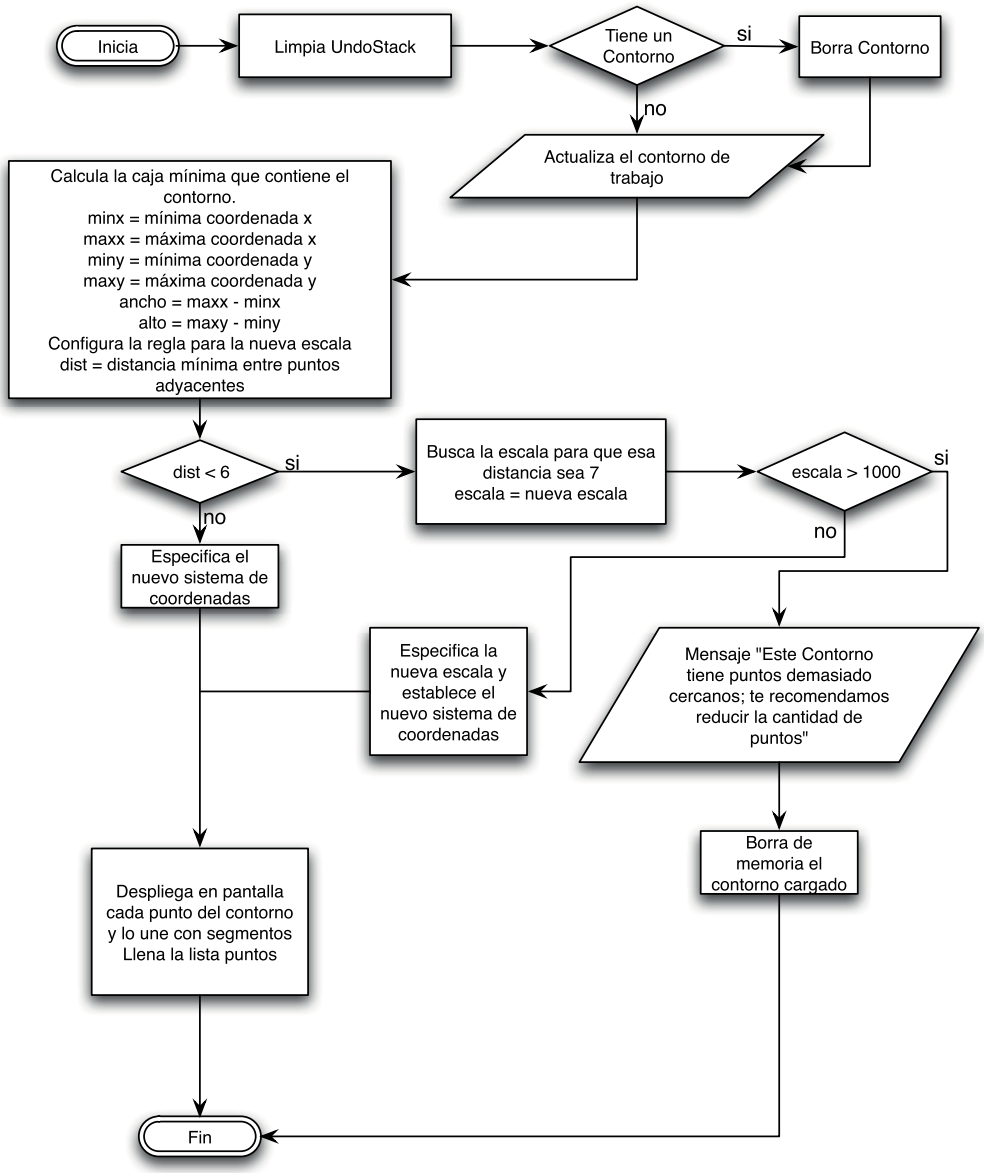


Diagrama de Flujo 3.3: Función *setContour*

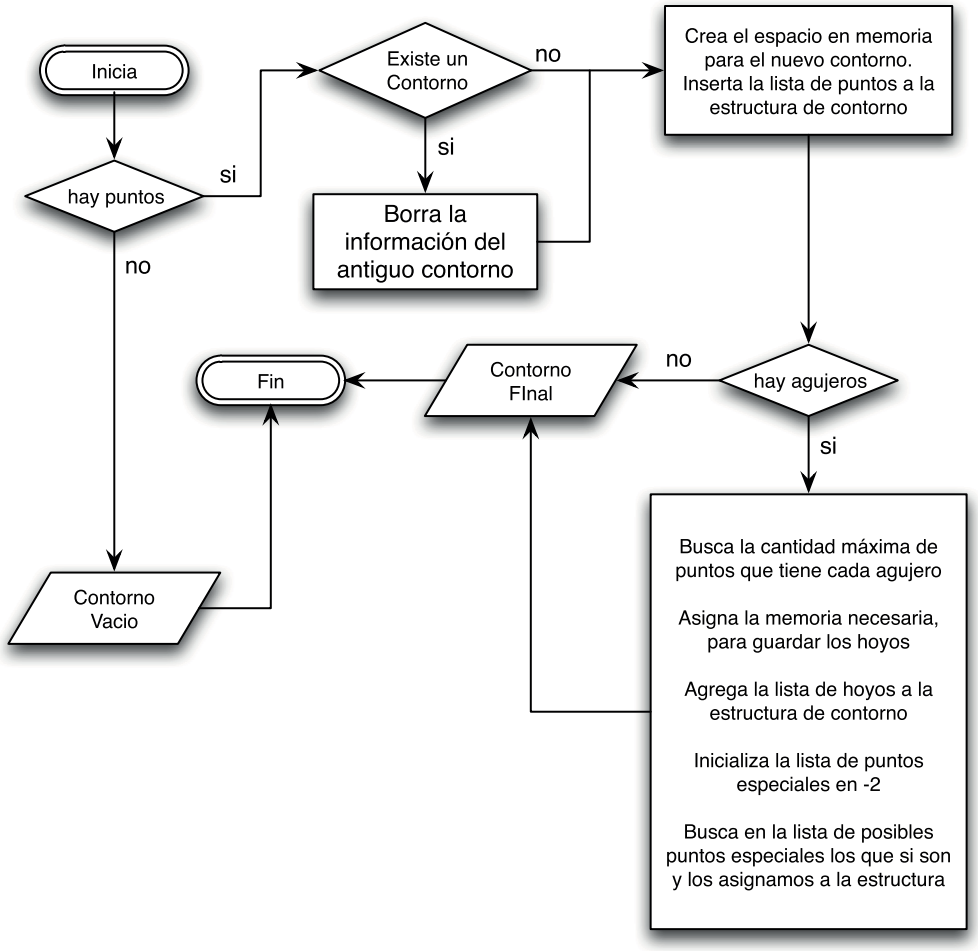


Diagrama de Flujo 3.4: Función *Contour*

if *el punto tiene segmentos* **then**
| Quita el segmento de la lista de segmentos del punto anterior el nuevo;
| Quita el segmento de la escena;
end
Quita el punto y su etiqueta de la escena;
Quita al punto y su etiqueta de la lista respectiva;

closeContourUC:
Constructor:
Input: SceneView
Guarda en espacio el SceneView;
Crea el espacio en memoria para el nuevo segmento;
Asigna el índice del contorno actual;

Redo:
Asigna al segmento nuevo como puntos de partida y final al último y primer punto;
Asigna el segmento como incidente en el último punto y el primero;
Agrega a la escena el segmento;
Indica que el contorno actual esta cerrado;
if *está en modo agujero* **then**
| Crea el espacio de memoria para la nueva lista de puntos (el siguiente agujero).;
end

Undo:
if *Sigue en modo agujero* **then**
| Quita el espacio reservado para la nueva lista de puntos (del posible nuevo agujero);
end
Del último punto del contorno actual revisa los segmentos que inciden en él;
if *el primer segmento incide en el primer y último punto* **then**
| Quita el primer segmento del escenario y como segmento de los puntos primero y último;
end
else
| Quita el segundo segmento del escenario y como segmento de los puntos primero y último;
end
Indica al espacio que no está cerrado el contorno actual;

delPointUC *redo* se describe en el diagrama de flujo 3.5 , y *undo* se describe en el diagrama de flujo 3.6.
Constructor:
Input: punto, SceneView
Guarda el punto y en espacio el SceneView;
Guarda la posición del punto y su índice dentro del contorno;
if *en el punto inciden dos segmentos* **then**
| Crea el espacio en memoria para el nuevo segmento;
| le asigna el índice del contorno actual;
end
Marca como falsa la bandera “abri” (para indicar “cambié de cerrado a abierto el contorno”);

addMidPointUC:

Constructor:
Input: segmento, SceneView, QUndoStack
Guarda en espacio el SceneView;
Guarda el segmento, y los dos puntos en los que incide;
Crea el espacio en memoria para el nuevo punto y los dos nuevos segmentos;
Configura el punto y los segmentos.;

Redo:
Guarda el índice del contorno actual;
Cambia al contorno donde está el segmento;
Quita el segmento de la escena y como segmento que incide en los puntos donde incidía;
Pone el nuevo punto entre los dos donde incidía el segmento;
Asigna los nuevos segmentos como incidentes con sus puntos respectivo;
Reenumera los puntos siguientes;
Agrega a la escena el punto y los dos segmentos nuevos;
Regresa al contorno que estaba antes;

Undo:
Guarda el índice del contorno actual;
Cambia al contorno donde esta el segmento;
Quita los puntos y segmentos de la escena y de la estructura;
Vuelve a poner el segmento anterior;
Reenumera los puntos;
Regresa al contorno que estaba antes;

movePointUC:
Constructor:
Input: punto, posición antigua, posición nueva
Guarda el punto y las posiciones;

Redo:
Mueve el punto a la posición nueva;

Undo:
Mueve el punto a la posición antigua;

addImageUC:
Constructor:
Input: ImagenActual, ImagenAntigua, SceneView
Guarda en espacio el SceneView;
Guarda la imagen actual escala al tamaño del lienzo;
if *ImagenAntigua tiene datos* **then**
  Activa la bandera de imagen antigua;
  y la guarda;
end
else
  Desactiva la bandera de imagen antigua;
end

Redo:

Asigna la imagen dentro de la clase QGraphicsPixmapItem;
if *No existe imagen antigua* **then**
| Agrega a la escena la QGraphicsPixmapItem;
end
Pone centrada la imagen y con opacidad a la mitad;

Undo:
if *Existe la imagen antigua* **then**
| Pone la antigua imagen;
end
else
| Quita la imagen de la escena;
end

changeImageUC: Esta clase se refiere al cambio de posición tamaño u opacidad de la imagen.
Constructor:
Input: Imagen, posición antigua, posición nueva, tamaño antiguo, tamaño nuevo, opacidad antigua, opacidad nueva
Guarda todas las entradas;

Redo:
Asigna la imagen con los parámetros nuevos;

Undo:
Asigna la imagen con los parámetros antiguos;

repopulateUC: El constructor se describe en el diagrama de flujo 3.7 ,*redo* se describe en el diagrama de flujo 3.8 , y *undo* se describe en el diagrama de flujo 3.9.

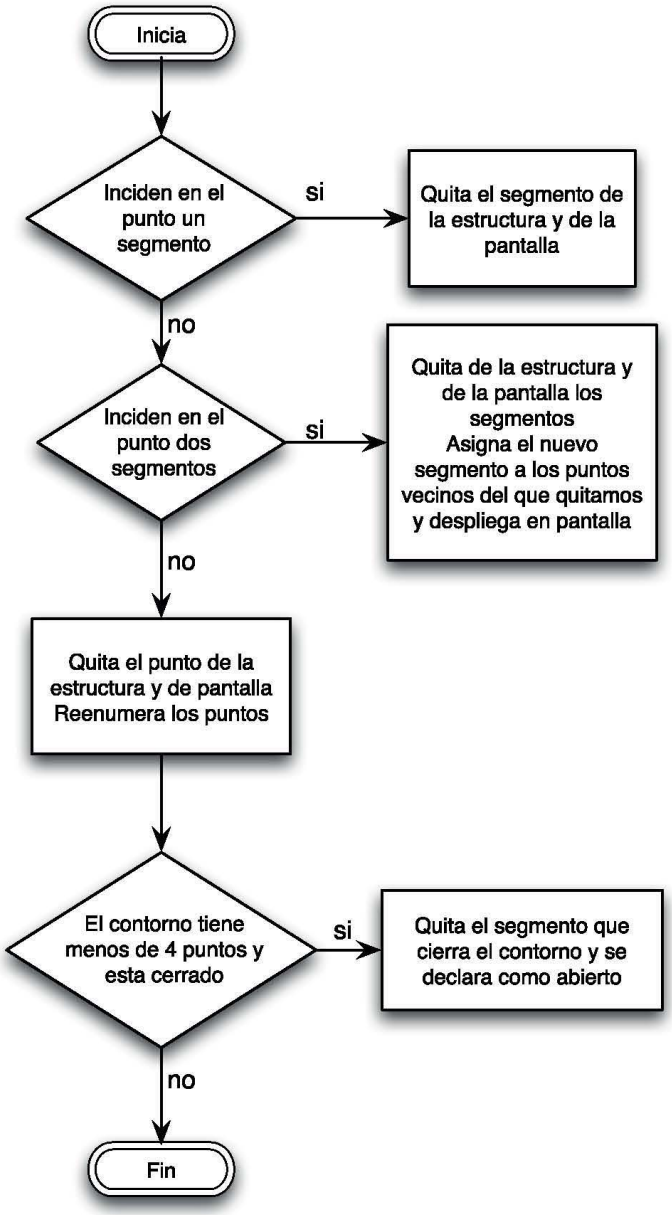


Diagrama de Flujo 3.5: Función *redo* de la clase *delPointUC*

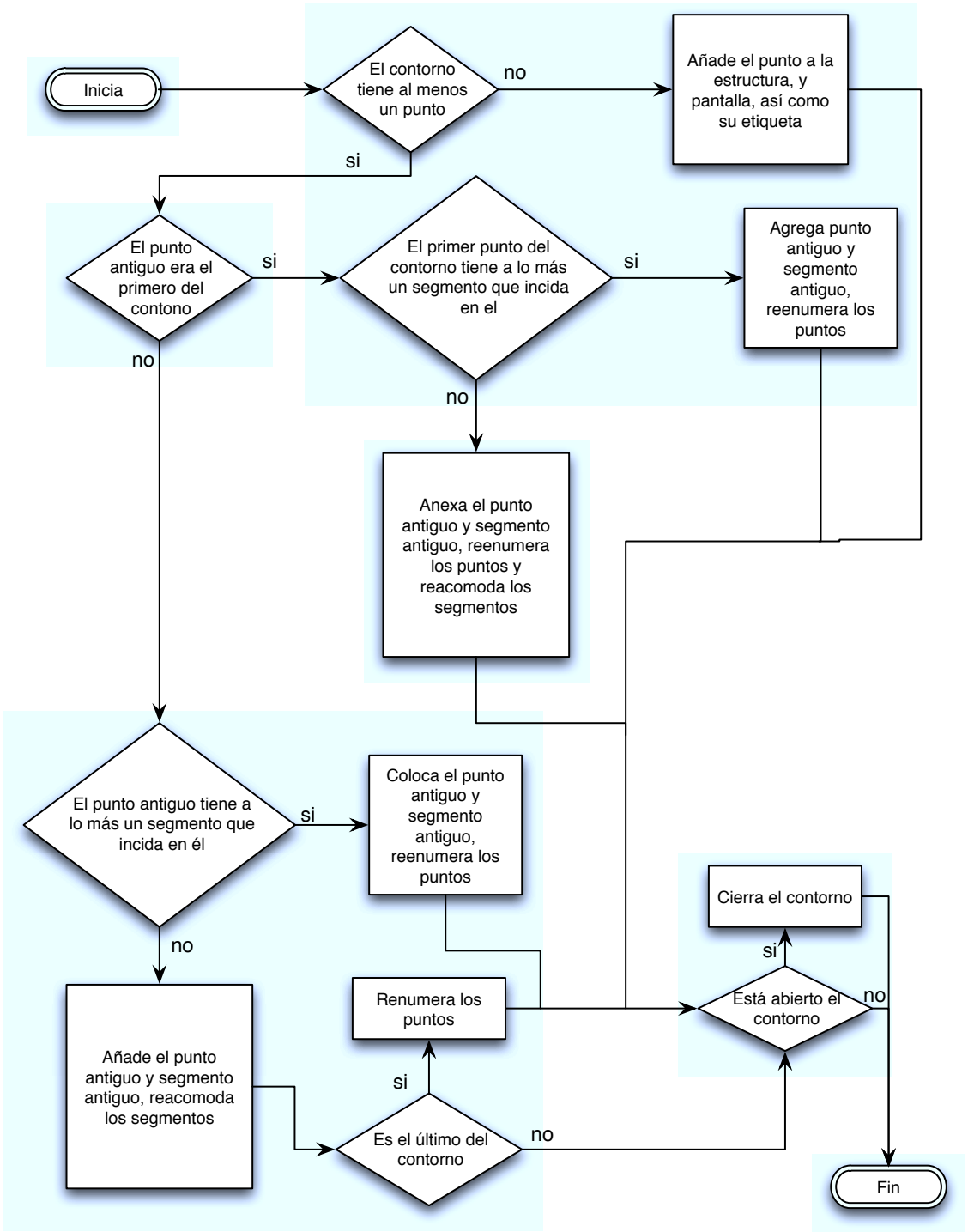


Diagrama de Flujo 3.6: Función *undo* de la clase *delPointUC*

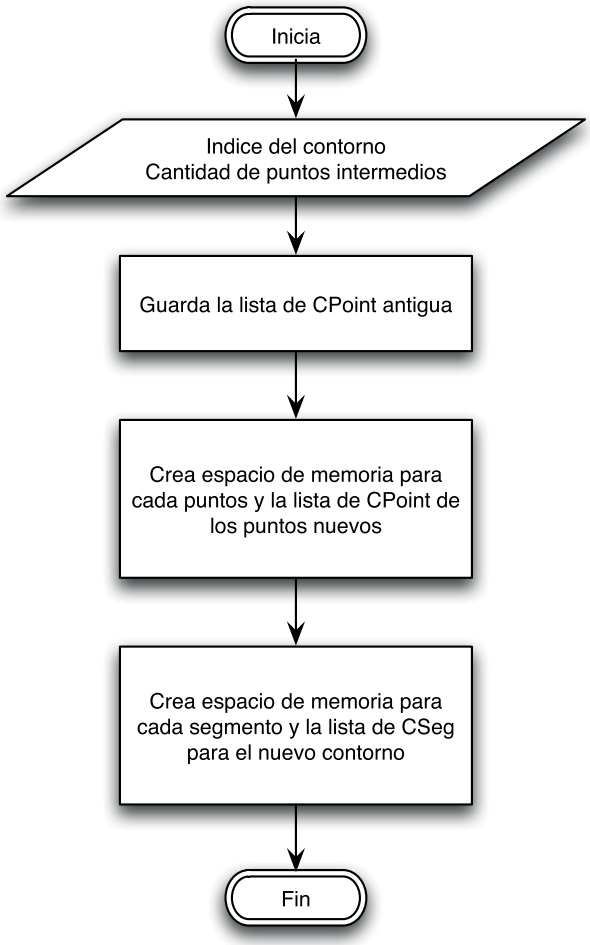


Diagrama de Flujo 3.7: Constructor de la clase *delPointUC*

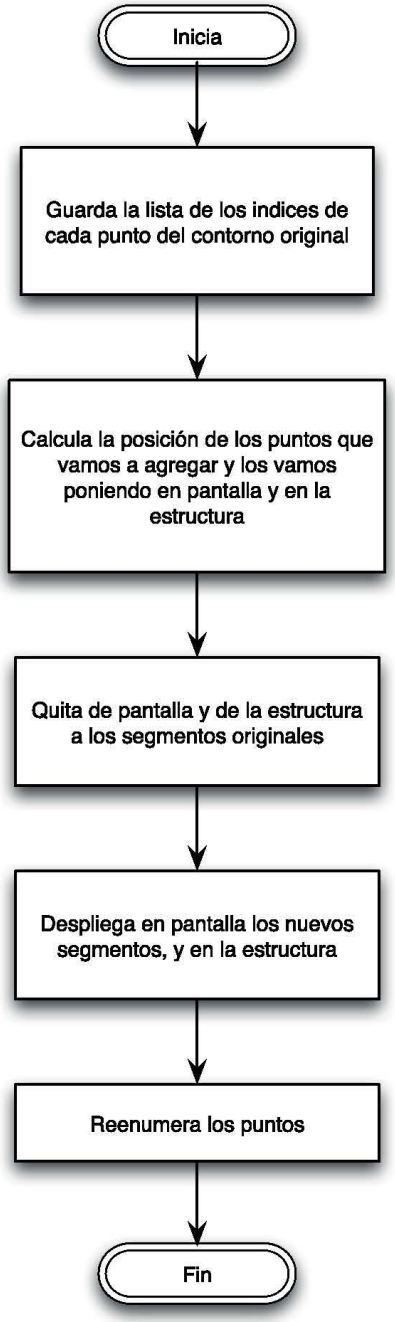


Diagrama de Flujo 3.8: Función *redo* de la clase *repopulateUC*

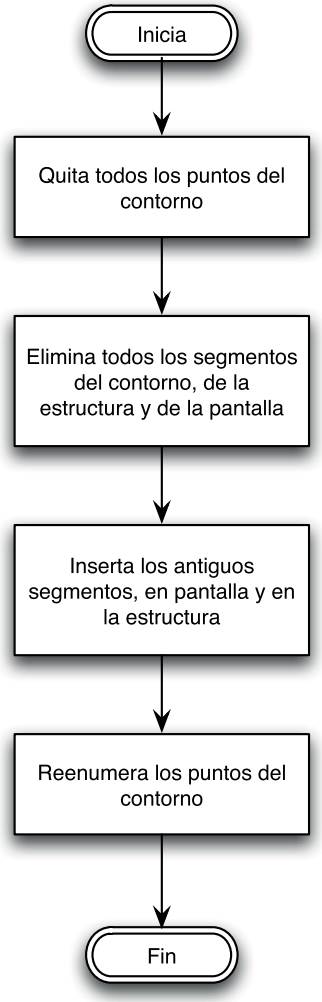


Diagrama de Flujo 3.9: Función *undo* de la clase *repopulateUC*

Capítulo 4

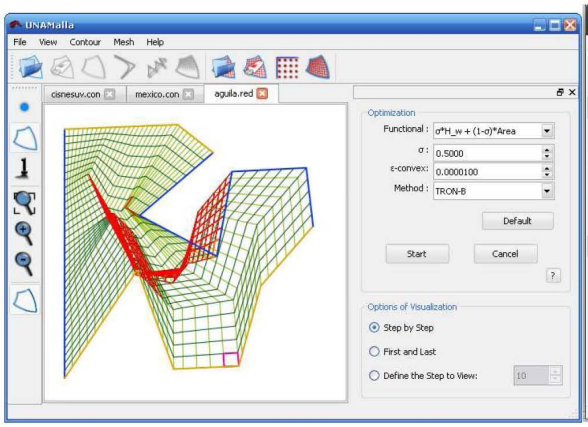
UNAMalla 4.5

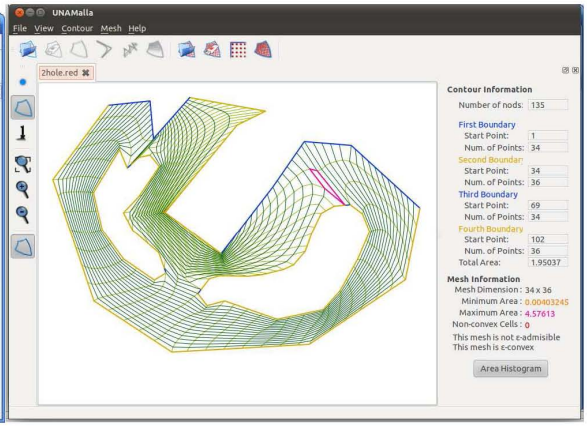
El programa UNAMalla es el resultado de más de 20 años de enorme esfuerzo de un grupo de trabajo de diversos profesores de la Facultad de Ciencias en la UNAM, el Instituto de Cibernética y Física en Cuba, la Facultad de Ciencias Físico-Matemáticas en la UMSNH, la Facultad de Ciencias en la UAC, construyendo programas o módulos que realizan tareas específicas para la generación de mallas razonables en regiones planas. A raíz de esto se han elaborado alrededor de 15 tesis de diversos niveles y algunas publicaciones. Para más información sobre la historia y el sistema UNAMalla 4.5 diríjase al sitio web : `http://www.matematicas.unam.mx/unamalla`.

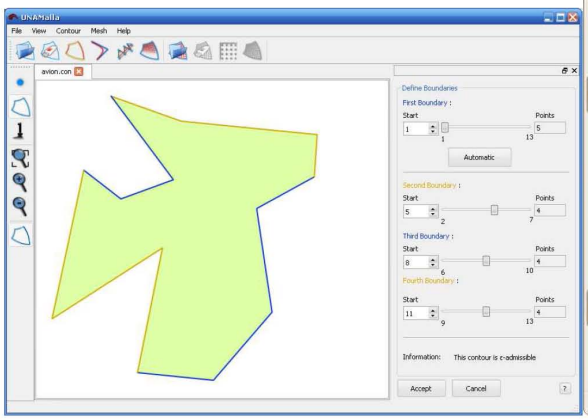
En este capítulo se dará una breve explicación del funcionamiento del sistema y de la implementación de la interfaz gráfica.

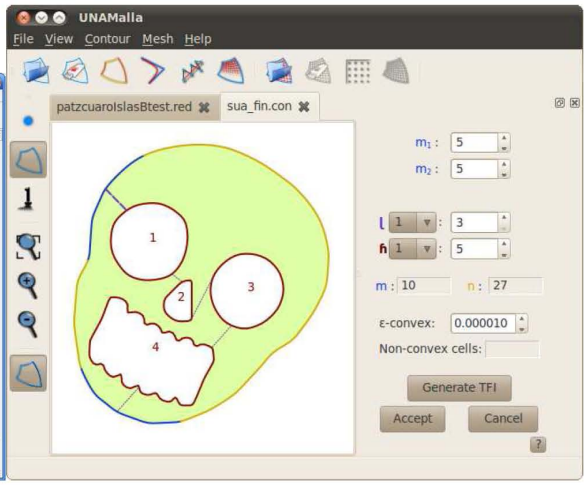
4.1. Generador de Mallas Estructuradas ϵ-Convexas

El programa UNAMalla provee de herramientas para la creación semiautomática de mallas estructuradas. La malla no depende sólo de la región por lo que permite que el usuario decida los parámetros que determinan la malla según sus necesidades. El entender los parámetros requiere de práctica; no es fácil explicar el cómo se deben elegir los parámetros para que la malla estructurada cuente con alguna característica en especifico; sin embargo, el programa UNAMalla ya tiene definidos parámetros que generan mallas de buena calidad para muchos casos [6]. Las herramientas que ofrece el programa están divididas en dos partes: para contornos y para mallas; dichas funciones hacen uso de la subventana "Tool Window" que se encuentra por default en la parte izquierda de la pantalla, esta subventana tiene la cualidad de poder desplegarse de manera independiente a la ventana principal, en caso de que sea cerrada por error se puede volver a mostrar con la función "Tool Window" dentro del menú "View". Cabe mencionar que UNAMalla permite abrir varios contornos y mallas a la vez. A continuación se muestran algunas pantallas del sistema.


UNAMalla
File View Contour Mesh Help
cisnesuv.con
mexico.con
aguila.red
Optimization
Functional :
σ :
ε-convex:
Method :
Default
Start
Cancel
Options of Visualization
Step by Step
First and Last
Define the Step to View:


UNAMalla
File View Contour Mesh Help
2hole.red
Contour Information
Number of nods:
First Boundary
Second Boundary
Third Boundary
Fourth Boundary
Total Area:
Mesh Information
Area Histogram


UNAMalla
File View Contour Mesh Help
avion.con
Define Boundaries
First Boundary :
Second Boundary :
Third Boundary :
Fourth Boundary :
Automatic
Information:
This contour is ε-admissible
Accept
Cancel


UNAMalla
File View Contour Mesh Help
patzcuaroIslasBtest.red
sua_fin.con
m1 :
m2 :
m :
n :
ε-convex:
Non-convex cells:
Generate TFI
Accept
Cancel

La pantalla principal esta dividida en cinco zonas principalmente (Fig. 4.1):

- **Barra de Menús y funciones básicas:** en esta parte se muestra los menús y las funciones básicas (excepto en el caso de Mac OS X, donde sólo se muestran las funciones básicas); como abrir, guardar, nuevo, etc.
- **Barra de Herramientas:** en esta zona se encuentran herramientas de visualización de la región activa.
- **Controles de Herramientas:** esta área esta destinada a mostrar los botones y diversos controles para utilizar las herramientas que ofrece el sistema, es la llamada "Tool Window".
- **Visor:** en esta zona se visualiza el contorno o malla que representa a la región de trabajo.
- **Barra de Estado:** despliega mensajes informativos discretos.

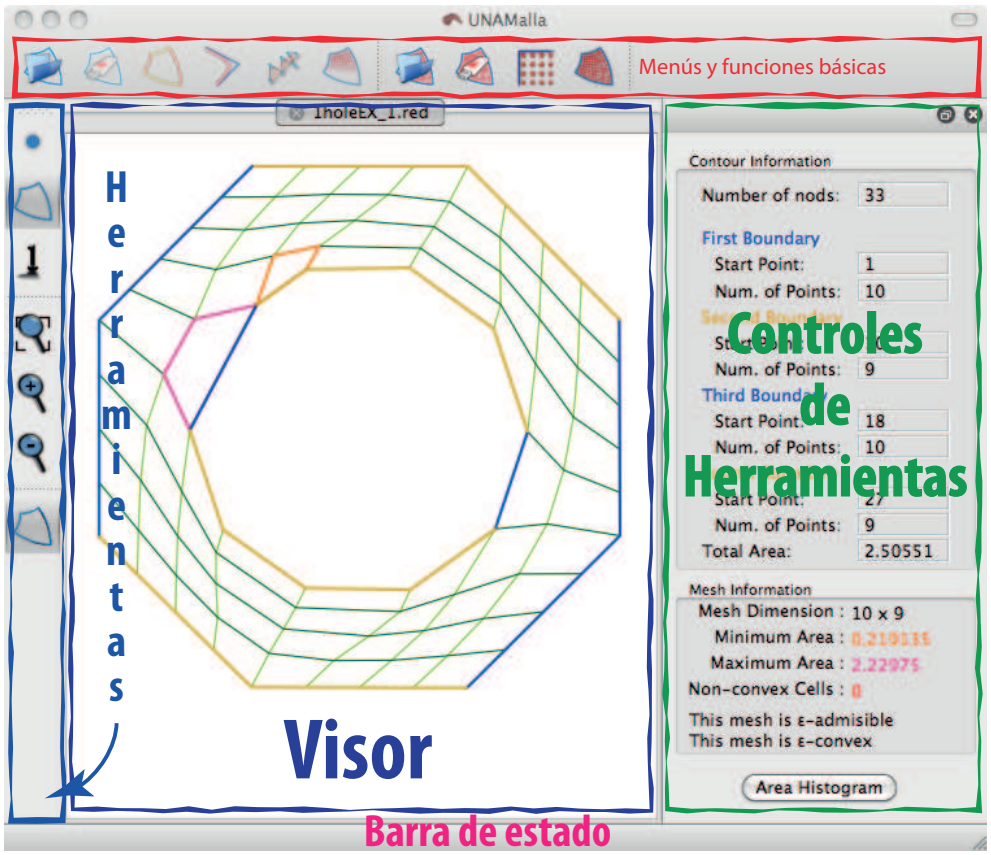


Figura 4.1: Interfaz UNAMalla 4.5

4.1.1. Tratamiento de Contornos

El programa UNAMalla a partir de su versión 4 tiene conexión directa con ContourCreator por lo que puede enviar al ContourCreator un contorno que se quiera editar de manera manual (adición de puntos, modificación de ellos), para esto existe el botón "Edit" que se encuentra junto con la información del contorno dentro de la "Tool Window". A continuación se muestran pantallas del sistema, para mostrar algunas de sus funciones.

Información del Contorno: esta ventana muestra algunos datos sobre el contorno como el número de puntos, la configuración de las fronteras, el área y en mensaje con advertencias para el usuario. En caso de que el contorno tenga alguna característica que no le permita el uso adecuado del sistema no se usa esta ventana si no que se muestra un mensaje emergente en pantalla que advierte la mala condición del contorno, por ejemplo cuando este tiene autointersecciones.

Definición de las fronteras: esta función especifica uno de los parámetros más importantes para la malla. Si recordamos la malla es resultado de un mapeo del cuadrado unitario a la región, para acotar este mapeo el usuario puede especificar que secciones del contorno vienen de los lados del cuadrado; de un mismo color se dibujan lados opuestos, el color azul se refiere a los lados de abajo y arriba, mientras el amarillo los lados izquierdo y derecho, con esta función podemos determinar las zonas del contorno que están asociados a cada lado. La elección de estas fronteras es decisiva para la malla final.
Esta especificación se hace por medio del botón "Automatic" que define una distribución de las fronteras, esta basada en la longitud de arco, intentando distribuirlo de manera uniforme. Una vez que se tienen las fronteras definidas uno puede redefinir las fronteras, si uno desea cambiar el primer punto se debe utilizar el botón automatic, mientras que para los demás puntos se pueden deslizar los botones que se encuentran dentro de esta ventana, dependiendo de la frontera que uno desea cambiar, una vez elegida una configuración se debe presionar el boton "Aceptar". (Fig. 4.2)

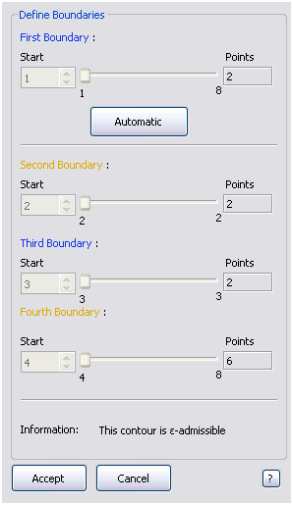
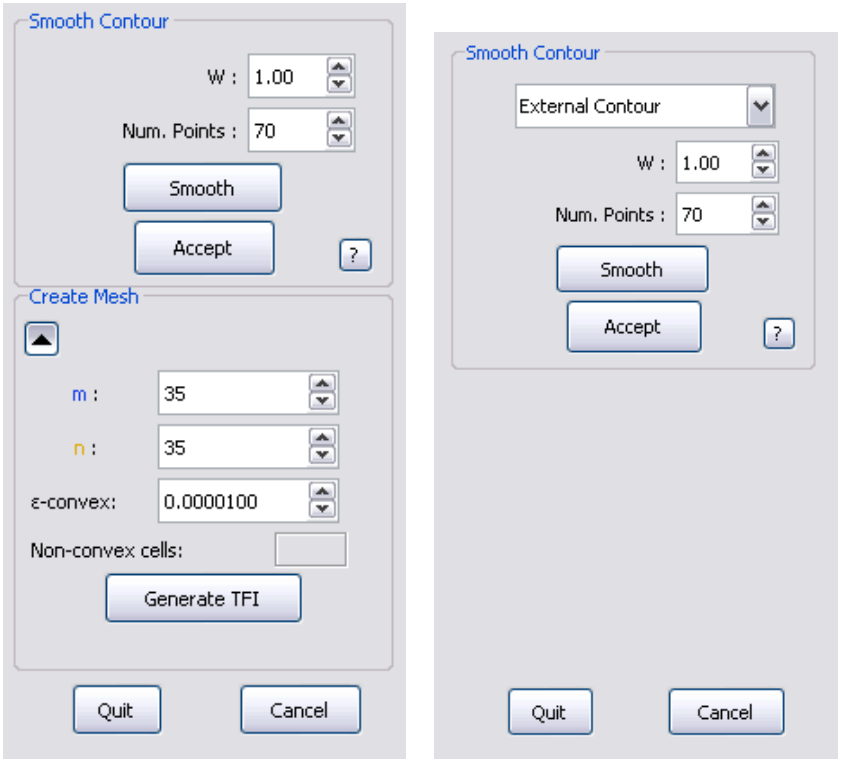


Figura 4.2: Controles para definir las fronteras

Suavizamiento de contorno: esta función provee un método de suavizamiento de contorno por medio de splines cónicos [8] y sirve para mejorar la distribución de la malla resultante, ya que en algunos casos los picos más acentuados inducen mallas muy elongadas. En esta función es muy simple de usar, primero se asigna un peso ω, si $\omega < 1$ el contorno suavizado se aleja del original, mientras que si $\omega > 1$ el contorno suavizado se acerca más al original, y un segundo parámetro que especifica la cantidad de puntos de cada frontera, por default setenta. En el caso de contornos sin agujeros se tienen cuatro fronteras, cuando se usan contornos con agujeros depende del subcontorno elegido; si es el exterior sigue conservando las cuatro fronteras mientras que cada agujero tiene dos fronteras inducidas por los puntos especiales. En caso de los contornos sin agujeros también se ofrece la utilidad de generar la malla inicial directamente utilizando el contorno suavizado. Es importante hacer notar que no es la misma malla la creada con este método que si se suaviza el contorno y sobre este se construye la malla inicial con la función "Generate TFI Mesh". (Fig. 4.3)

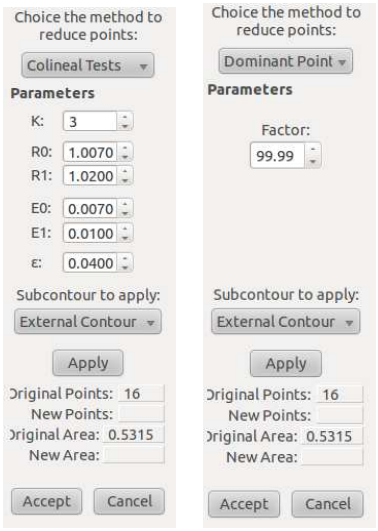


(a) Contorno sin agujeros (b) Contorno con agujeros

Figura 4.3: Controles de la herramienta de suavizamiento

Reducción de puntos: esta función proporciona al usuario dos métodos para al reducción de puntos. El propósito de estos métodos es devolver un contorno que tenga una cantidad menor de puntos sin perder mucho su forma, el primero es una prueba de colinealidad [8] donde se define una banda alrededor de un subgrupo de puntos (determinado por K) para decidir si los puntos

forman una curva muy parecida a un segmento (de ahí los otros parámetros). El otro método se denomina puntos dominantes donde la idea es fijarse en el cambio de ángulo para elegir puntos que se consideran relevantes al definir la forma del contorno y eliminar los demás, sólo usa un parámetro llamado Factor. Los parámetros por default son buenos en muchos casos (para ambos métodos), y es cuestión de probar cuál es la configuración que más conviene en dependencia del problema particular que se quiera resolver. (Fig. 4.4)



(a) Prueba de colinealidad

(b) Puntos Dominantes

Figura 4.4: Controles de la herramienta de reducción de puntos

4.1.2. Herramientas para Mallas Estructuradas

Generación de malla por TFI: en esta función se deben definir el tamaño de la malla inicial (en general esta malla contiene celdas no convexas), si el contorno no tiene agujeros basta con dar sólo dos números, el asociado a la "altura": m y el asociado a la "anchura": n(Fig. 4.6). Si tiene agujeros se debe especificar la cantidad de puntos que se desea tener como se ve en la figura (m_1 siempre se refiere a la parte de adelante al primer punto especial y m_2 a la parte de atrás [adelante y atrás según el recorrido de contorno de forma contraria a las manecillas del reloj]) y nos dará una matriz de tamaño $(m1 + m2) \times (l_{n+1} + \sum_{i=1}^{n}(l_i + h_i))$ con n la cantidad de agujeros (Fig. 4.5).

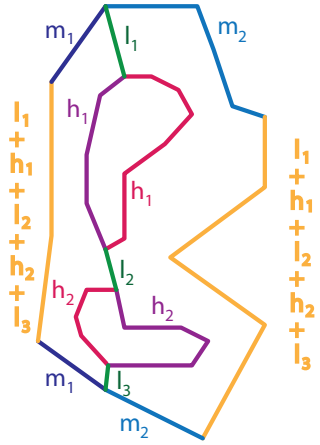
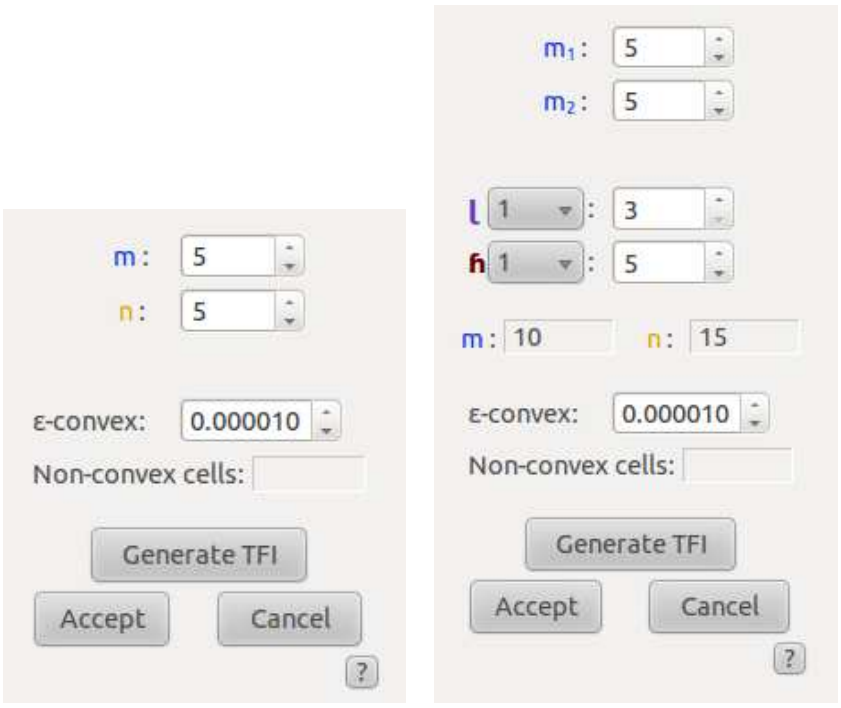


Figura 4.5: Notación del sistema UNAMalla para generar la malla inicial



(a) Contorno sin agujeros (b) Contorno con agujeros

Figura 4.6: Controles de la herramienta de generación de malla inicial

Generación de malla TFI con suavizamiento spline: esta función solo esta activa para los contornos sin agujeros, y utiliza la parametrización a longitud de arco(dada la cantidad de puntos que pidas) y el suavizamiento para con esta generar la malla por TFI. (Fig. 4.3a)

Puntos fijos y celdas inactivas: esta utilidad permite asignar una colección de puntos como fijos dentro de la optimización (esto hace que el problema de optimización sea un problema restringido), además de poder crear conjuntos de puntos conexos que no se vean en pantalla a modo de agujero; este fue el primer acercamiento hacia manejar regiones con agujeros en el sistema UNAMalla 4.0. Esta técnica sigue presente pues es la que se utiliza para fijar las fronteras de los agujeros, pero si se hace algún cambio dentro del sistema deja de manejarse el formato de agujero descrito en el capítulo 2, y cambiar por uno que especifica los puntos que son agujeros por medio de una matriz. Si se selecciona sólo un punto te permite cambiar su posición. También permite hacer una malla a partir de un rectángulo seleccionado dentro de la malla virtual (submalla).

Para seleccionar los puntos se muestra en otra pantalla la representación de la malla de la región como malla del cuadrado unitario, y sobre esta se seleccionan los puntos. (Fig. 4.7)

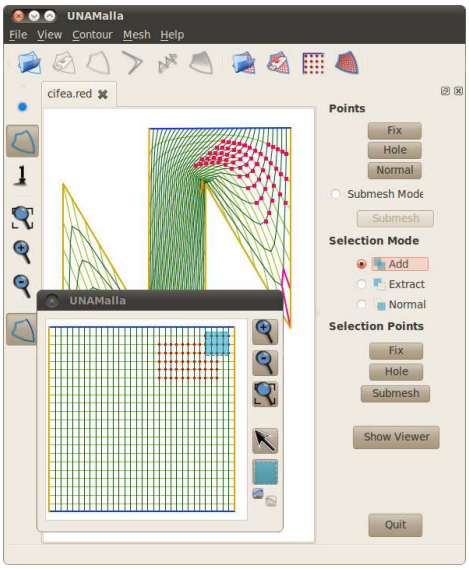


Figura 4.7: Controles para definir puntos fijos y celdas activas/inactivas

Optimización de la malla inicial: esta es la función principal del sistema UNAMalla. Permite elegir alguno de los funcionales del sistema y configurar el valor σ que es el parámetro para regular la combinación de los funcionales, el valor de la ϵ-convexidad que se debe tomar en cuenta, y el método para solucionar el problema de optimización. Por el momento sólo contamos con dos: *TRON-B* (Trust Region Newton Bounded) y *L-BFGS-B* (Limited memory-Broyden–Fletcher–Goldfarb–Shanno-Bounded) [18]. Los funcionales que se utilizan ya fueron mencionados en el capítulo 1. Por default se utiliza el funcional S_ω combinado con área-ortogonalidad, σ a 0.5 y ϵ-convexidad de 1×10^{-5}. La función provee opciones de visualización

del proceso de optimización:

***Paso a paso*:** visualiza paso a paso la animación de la homotopía.

***Primero y último*:** no se ve la animación y se espera a terminar para mostrar el resultado final.

***Pasos definidos*:** se espera una cantidad de pasos definida por el usuario y dibuja un paso.

Todas estas opciones se pueden cambiar en cualquier momento mientras se hace la optimización. El propósito de estas opciones es evitar el procesamiento que utiliza en dibujar y dedicar lo menos posible el CPU en el dibujo (este dibujo no esta acelerado por GPU). (Fig. 4.8)

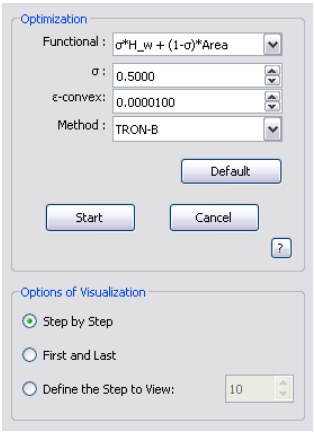


Figura 4.8: Controles para definir el proceso de la optimización de la malla

4.2. Implementación

El programa fue escrito en C++ utilizando las bibliotecas de Qt 4.7 de Nokia [19], y se utilizan rutinas escritas en fortran 90 para la optimización principalmente. La figura 4.9 describe el esquema de las clases que se utilizan dentro del sistema (parecido a un diagrama de clases).

Para el programa se han creado varias clases, por lo que es necesario explicar la función de cada una dentro del sistema, algunas son más sencillas de describir que otras, por lo que dedicaré una sección a explicar las que no son sencillas y a continuación explicaré las sencillas. Daremos por sentado el conocimiento del lector sobre la biblioteca Qt. También esperamos un conocimiento fluido de en lenguaje C++ para entender el código del programa y manejo del concepto de la llamada "Programación Orientada a Objetos" (POO).

4.2.1. Interfaz

La interfaz gráfica está divida en varias secciones (Fig. 4.10) la idea es tener acceso a todas las funciones desde ella, en la parte llamada Barra de Menú se encuentran las funciones del sistema, mientras que en la barra de herramientas lateral se encuentran funciones auxiliares sobre la visualización del contenido, y en la ventana de la izquierda se muestran los controles de las funciones que

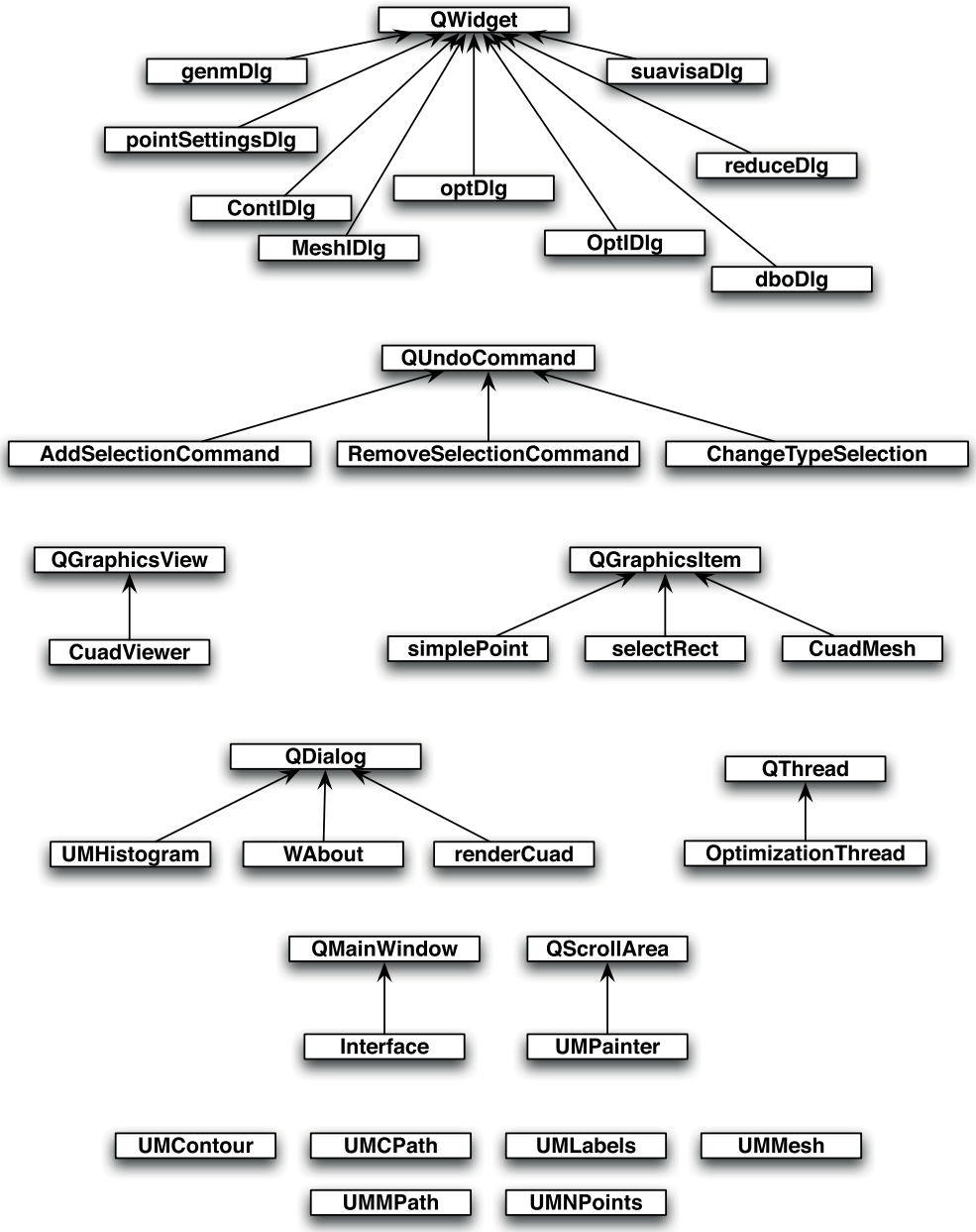


Figura 4.9: Clases del programa UNAMalla 4.5

ofrece el programa, en la parte llamada Visor se dibuja el contenido que se hay abierto (ya sea un

contorno o una malla) y permite abrir varios dentro del mismo programa. Finalmente existe una barra de estado en la que se mandan algunos mensajes sencillos.

Figura 4.10: Distribución Gráfica de la GUI

La clase Interface se encarga de comunicar las funciones con el usuario, y administrar a los controles y menús disponibles en dependencia del contenido que se despliegue, además de guardar la función que se esta utilizando en cada contenido para desplegar en caso de cambiar de ventana. En la figura 4.11 se muestran las clases que pertenecen a la clase Interface. Los diagramas de flujo 4.1 muestran como se abren los contornos (donde verificar puntos es similar al diagrama de flujo 3.1 del capítulo), el algoritmo para abrir las mallas en el diagrama de flujo 4.2 , en el diagrama de flujo 4.3 muestra como se administra el cambio de ventana. A continuación se describen las clases que intervienen dentro de la interfaz.

ContIDlg: despliega la información del contorno como su área, el número de puntos en cada frontera (si es que tiene las fronteras definidas), el número de puntos de todo el contorno y avisos sobre alguna propiedad especial que presente el contorno, además de un botón que manda dicho contorno al ContourCreator para ser editado.

dboDlg: muestra los controles para definir las cuatro fronteras del contorno; los controles cambian de rango en dependencia de los demás. Cuando el contorno tiene agujeros hay restricciones para asegurarse que estén los puntos especiales del contorno exterior en fronteras opuestas (y en esta versión sólo pueden estar en la frontera azul). En el diagrama de flujo 4.4 se muestra como se hace la defincón de la fronteras cuando se tienen agujeros.

reduceDlg: muestra los controles para definir la reducción de puntos de un contorno, el método que se desea usar, y en caso de tener agujeros pide el subcontorno donde se desea aplicar el método, además de llamar a la función correspondiente del método y aplicarlo; los métodos

disponibles son puntos dominantes y prueba de colinealidad [8]. Hace una copia del contorno para trabajar con ella.

suavisaDlg: muestra los controles para la generación de un contorno suavizado por medio de splines cónicos, y también crea una malla utilizando la parametrización del contorno suavizado; en caso de tener agujeros no despliega el menú para crear mallas y pregunta el subcontorno con el cuál se va a trabajar. Hace una copia del contorno para trabajar con ella.

genmDlg: en esta ventana se muestran los controles para generar la malla inicial, aparte de tener los algoritmos que generan la partición y unión de la malla inicial para los contornos con agujeros, y hace llamada a las rutinas de fortran para los contornos sin agujeros.

MeshIDlg: despliega la información del contorno y de la malla (dimensión, área mínima de las celdas y área máxima de las celdas, número de celdas no convexas, algún mensaje informativo sobre la malla) y llama una ventana para ver un histograma de área de las celdas de la malla.

pointSettingsDlg: esta ventana provee los controles para definir y editar los puntos fijos y/o los falsos agujeros, además de tener el control sobre la ventana que muestra la malla virtual (renderCuad).

optDlg: esta ventana despliega los controles para configurar el optimizador además de elegir entre el TRON y L-BFGS-B, y configurar la visualización. Este llama a OptIDlg que es el ejecuta el optimizador.

WAbout: esta es la ventana del "acerca de.." la cuál muestra información sobre el sistema y sus autores.

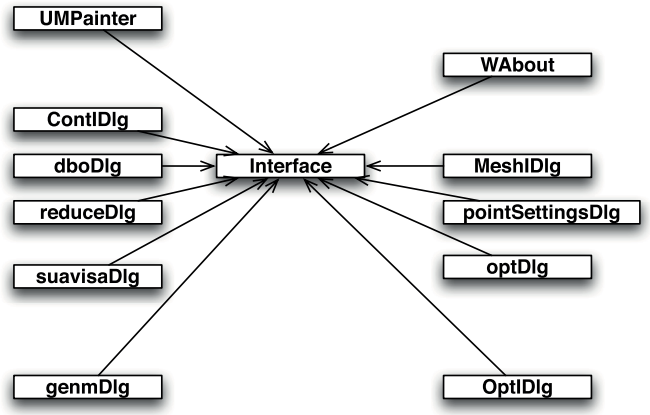


Figura 4.11: Clases contenidas en la clase Interface

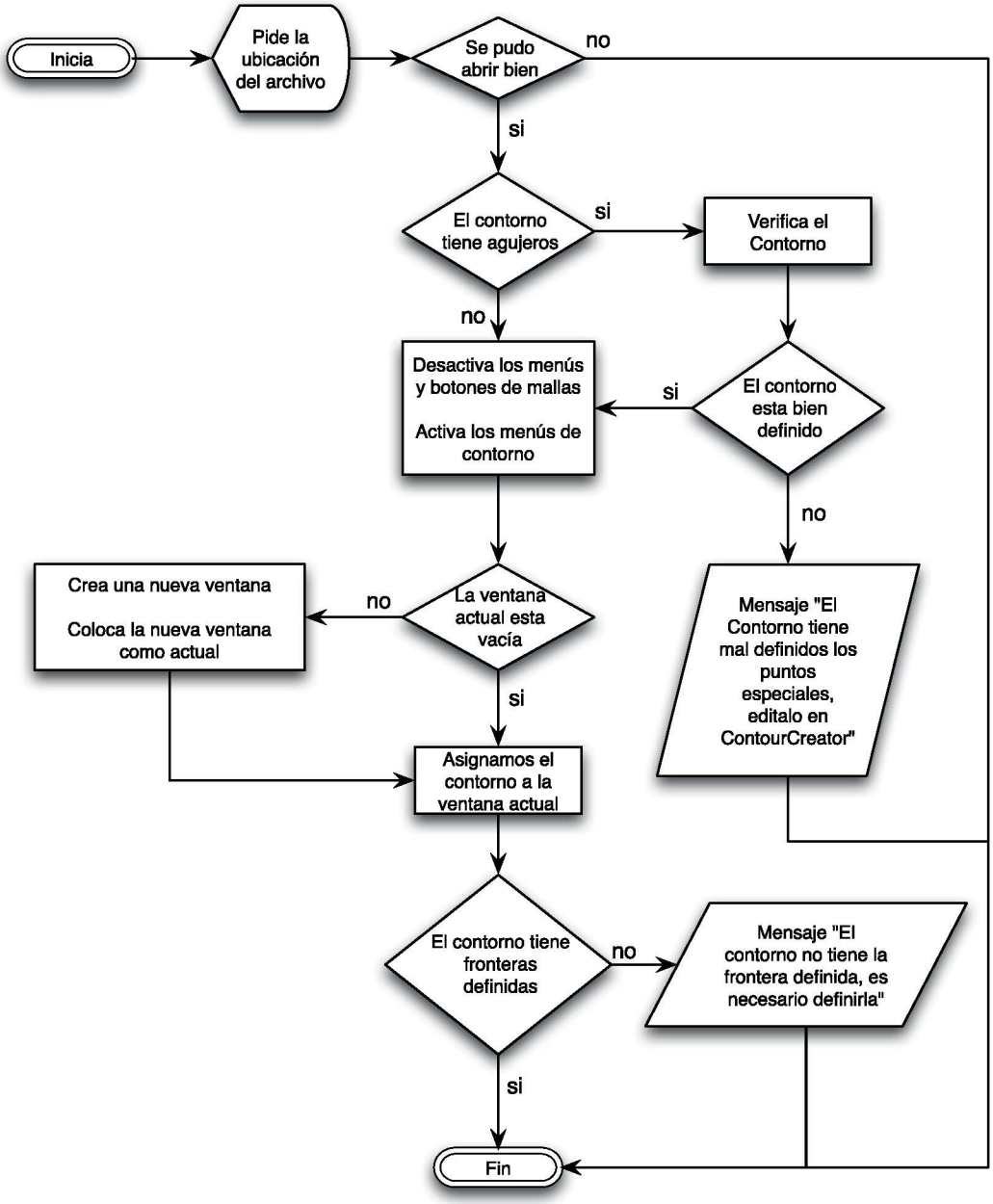


Diagrama de Flujo 4.1: Función *openc*

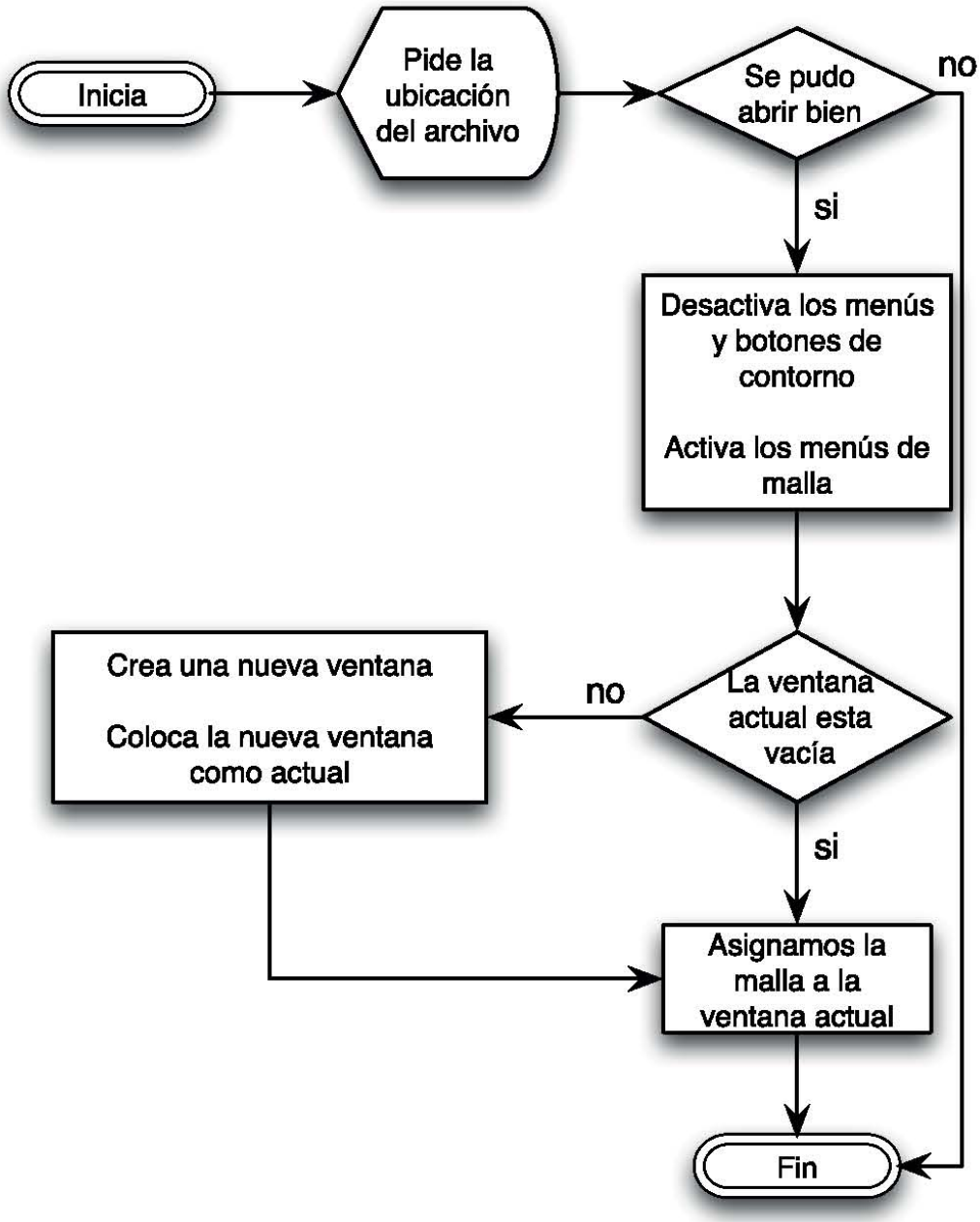


Diagrama de Flujo 4.2: Función *openm*

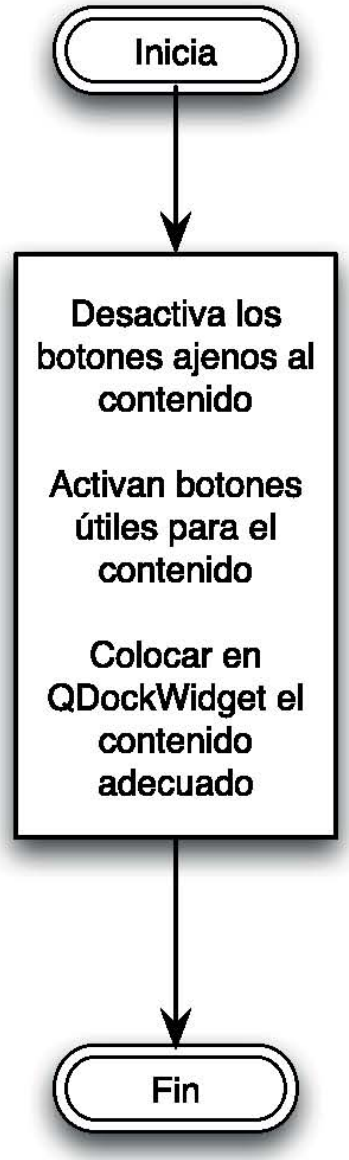


Diagrama de Flujo 4.3: Función *closetab* y *changetab*

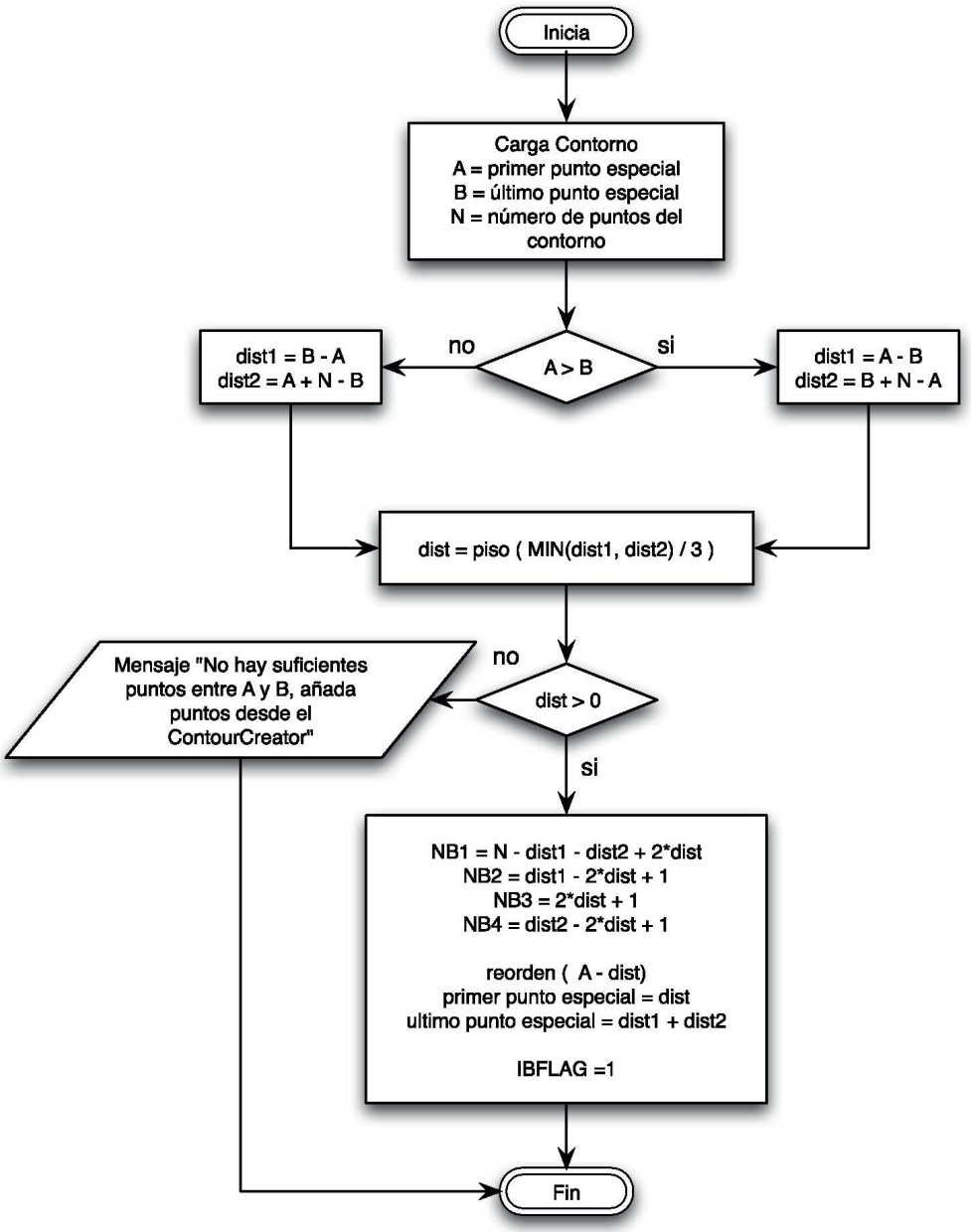


Diagrama de Flujo 4.4: Función *boundariesHoles*

4.2.2. Manejo Gráfico

El manejo gráfico del sistema fue hecho a partir de la herramienta de Qt4 llamada QPainter, este es un sistema muy básico de dibujo; sin embargo, es la forma más rápida de dibujo que ofrece Qt y es útil cuando no es necesario interactuar con los objetos gráficos. Esta se encuentra dentro de la clase **UMPainter** que se encarga de pintar el contenido (y de la configuración del dibujado) y guardarlo en una imagen para ya no volver a calcular todo el dibujo cada vez que se cambie de ventana.

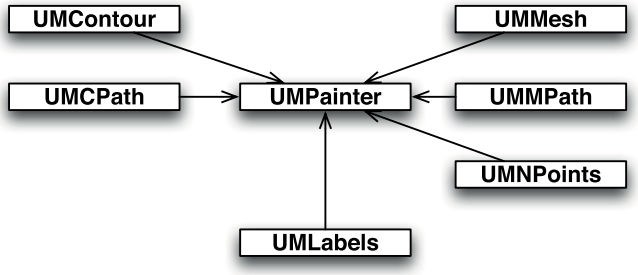


Figura 4.12: Clases contenidas en la clase UMPainter

La siguientes clases tienen que ver con la visualización de cada contenido.

UMContour: esta clase se encarga de guardar la estructura del contorno y asociarle un QPainterPath, actualizar la posición de los puntos (gráficos), calcular el rectángulo mínimo que lo contiene, limpiar los puntos y segmentos, dar la lista de puntos y decir si tiene o no contenido.

UMCPath: esta clase crea los QPainterPath basándose en las posiciones de los vértices del contorno y se encarga de dibujarlo (para esto tiene asociado un grosor, color, y relleno).

UMMesh: se encarga de administrar la estructura malla (para el optimizador, los puntos fijos) también de asociarle un QPainterPath que será su representación gráfica, además de crear una estructura de matriz de apuntadores que permite ver la estructura RED como una matriz, calcula dada una posición en la matriz cuál es la posición asociada en el arreglo RED (haciendo uso de la matriz de apuntadores "meshview").

UMMPath: esta clase crea los QPainterPath basándose en las posiciones que da la matriz de apuntadores "meshview" y se encarga de dibujar (para esto tiene asociado un grosor y color), también se encarga de dibujar las mallas sin agujeros(evitando pintar segmentos dentro del agujero).

Como se ve en la figura 4.9 se utilizan las clases QGraphicsItem, QGraphicsView y QGraphicsScene (esta última no aparece en el diagrama de clases, pero si es usada) son las partes que conforman el esquema gráfico de Qt llamado "Graphic View Framework" este un sistema de visualización muy completo que ofrece herramientas para facilitar el trabajo con elementos gráficos en 2D, se utiliza dentro del UNAMalla en la clase **CuadViewer** la cuál se encarga de administrar el dibujado y manejo de la malla virtual. Este esquema permite tener la función para detectar a qué objeto

gráfico se le ha hecho clic y dar facilidades para seleccionar uno o más objetos en pantalla, lo que hace a este esquema idóneo para hacer selecciones complicadas de nodos internos de la malla. Los QGraphicsItem son los elementos gráficos que aparecen en pantalla y a los que se podrá manejar de manera fácil con este esquema. El QGraphicsScene es la clase que se encarga de administrarlos en la pantalla, mientras que QGraphicsView se encarga de pintarlo en pantalla, las siguientes clases se utilizan en este esquema (sólo para el manejo de la malla virtual para la selección de puntos fijos y agujeros falsos).

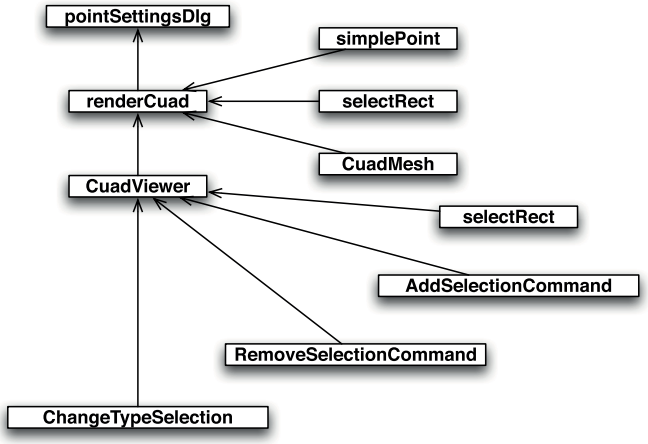


Figura 4.13: Clases relacionadas con el manejo gráfico de los puntos fijos y celdas inactivas

renderCuad: esta es la interfaz entre el CuadView y algunas funciones de visualización (Zoom, área de selección o puntero de selección, y hacer/deshacer), administra la lista de acciones que se han hecho y se encarga del tamaño de la visualización.

simplePoint: dibuja un pequeño cuadrado rojo en representación de un punto, además de estar oculto hasta que es seleccionado, y guardar la posición del punto al que esta asociado dentro de la malla virtual (coordenadas (i, j)).

selectRect: dibuja un cuadrado semiopaco que indica el área donde se desea seleccionar puntos.

CuadMesh: dibuja la malla virtual propiamente.

Las siguientes clases ejecutan las acciones de hacer y deshacer para poder regresar a una configuración antigua. Esto lo consideramos importante ya que es difícil en algunos casos elegir el área correcta que uno desea.

AddSelectionCommand: dada una región con área se seleccionan los elementos que estén contenidos en dicha región y se guarda la región antigua (en caso de deshacer se seleccionan los elementos que están dentro de la antigua).

RemoveSelectionCommand: se quita la lista de selección de la escena y se guarda la región antigua de selección y en caso de deshacer se seleccionan los puntos dentro del antigua área de selección.

ChangeTypeSelection: simplemente guarda el actual tipo de selección de áreas (union, intersección, resta) y el nuevo, para cambiar el tipo de selección se asignan como el estado "actual" (en caso de *redo* asigna el nuevo y en caso de *undo* asigna el antiguo).

Para el dibujo del Histograma (al cuál se llama en la información de la malla) se hace tomando en cuenta el tamaño de la letra para dibujar los intervalos en donde se encuentran las áreas y el tamaño de la ventana; es decir, que mientras más grande sea la ventana más intervalos habrán y por lo tanto más fino es el histograma, también se dibuja lineas que representan los ejes. (para diferenciar las celdas no convexas). El esquema gráfico que se utiliza es el QPainter y también se dibuja directamente en la ventana, además de permitir guardar como imagen el resultado.

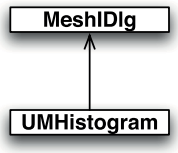


Figura 4.14: Clases contenidas en la clase meshDlg

4.2.3. Optimizador

La función principal del sistema UNAMalla es encontrar una malla a partir de un problema de optimización, esto se hace por medio de una rutina que modifica paso a paso (paso del optimizador) (Diagrama de Flujo 4.5). Esto se realiza por medio de una técnica de programación llamada "Threads" que permite ver al optimizador como un proceso por separado del sistema y con esto el procesador le dedique un tiempo a la parte gráfica y no sólo al optimizador; utilizando la librería **QThread** que incluye el Qt 4, además de que en sistemas en paralelo este método es muy efectivo. La clase **OptIDlg** despliega la información del proceso de optimización además de ser el que llama

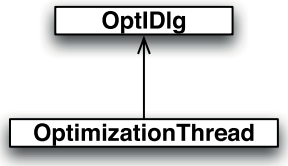


Figura 4.15: Clases contenidas en la clase OptIDlg

y controla al optimizador, también sirve de interfaz entre el optimizador y la ventana donde se despliega el dibujo de la malla. La clase **OptimizationThread** contiene el algoritmo para cada solver (TRON, L-BFGS-B), y manda un aviso para comunicarse con la ventana de información (OptIDlg) y desplegar los datos a cada paso además de avisar que ya terminó un paso.

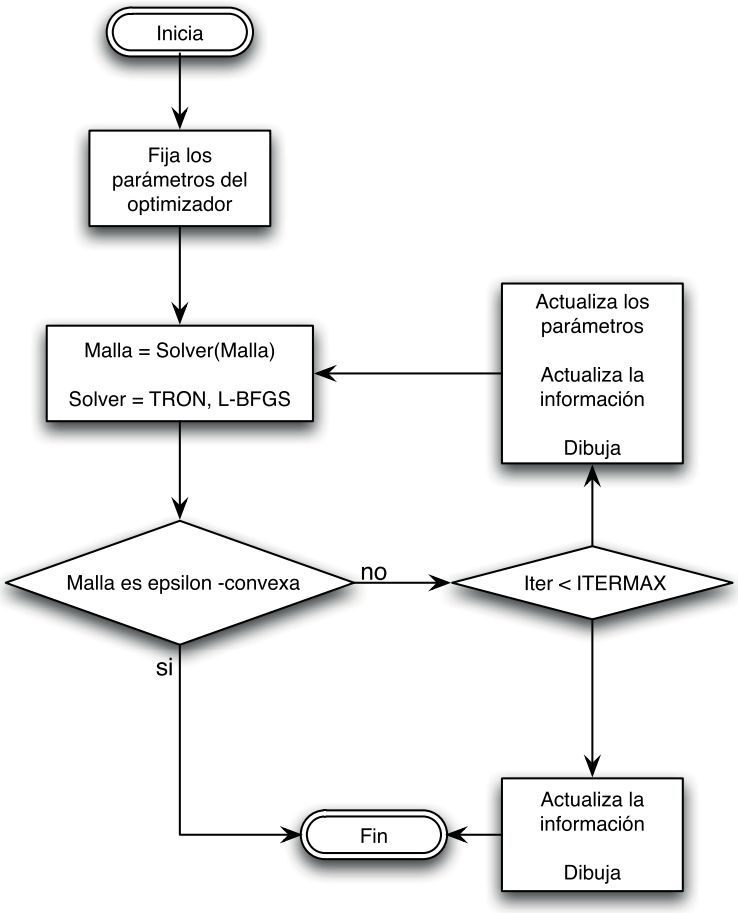


Diagrama de Flujo 4.5: Proceso de Optimización

Capítulo 5

Conclusiones

Se ha propuesto un método semi-automático para generar mallas bloque-estructuradas para regiones con agujeros sencillas de manejar por ser estructurada la malla y ser muy sencillo identificar las fronteras internas.

Se ha ampliado el dominio de regiones para los métodos variacionales discretos desarrollados por el grupo UNAMALLA, permitiendo usar regiones con n agujeros. Lo que es importante para resolver adecuadamente algunas ecuaciones diferenciales parciales sobre estas regiones.

Se ha hecho una revisión y actualización al sistema UNAMalla, corrigiendo algunos fallos de la versión 4.0 e implementando el método que hemos desarrollado para regiones con n-agujeros. UNAMalla 4.5 es un sistema multiplataforma que permite crear regiones con agujeros y generar una malla sobre estas regiones de una manera sencilla. Incluye dos programas que proveen de las siguientes funciones (Capítulos 3 y 4):

ContourCreator 1.5

- Agregar Puntos
- Quitar Puntos
- Mover Puntos
- Añadir agujeros
- Agregar punto medio
- Repoblar el contorno
- Definición de los puntos especiales

UNAMalla 4.5

- Definición de las fronteras
- Suavizamiento de contorno
- Reducción de puntos

- Generación de malla por TFI y con suavizamiento spline cónico
- Puntos fijos y celdas inactivas
- Optimización de malla (Incorpora los métodos TRON-B, L-BFGS-B)
    - Incluye los funcionales "cuasi-ármonico S_ω" y "área H_ω", combinados con funcionales de área. ortogonalidad y longitud.

Más información del sistema UNAMalla en la página web:
`http://www.matematicas.unam.mx/unamalla`.

Trabajo a futuro

El usuario es quien decide la manera de dividir la región de estudio, lo ideal seria proponer de forma automática una elección.

Cada subfrontera opuesta esta unida a la otra por medio de un segmento que forma parte del camino que divide la región (Fig. 2.12), una propuesta es cambiar dicho segmento por una curva poligonal que pueda ser fijada, con la condición de estar contenida en la región y no intersectar la frontera de la región con agujeros. Esto permitiría especificar la forma de un flujo particular en la malla.

El método que hemos propuesto genera mallas que no necesariamente son adecuadas para todos los casos; el cual provoca que las curvas que forman la malla "sigan el camino" lo que limita algunas características de calidad de las celdas. Por esta razón hay que investigar e implementar métodos alternativos para dividir la malla por bloques que contengan los agujeros, y permitir una distribución de los agujeros en la estructura matricial que aproveche toda la estructura y no sólo una parte (pues nuestro método sólo utiliza una columna de la estructura matricial).

El sistema UNAMalla esta pensado para trabajar con regiones conexas; sin embargo, no cuenta con mecanismos para verificar la contención correcta de los subcontornos, ni que los subcontornos sean ajenos entre si; requiere de algoritmos eficientes que verifiquen las autointersecciones de los subcontornos entre si, la conexidad de la región, y la contención de los agujeros en el contorno exterior.

Mallas en 3D

En muchas aplicaciones la región de estudio se encuentra en un espacio de tres dimensiones, la cual es posible modelar por medio de una colección de mallas planas empatadas, auxiliándose de una función que les asigne una coordenada Z para representar adecuadamente la región. A continuación se muestra un pseudo-código con esta idea:

Input: lista de mallas 2D (dimensión $m \times n$) de tamaño l
Output: Malla en 3D (m3d)
m3d = new double[m][n][l];
for $k \leftarrow 0$ **to** l **do**
    abre la k-malla 2D;
    if *abrió bien la k-malla* **then**
        for $i \leftarrow 0$ **to** m **do**
            for $j \leftarrow 0$ **to** n **do**
                m3D[i][j][k].x = malla2D[k].x;
                m3D[i][j][k].y = malla2D[k].y;
                m3d[i][j][k].z = setZ(malla2D[k].x,malla2D[k].y,k);
            end
        end
    end
end

Esto nos crea un arreglo tridimensional que se puede pensar como un cubo (Fig. 5.1), que nos representa la región en 3D. A continuación hay una muestra de ejemplos creados de esta manera, el primer ejemplo es real, se trata de una llanta para auto "formula 1", los siguientes son ejemplos creados combinando diferentes funciones que asignan el valor en Z y tres regiones simples.

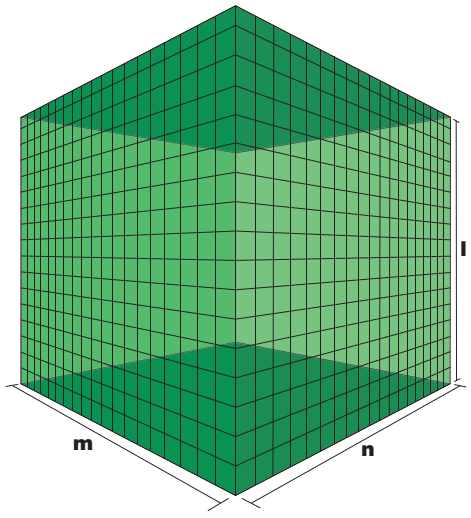
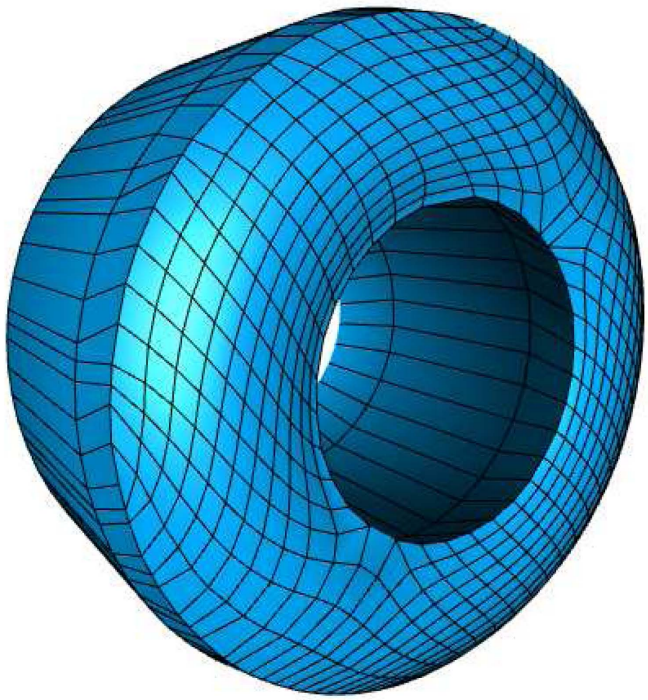
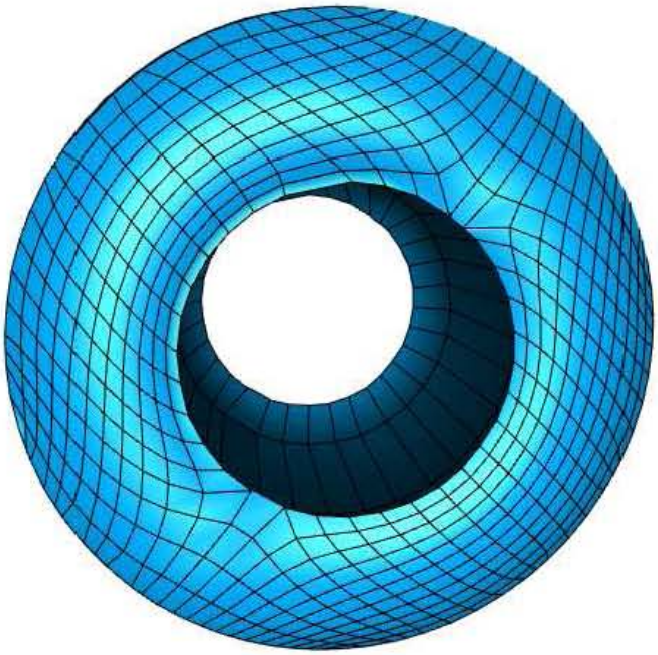
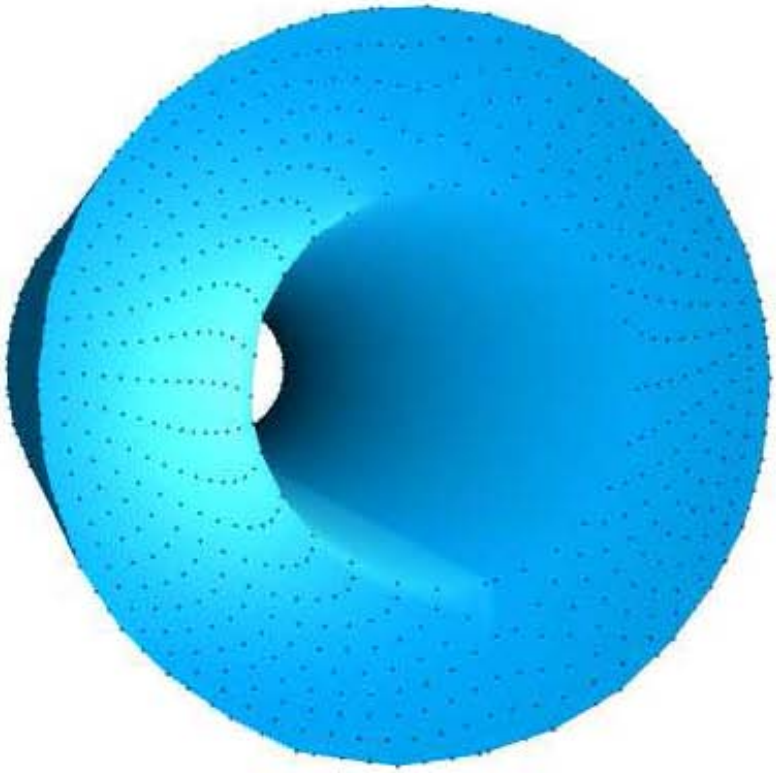
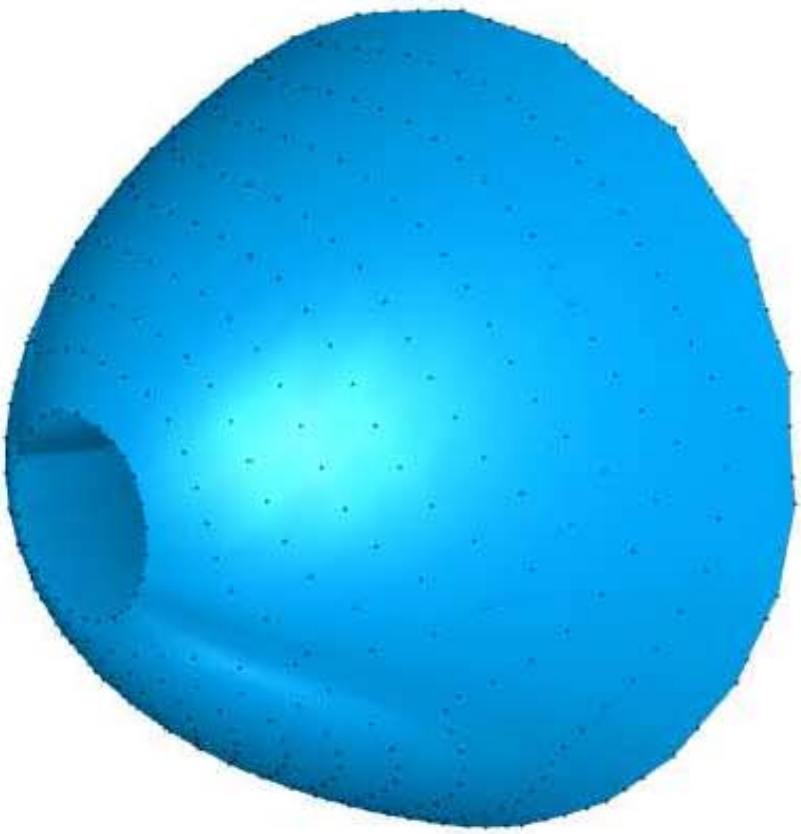


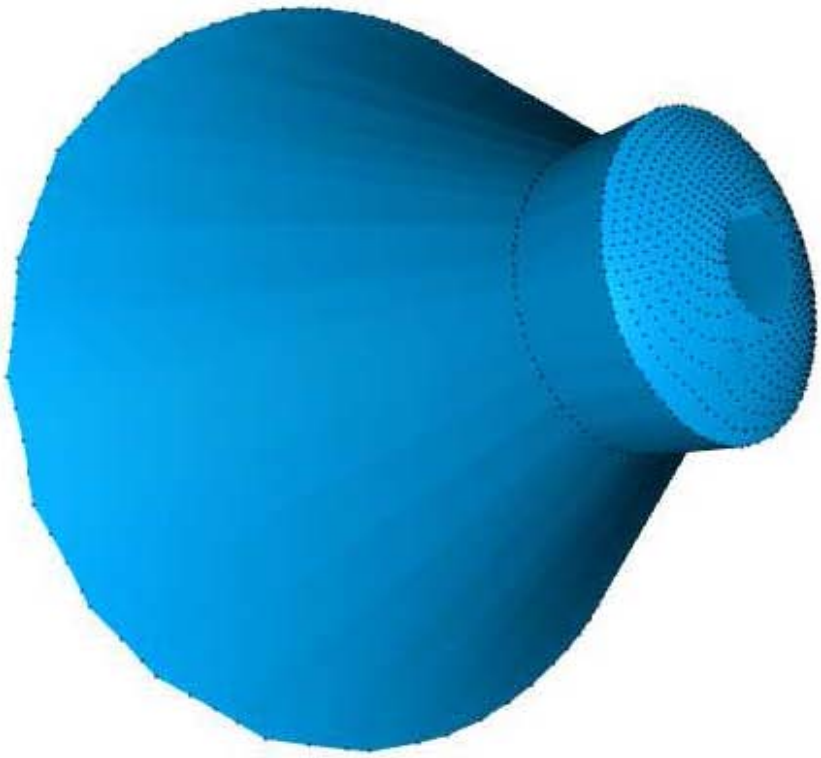
Figura 5.1: Malla Virtual en 3D

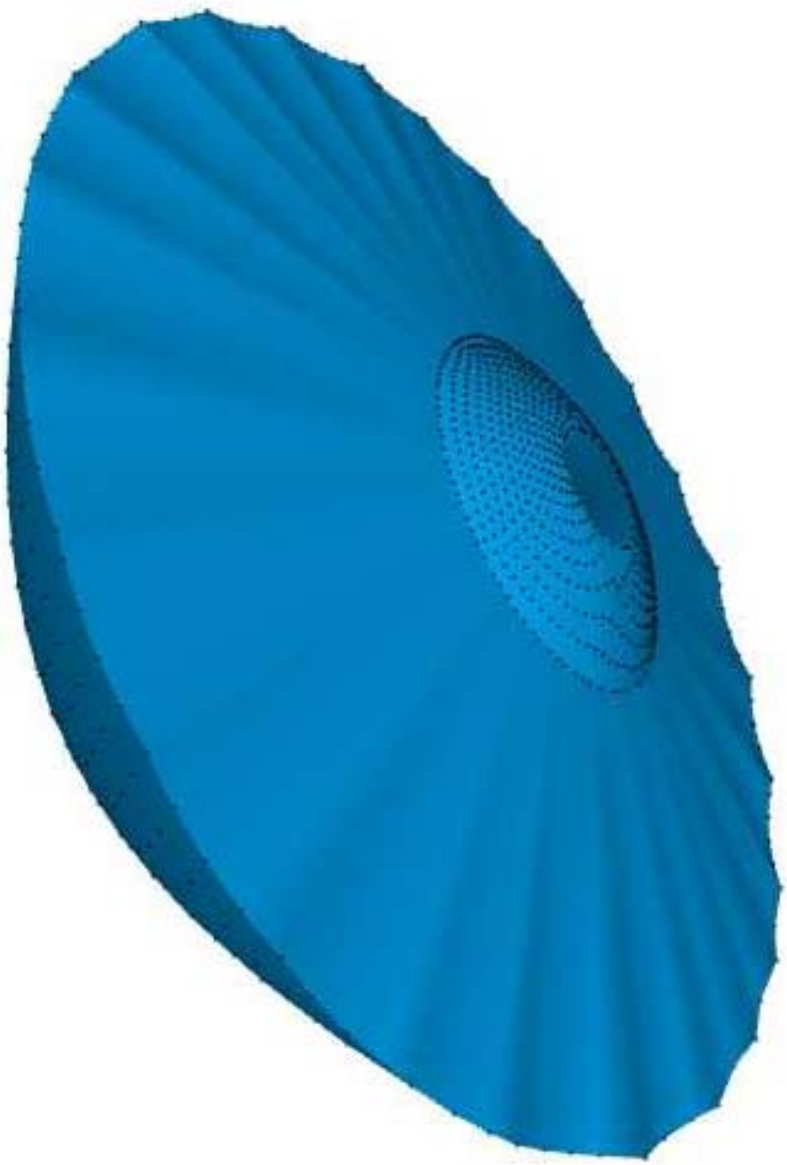












Apéndice A

Galería de Mallas

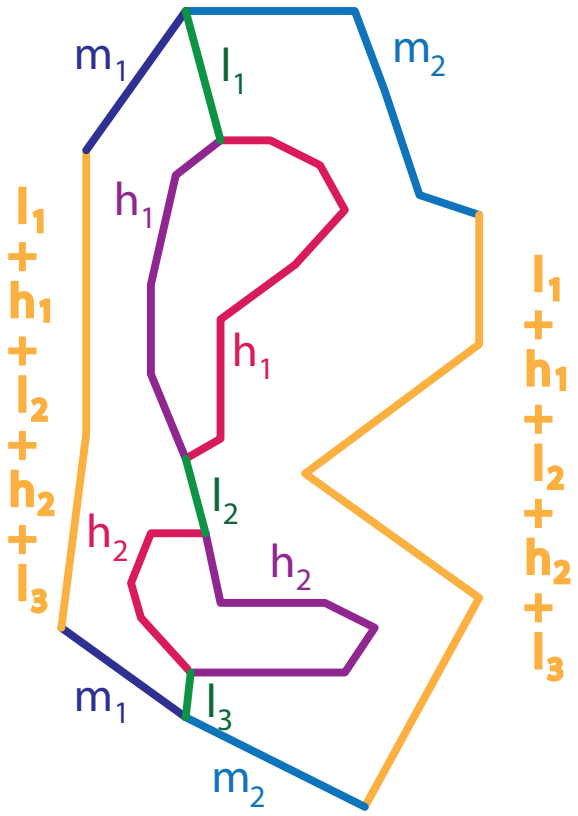


Figura A.1: Notación del sistema UNAMalla para generar la malla inicial

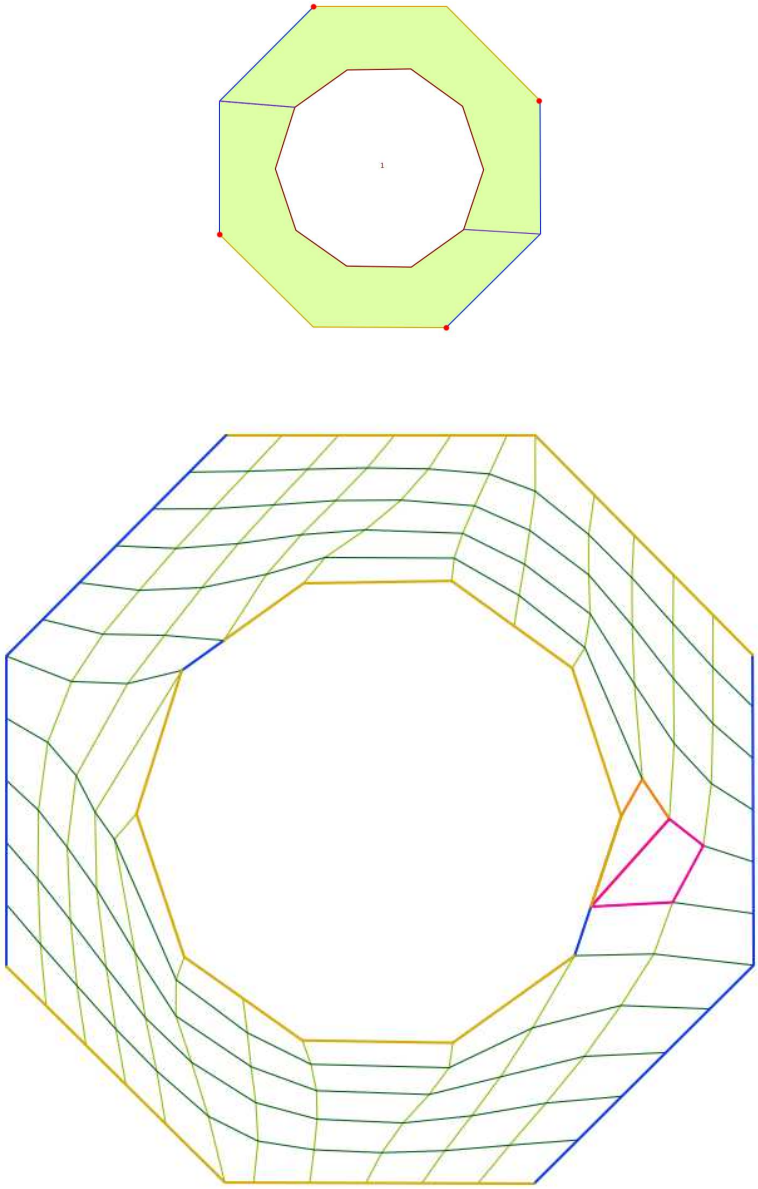


Figura A.2: Dimensión: 12x12 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

$m_1 = 6 \quad m_2 = 6 \quad m = 12$
$l_1 = 4 \quad l_2 = 3$
$h_1 = 7 \quad n = 12$

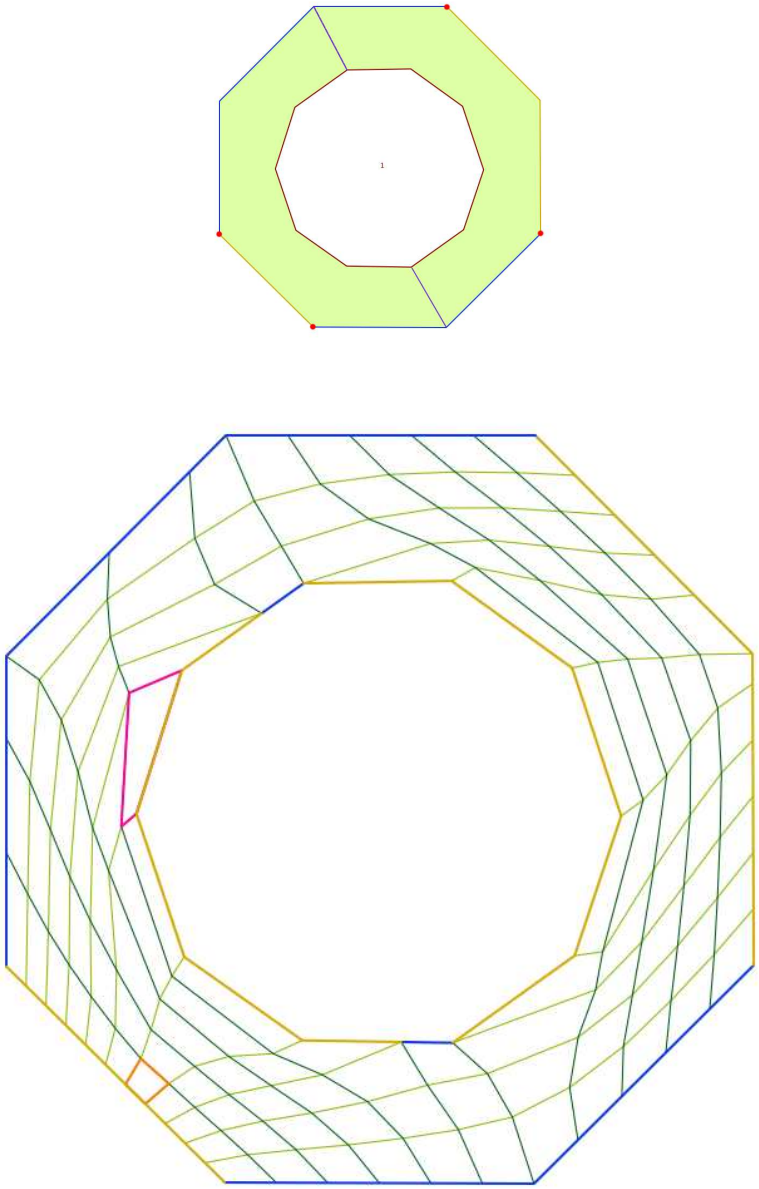


Figura A.3: Dimensión: 12x12 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

$m_1 = 6 \quad m_2 = 6 \quad m = 12$
$l_1 = 4 \quad l_2 = 4$
$h_1 = 6 \quad n = 12$

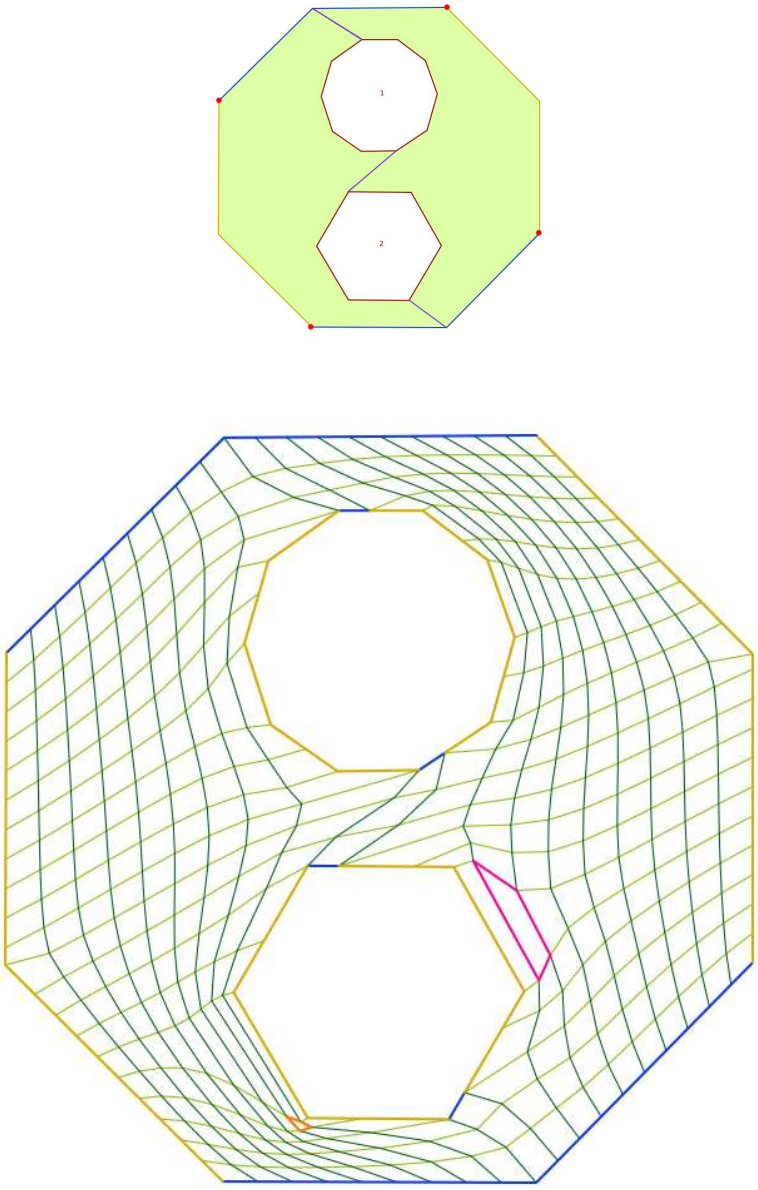


Figura A.4: Dimensión: 20x22 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

$m_1 = 10$ $m_2 = 10$ $m = 20$
$l_1 = 4$ $l_2 = 4$ $l_3 = 4$
$h_1 = 8$ $h_2 = 6$ $n = 22$

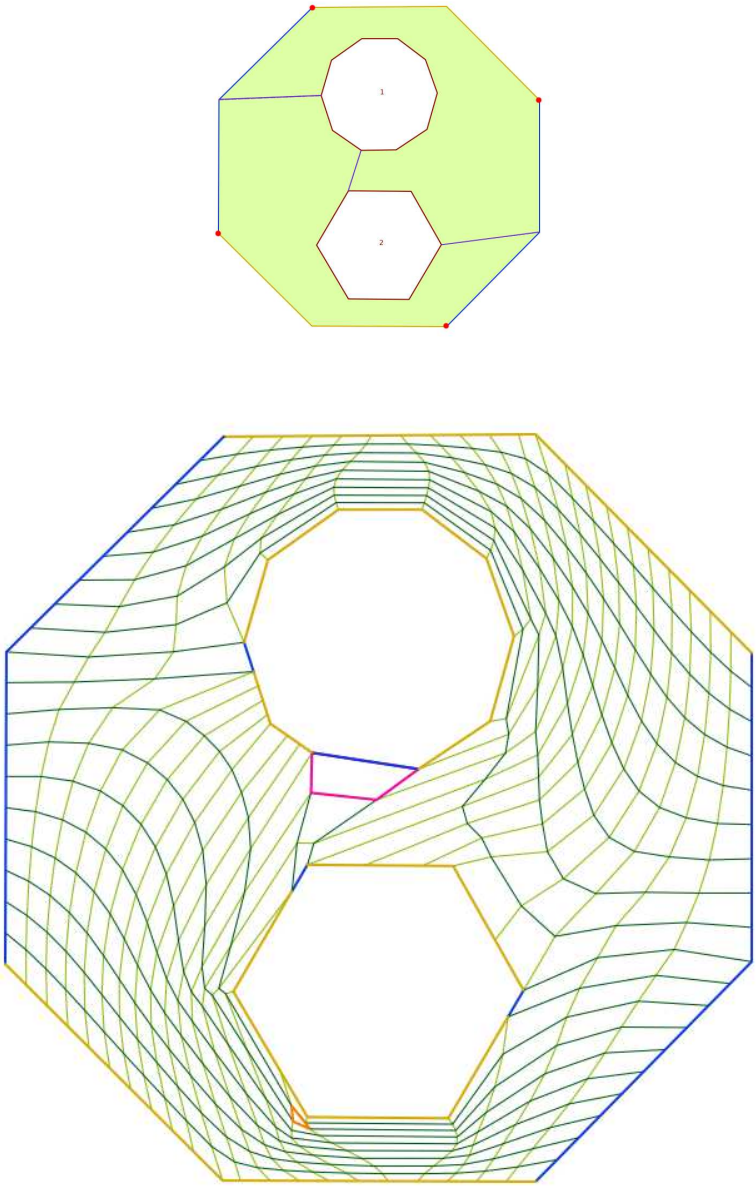


Figura A.5: Dimensión: 20x22 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

$m_1 = 10$	$m_2 = 10$	$m = 20$
$l_1 = 4$	$l_2 = 4$	$l_3 = 4$
$h_1 = 8$	$h_2 = 6$	$n = 22$

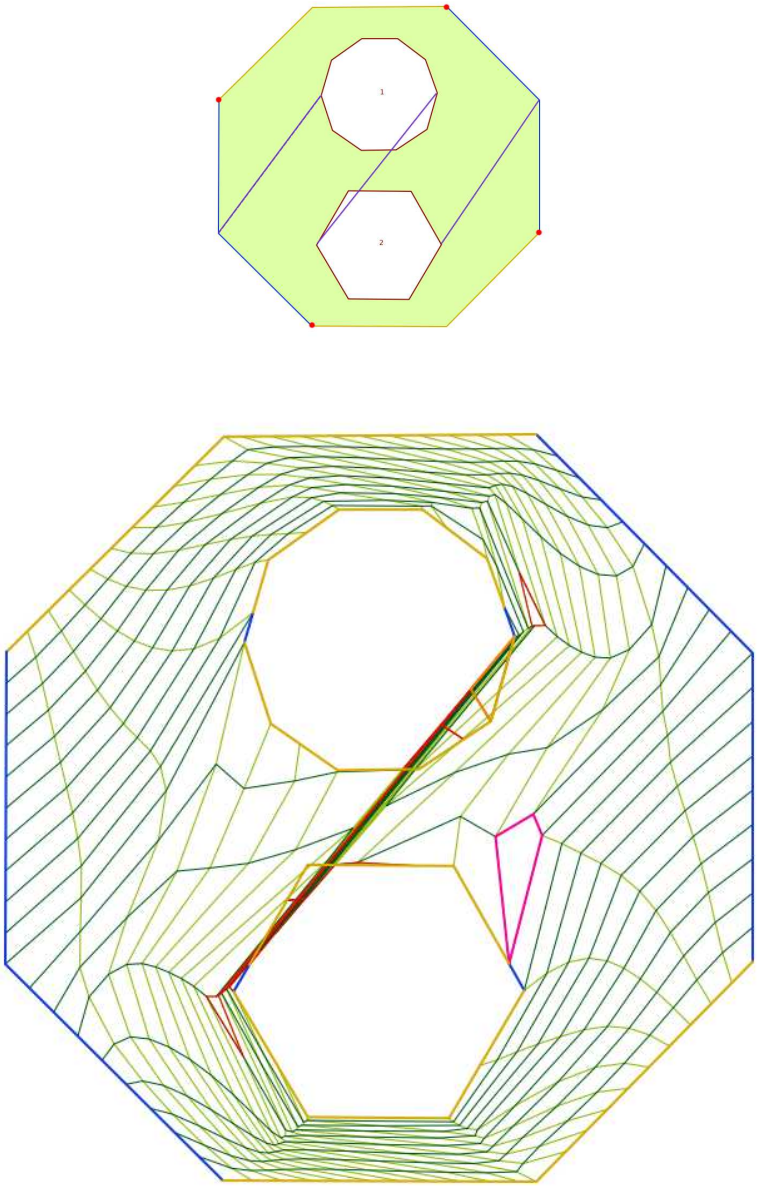


Figura A.6: Dimensión: 20x22 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

$m_1 = 10$	$m_2 = 10$	$m = 20$
$l_1 = 4$	$l_2 = 4$	$l_3 = 4$
$h_1 = 8$	$h_2 = 6$	$n = 22$

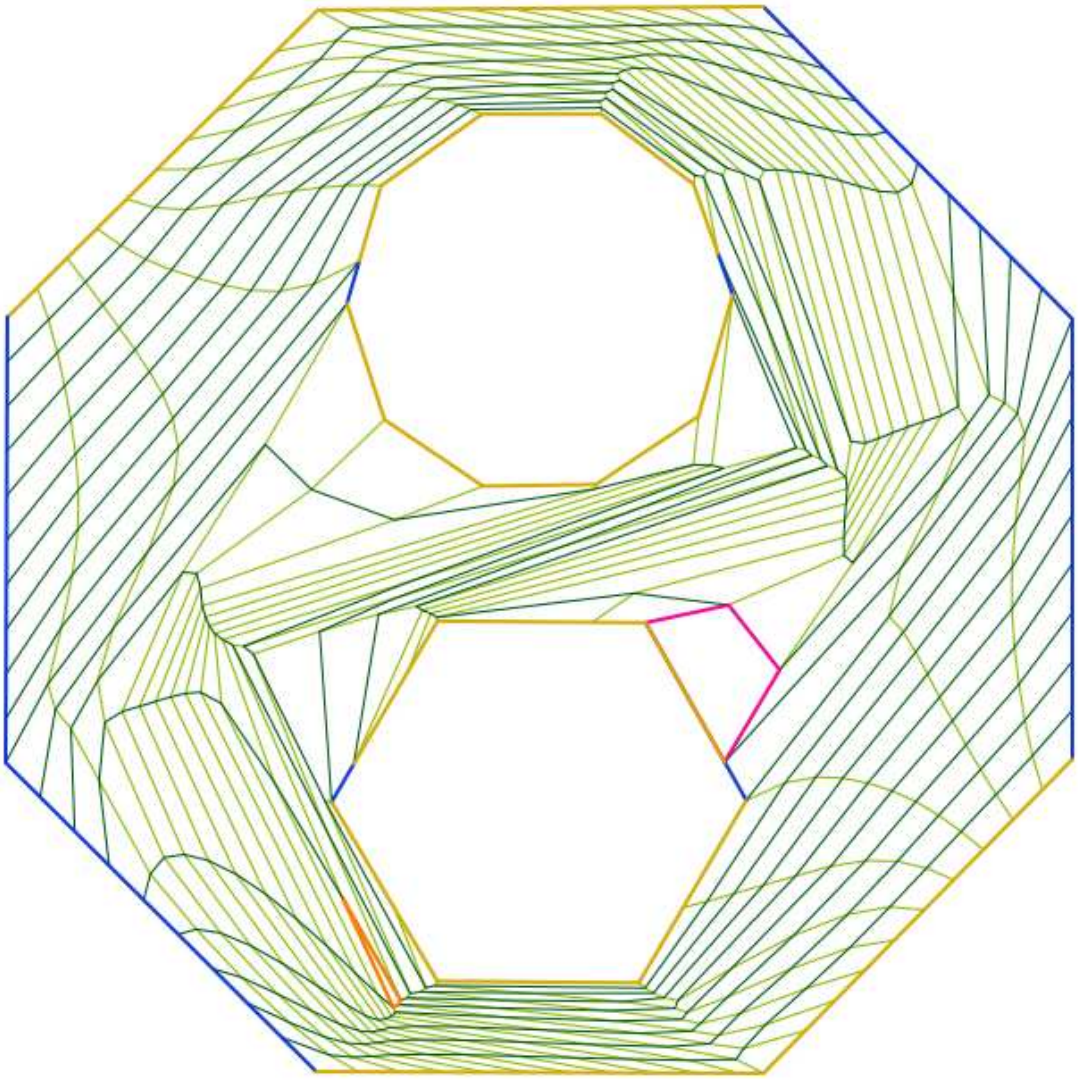
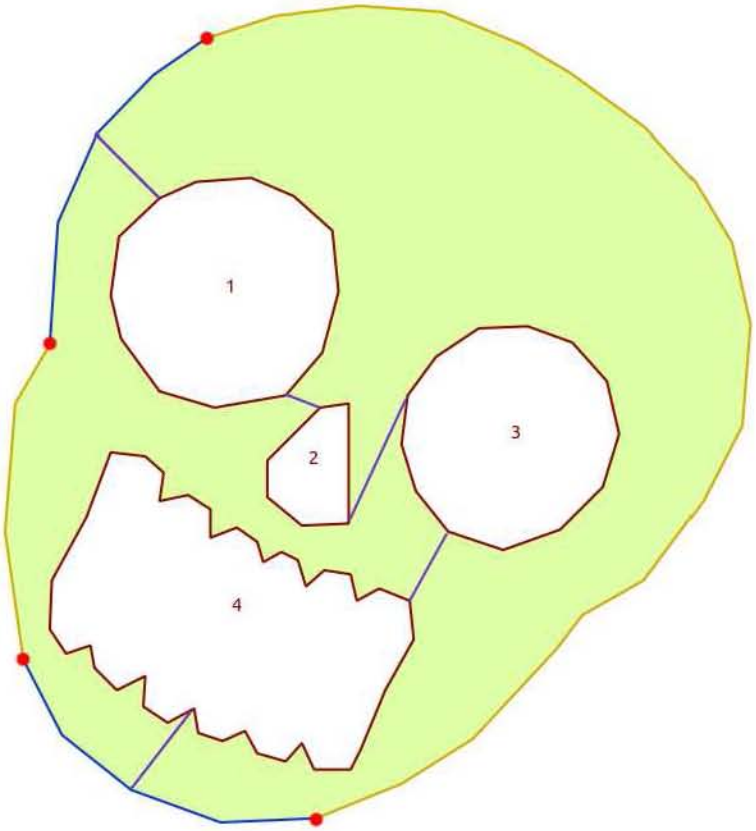


Figura A.7: Dimensión: 20x22 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

$m_1 = 10$ $m_2 = 10$ $m = 20$
$l_1 = 4$ $l_2 = 6$ $l_3 = 4$
$h_1 = 6$ $h_2 = 6$ $n = 22$


1
2
3
4

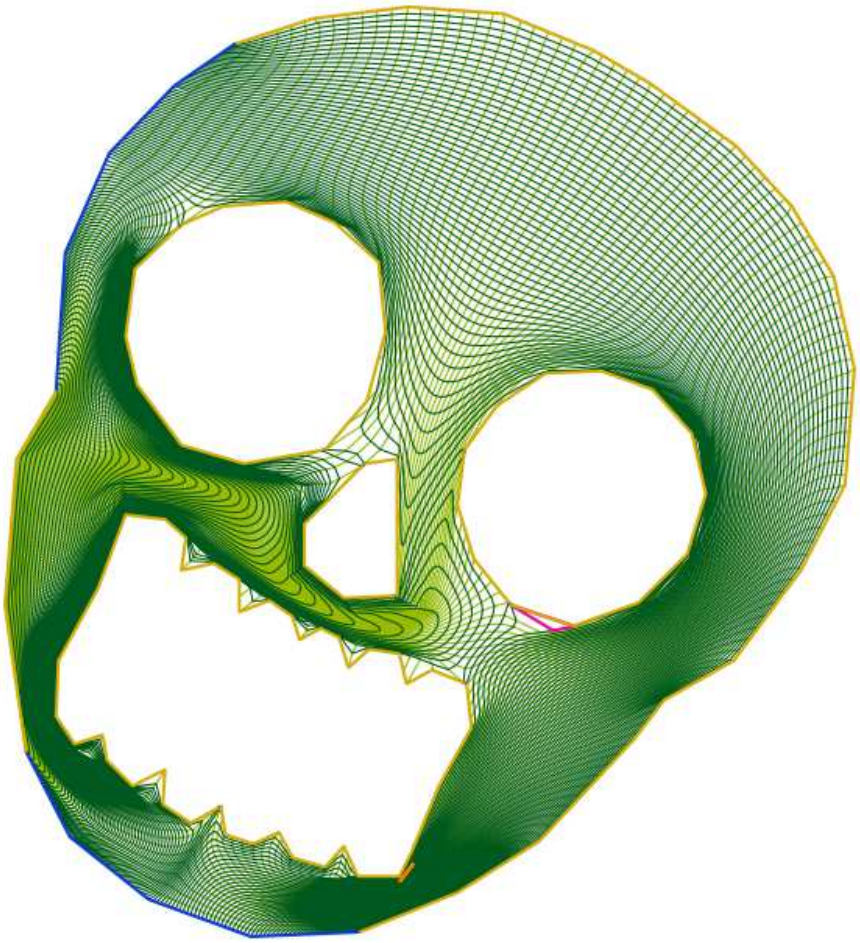


Figura A.8: Dimensión: 110x110 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

$m_1 = 10$ $m_2 = 15$ $m = 110$
$l_1 = 9$ $l_2 = 4$ $l_3 = 15$
$l_4 = 15$ $l_5 = 9$
$h_1 = 13$ $h_2 = 6$
$h_3 = 13$ $h_4 = 39$ $n = 110$

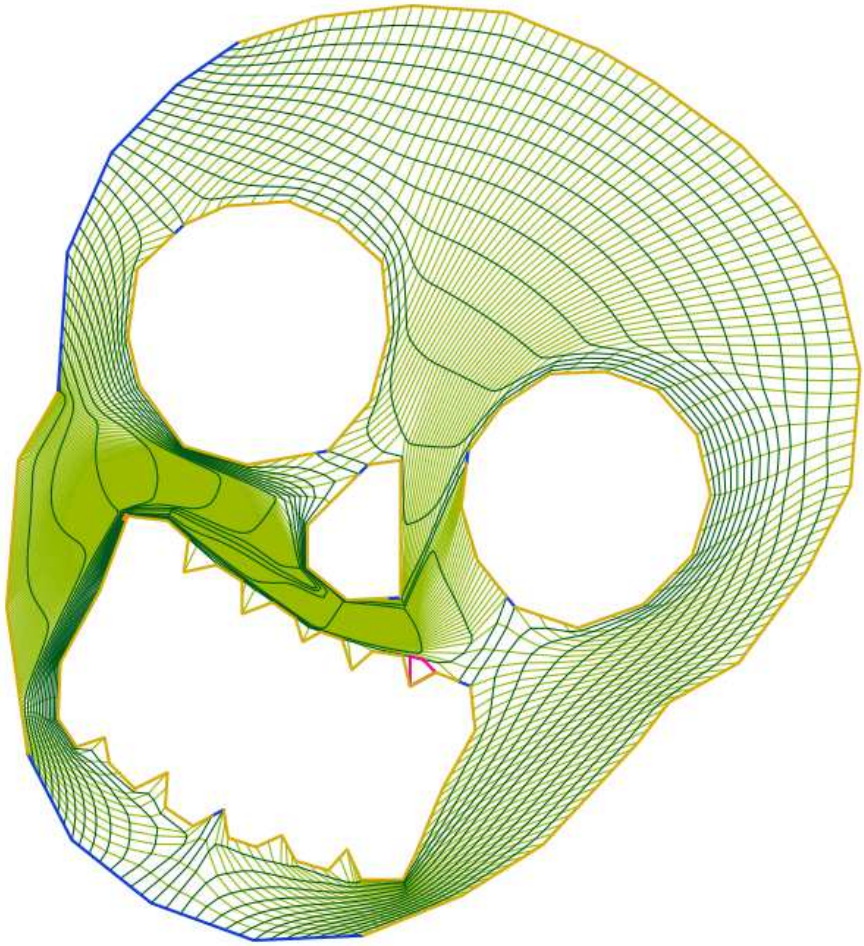


Figura A.9: Dimensión: 30x127 Funcional: $0.5 * H_\omega + 0.5 * (\textit{Área-Ortogonalidad})$

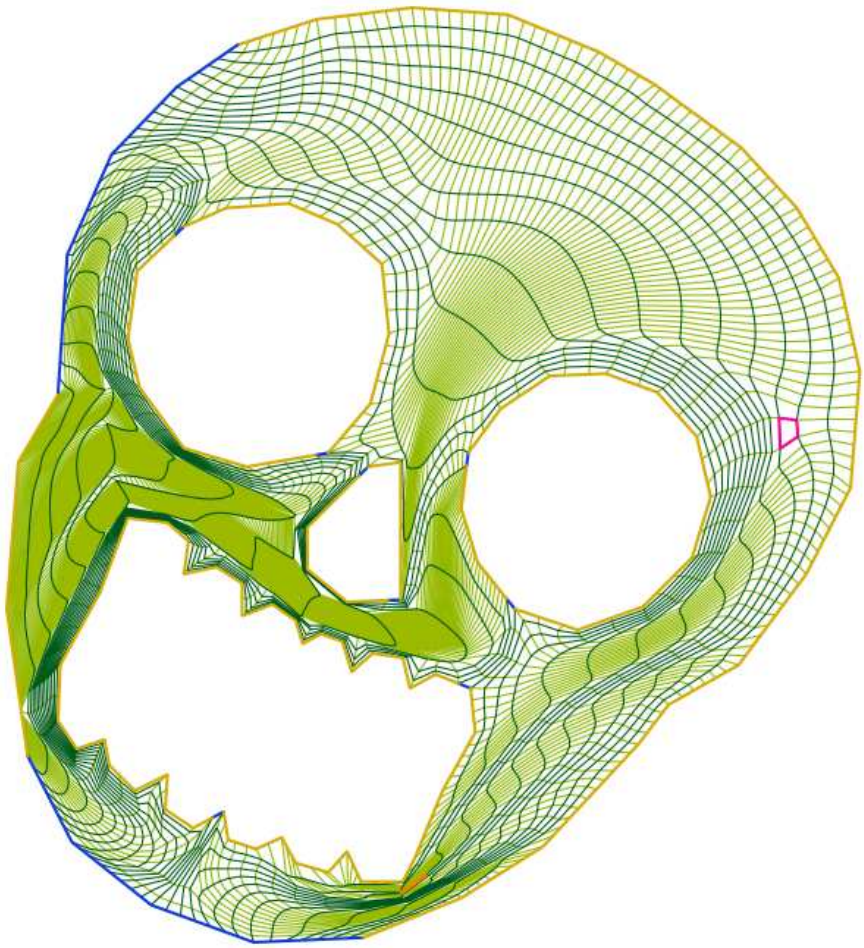


Figura A.10: Dimensión: 30x127 Funcional: $0.5 * S_{\omega} + 0.5 * (\textit{Área-Ortogonalidad})$

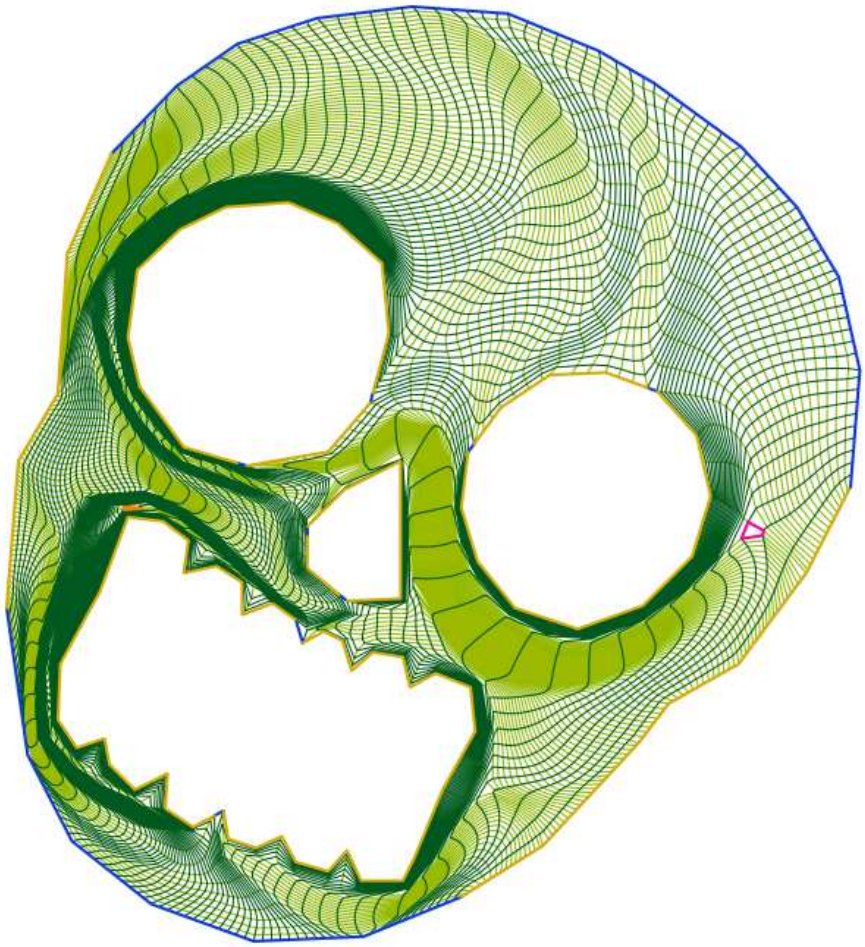


Figura A.11: Dimensión: 75x100 Funcional: $0.5 * S_\omega + 0.5 * (\textit{Área-Ortogonalidad})$

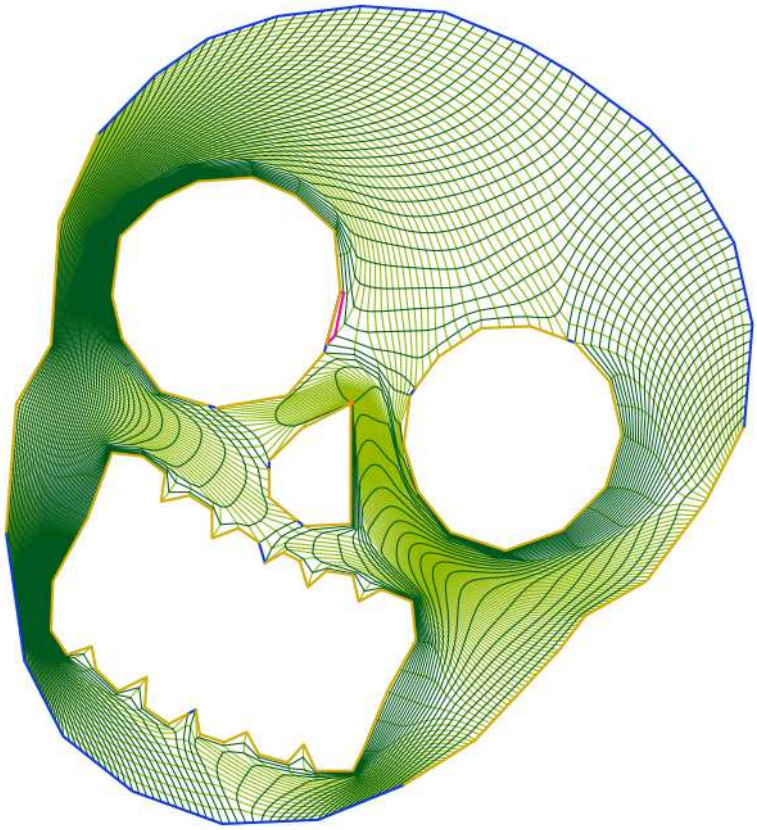
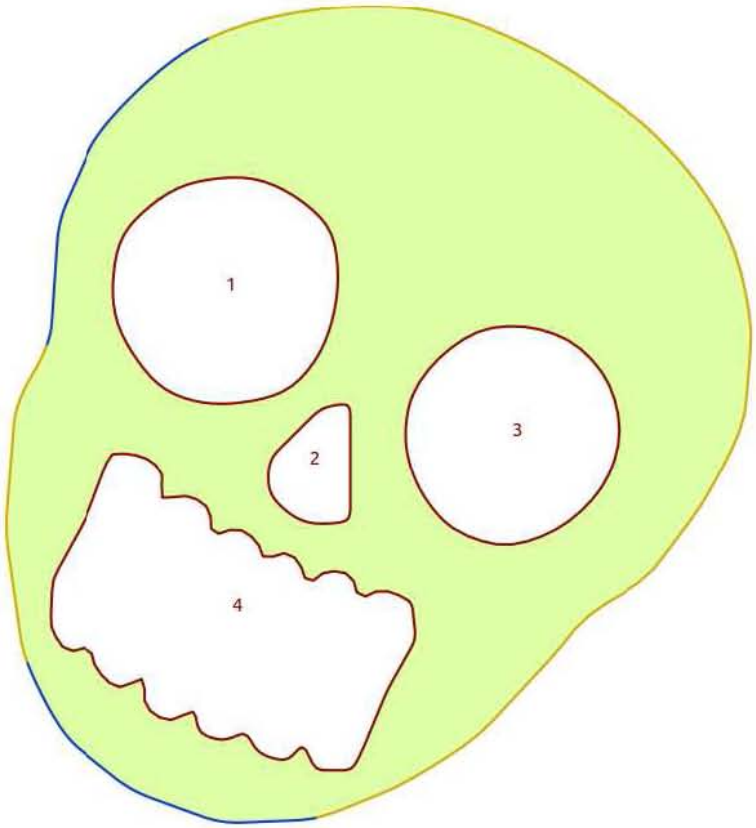


Figura A.12: Dimensión: 75x110 Funcional: $0.5 * H_{\omega} + 0.5 * (\textit{Área-Ortogonalidad})$


1
2
3
4

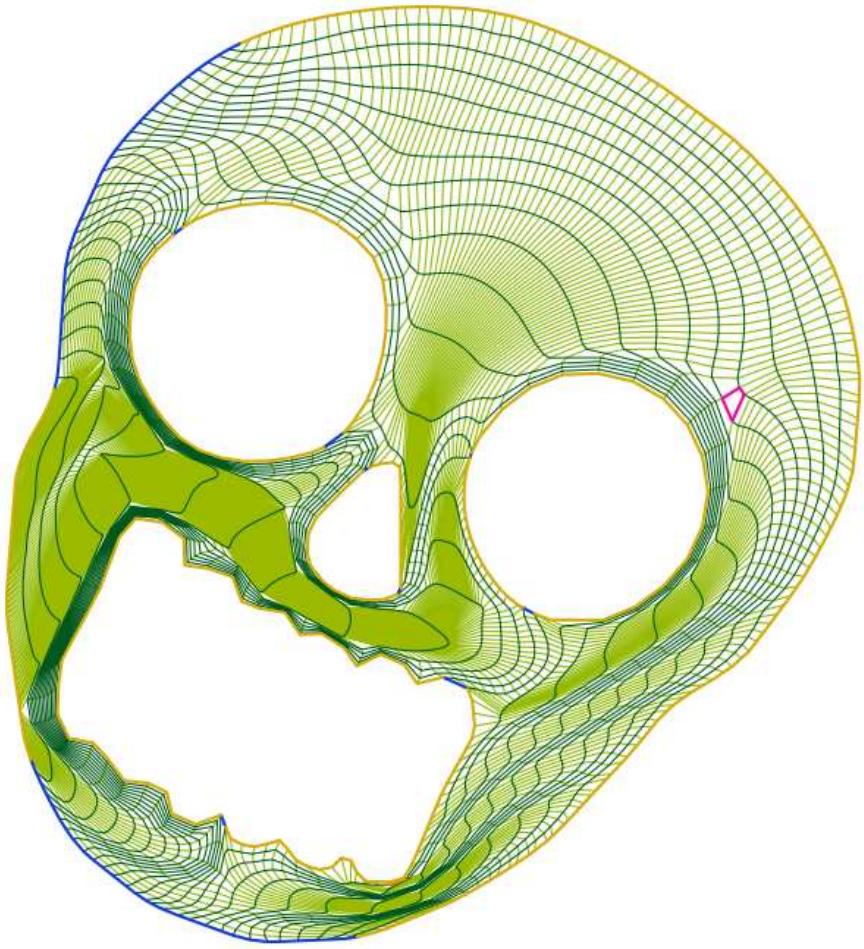


Figura A.13: Dimensión: 30x142 Funcional: $0.5 * S_{\omega} + 0.5 * (\acute{A}rea\text{-}Ortogonalidad)$

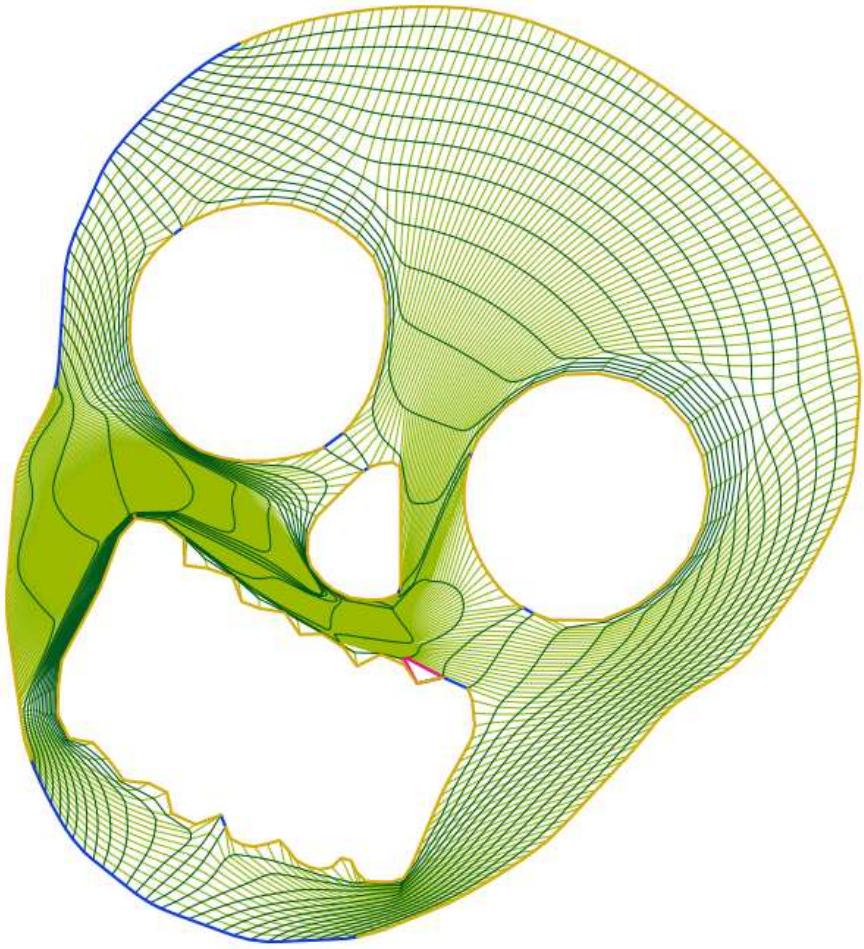
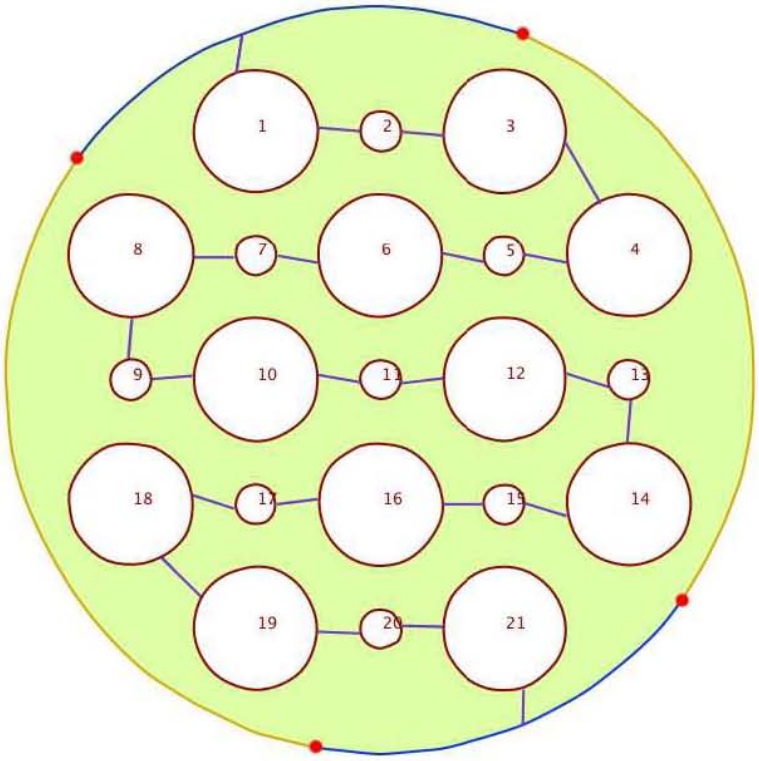


Figura A.14: Dimensión: 30x142 Funcional: $0.5 * H_\omega + 0.5 * (\textit{Área-Ortogonalidad})$


1
2
3
8
7
6
5
4
9
10
11
12
13
18
17
16
15
14
19
20
21

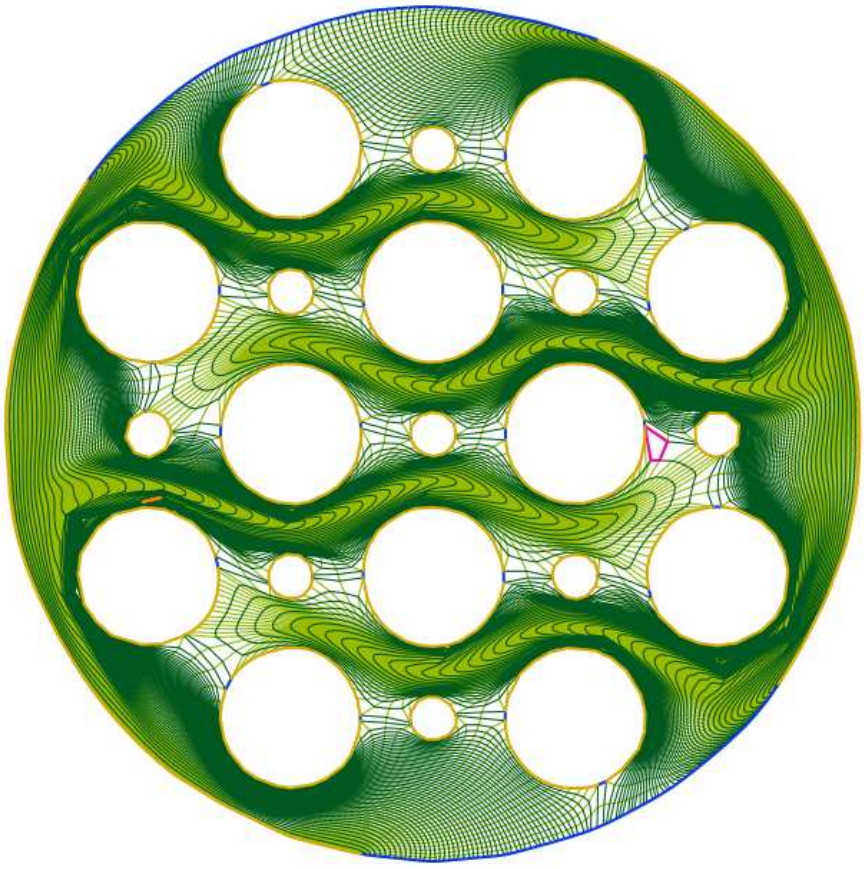


Figura A.15: Dimensión: 100x123 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea) + S_\omega + 0.5(Length)$

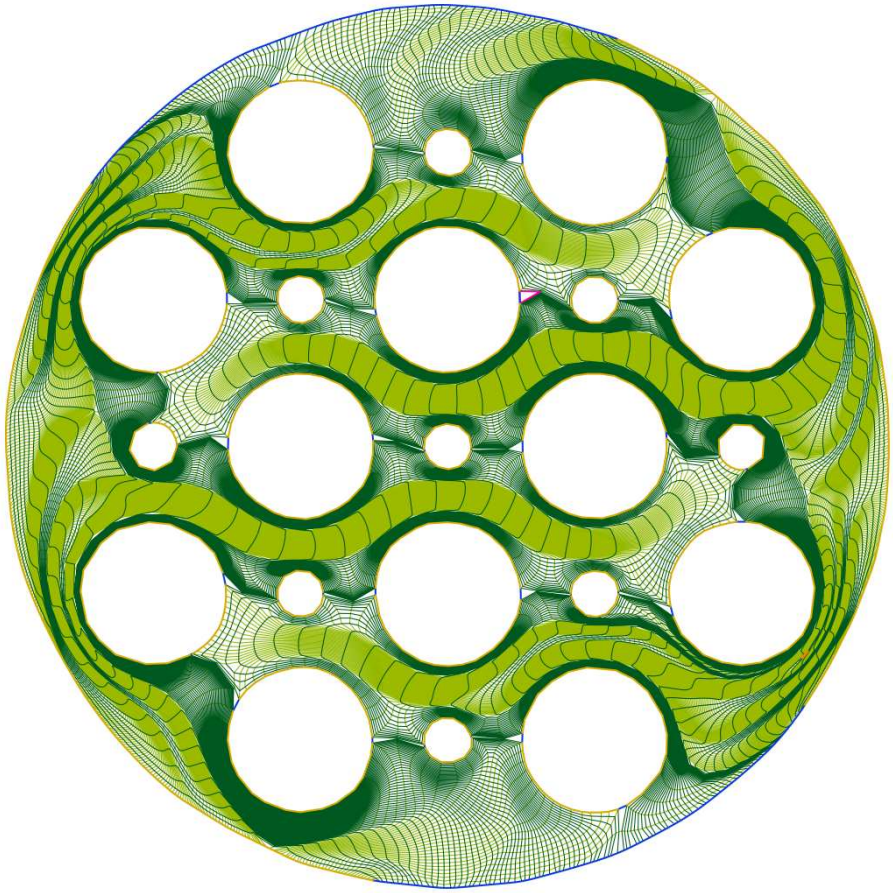


Figura A.16: Dimensión: 100x123 Funcional: $0.5 * S_\omega + 0.5 * (\acute{A}rea\text{-}Ortogonalidad)$

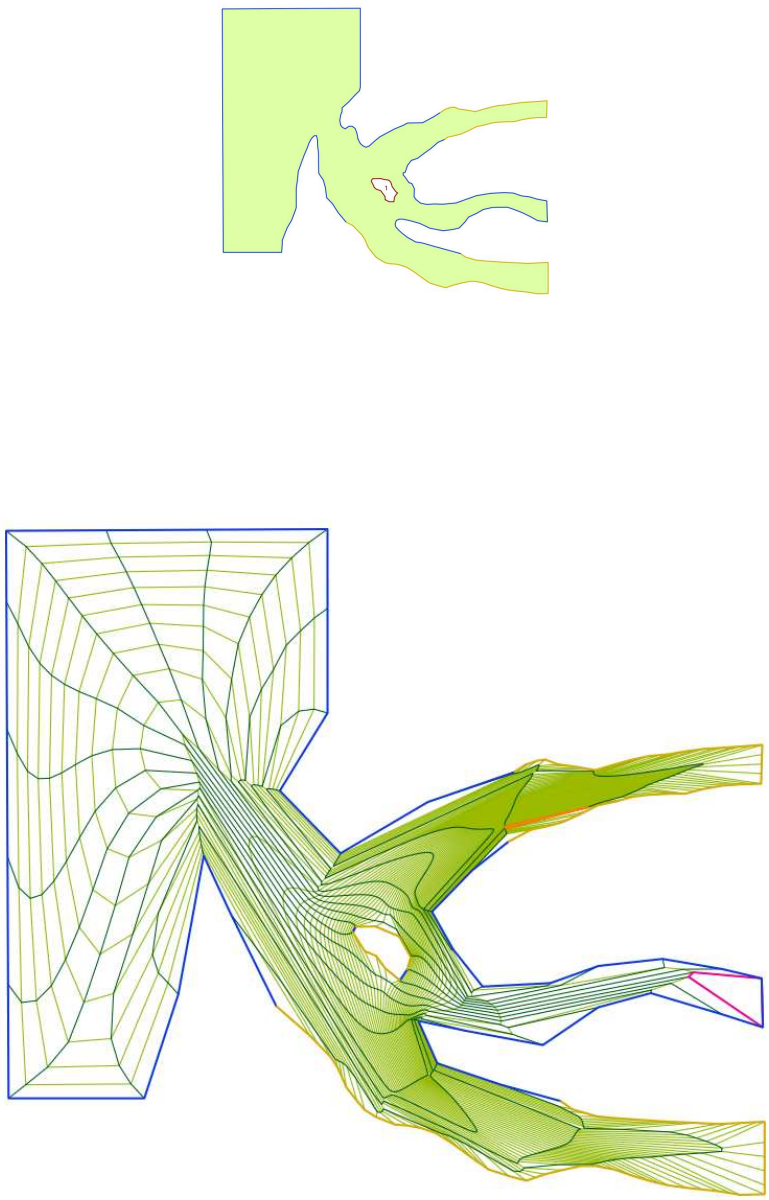


Figura A.17: Dimensión: 20x58 Funcional: $0.5 * H_\omega + 0.5 * (Área)$

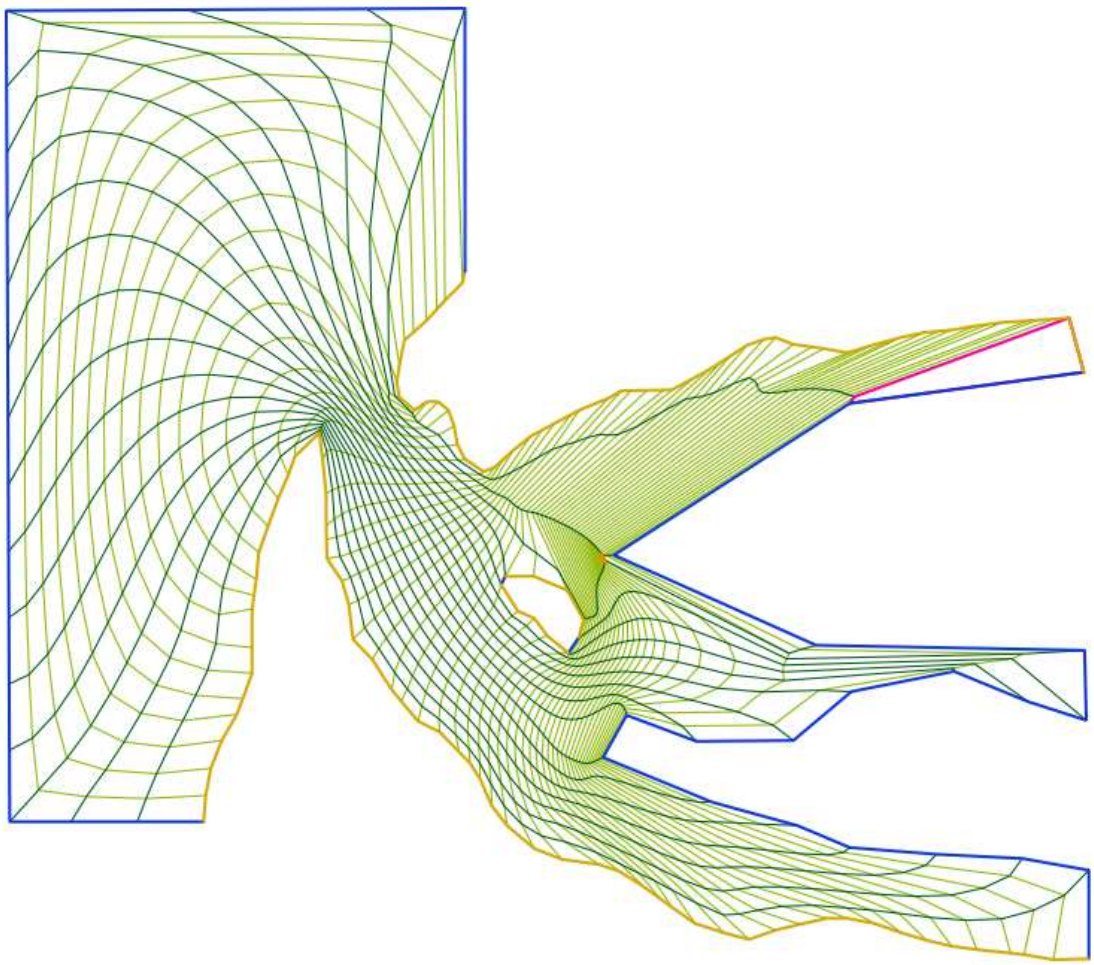


Figura A.18: Dimensión: 20x58 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

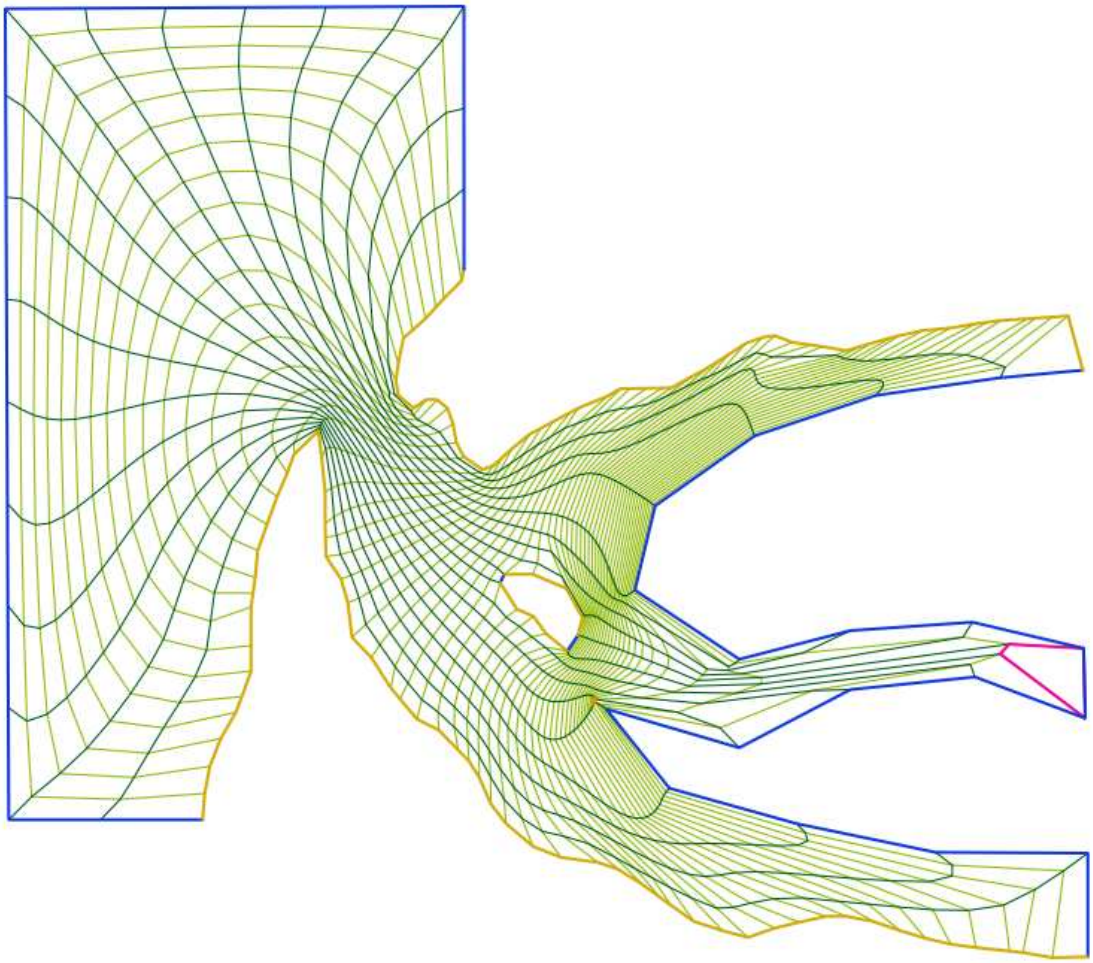


Figura A.19: Dimensión: 20x58 Funcional: $0.5 * H_{\omega} + 0.5 * (\acute{A}rea)$

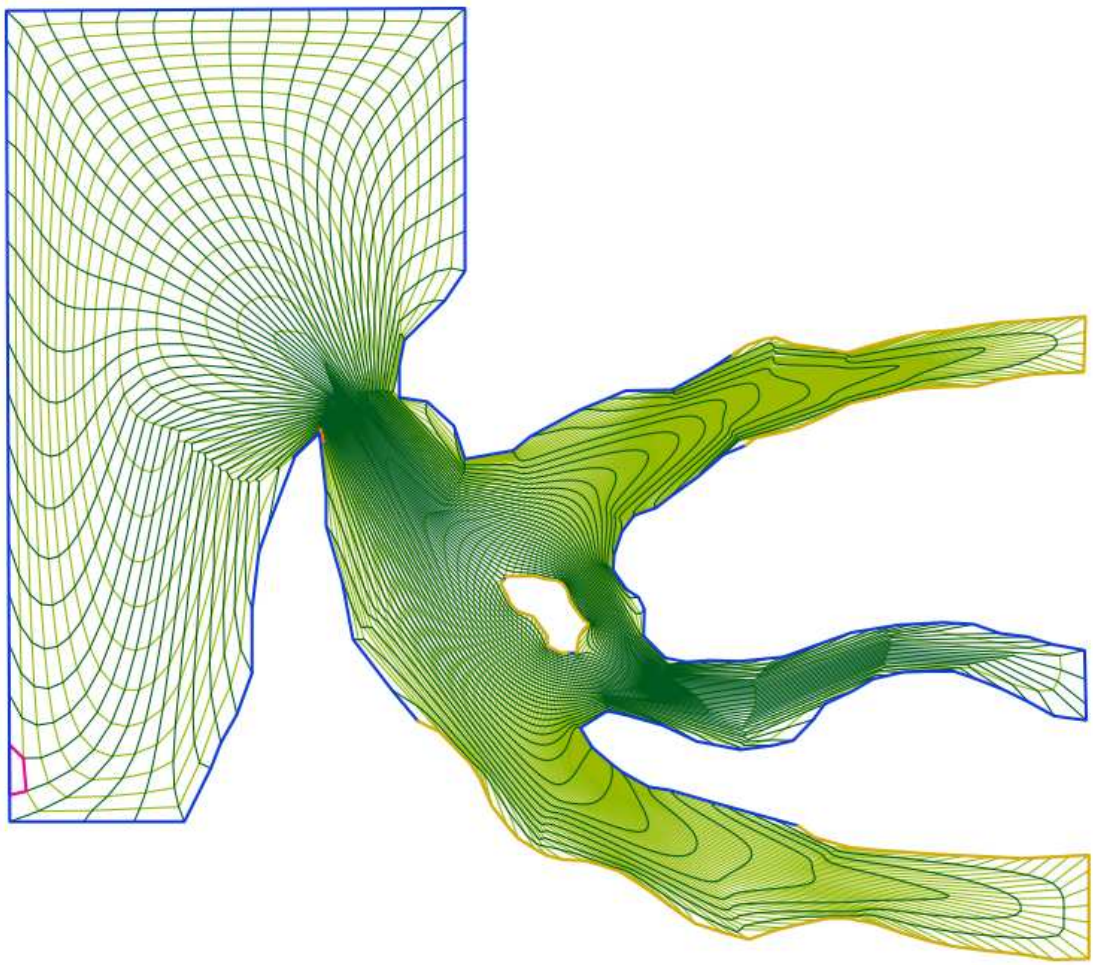


Figura A.20: Dimensión: 70x68 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

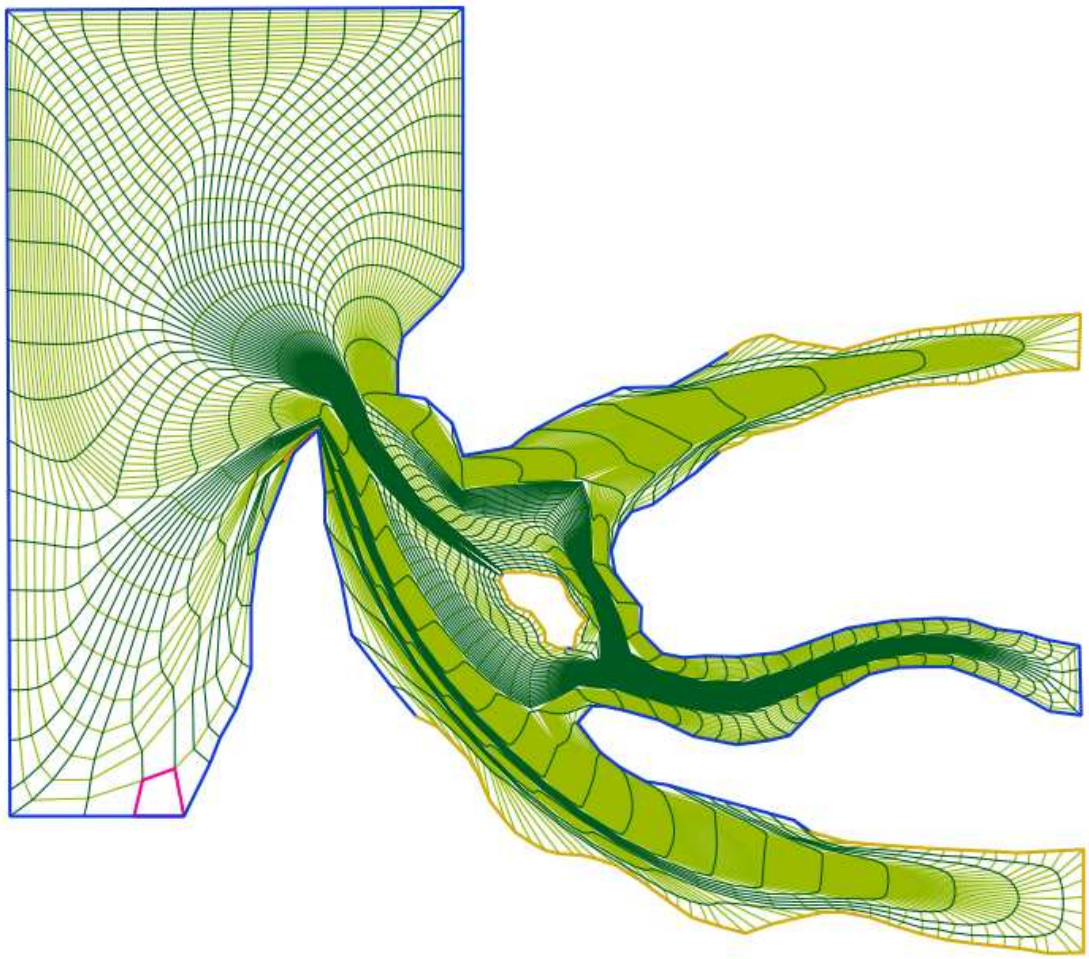


Figura A.21: Dimensión: 70x68 Funcional: $0.5 * S_\omega + 0.5 * (\textit{Área-Ortogonalidad})$

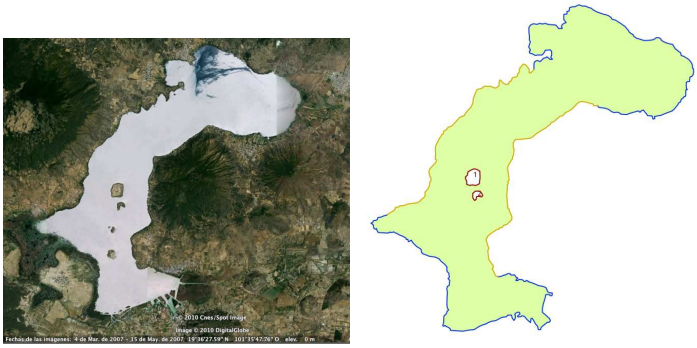


Figura A.22: Lago de Patzcuaro, Michoacán

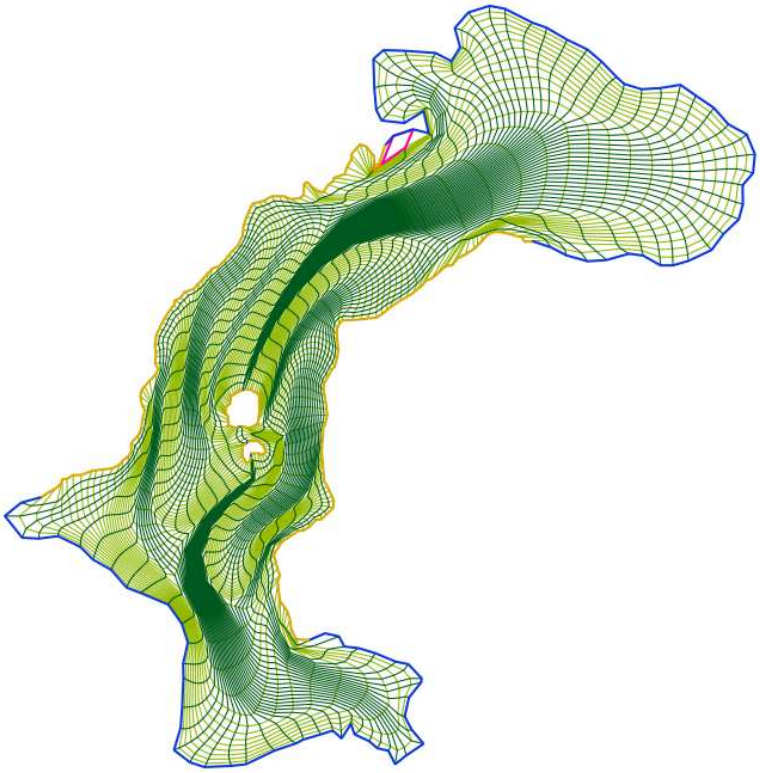


Figura A.23: Dimensión: 50x86 Funcional: $0.5 * S_\omega + 0.5 * (\acute{A}rea\text{-}Ortogonalidad)$

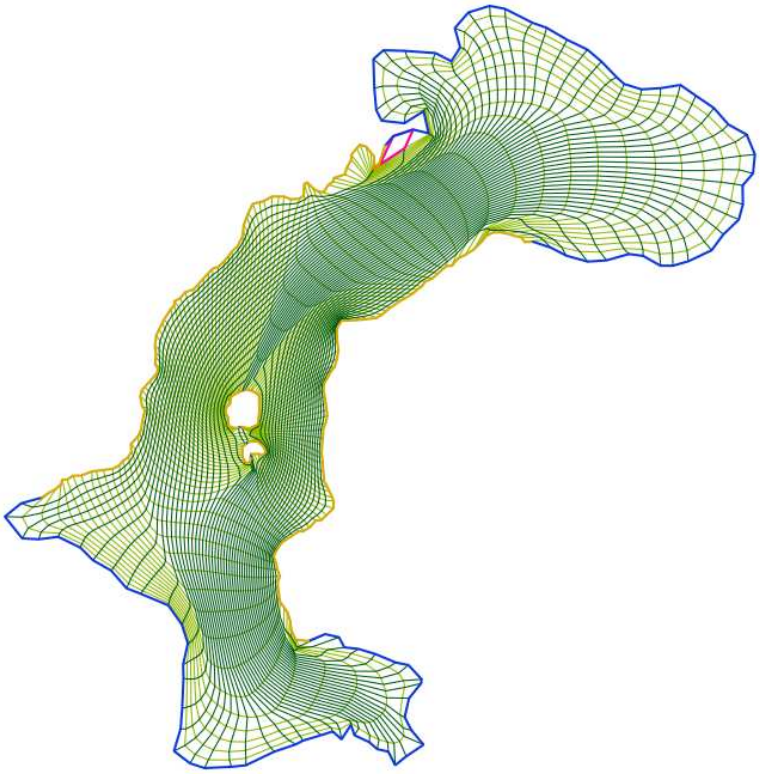


Figura A.24: Dimensión: 50x86 Funcional: $0.5 * H_{\omega} + 0.5 * (\textit{Área-Ortogonalidad})$

Figura A.25: Mar de Aral, Kazajistán / Uzbekistán

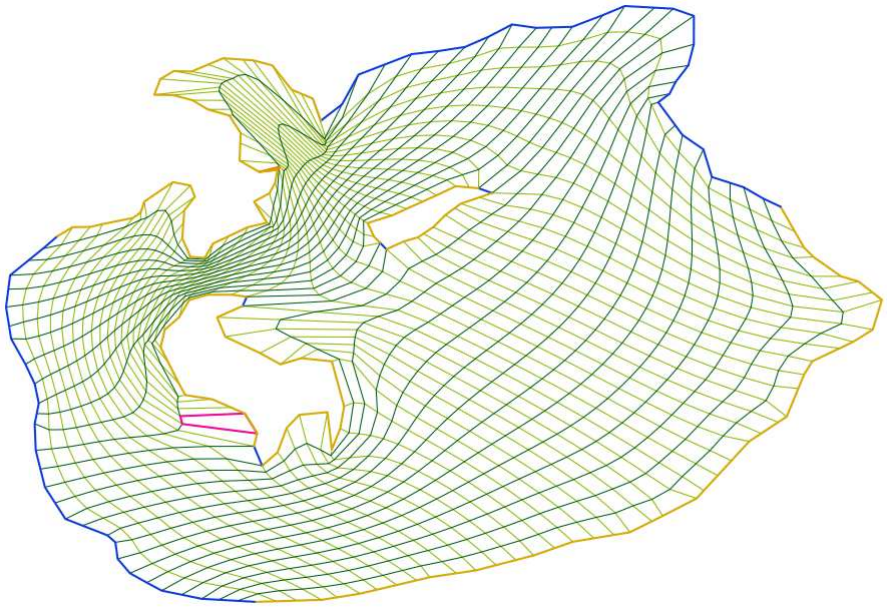


Figura A.26: Dimensión: 30x44 Funcional: $0.5 * H_{\omega} + 0.5 * (\acute{A}rea)$

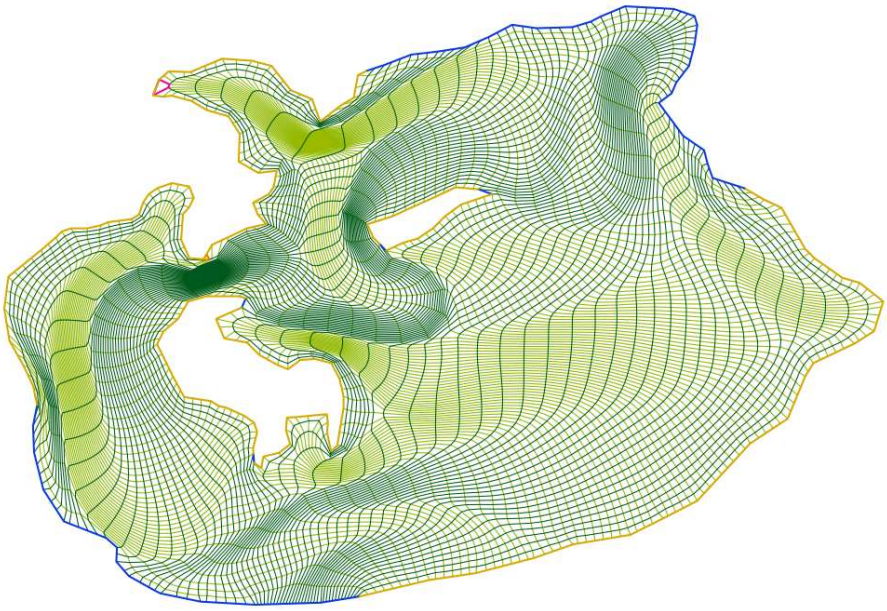


Figura A.27: Dimensión: 70x96 Funcional: $0.5 * S_{\omega} + 0.5 * (\acute{A}rea\text{-}Ortogonalidad)$

Figura A.28: Rocas Baimbridgen, Islas Galápagos, Ecuador

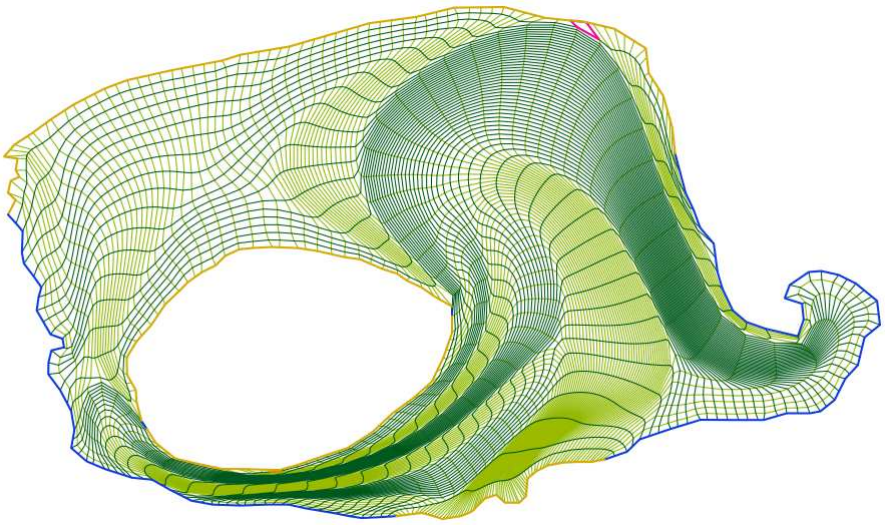


Figura A.29: Dimensión: 60x78 Funcional: $0.5 * S_\omega + 0.5 * (\textit{Área-Ortogonalidad})$

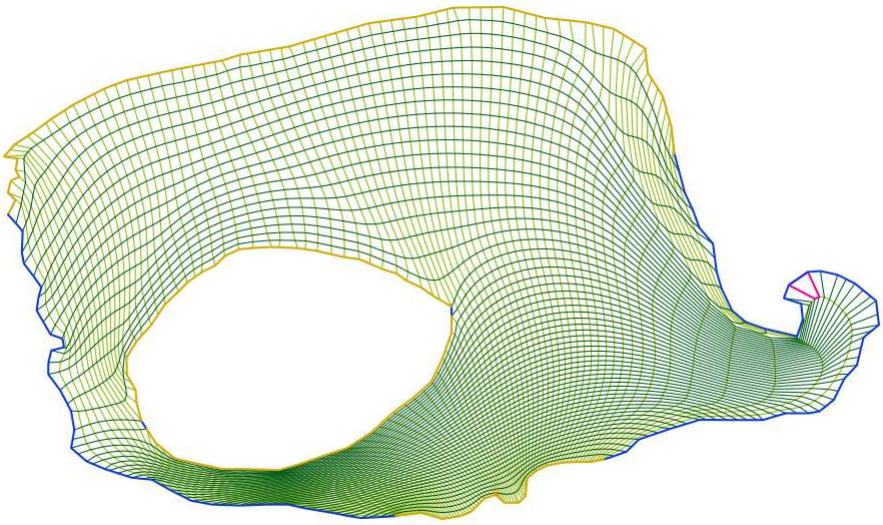


Figura A.30: Dimensión: 60x78 Funcional: $0.5 * H_\omega + 0.5 * (\textit{Área-Ortogonalidad})$

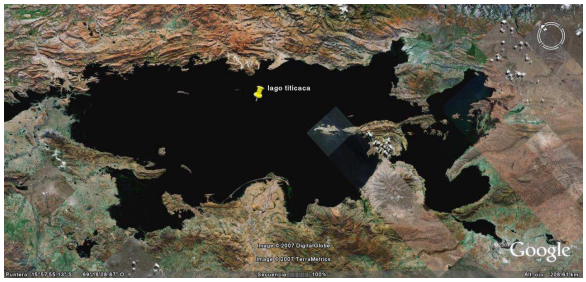


Figura A.31: Lago Titicaca, Andes Centrales, Bolivia y Perú

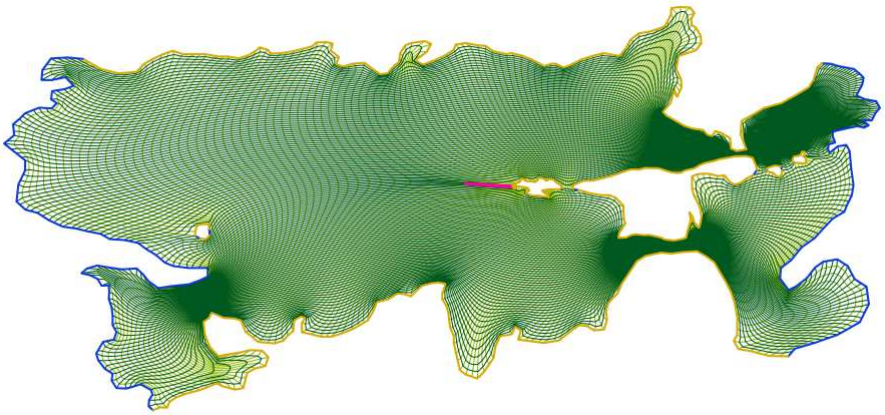


Figura A.32: Dimensión: 130x145 Funcional: $0.5 * H_\omega + 0.5 * (\acute{A}rea)$

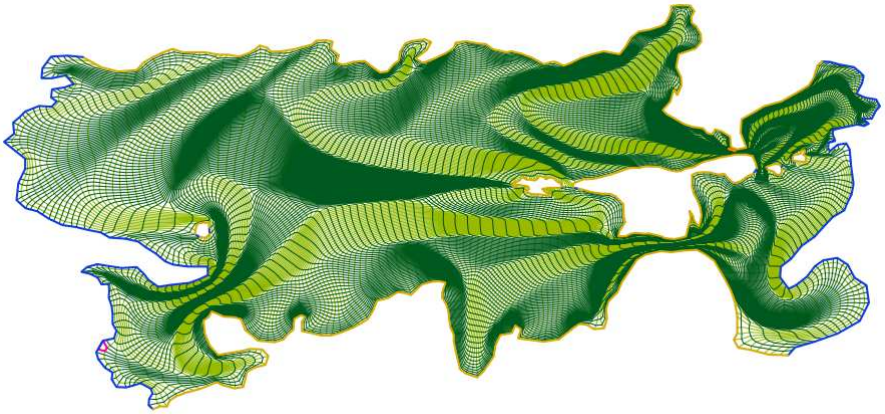


Figura A.33: Dimensión: 130x145 Funcional: $0.5 * S_\omega + 0.5 * (\acute{A}rea\text{-}Ortogonalidad)$

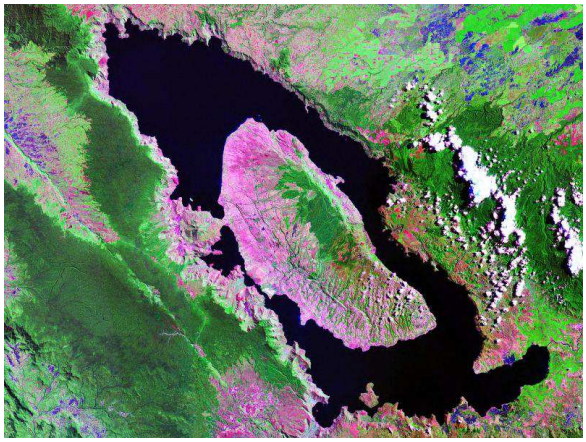


Figura A.34: Lago Toba, Sumatra Central, Indonesia

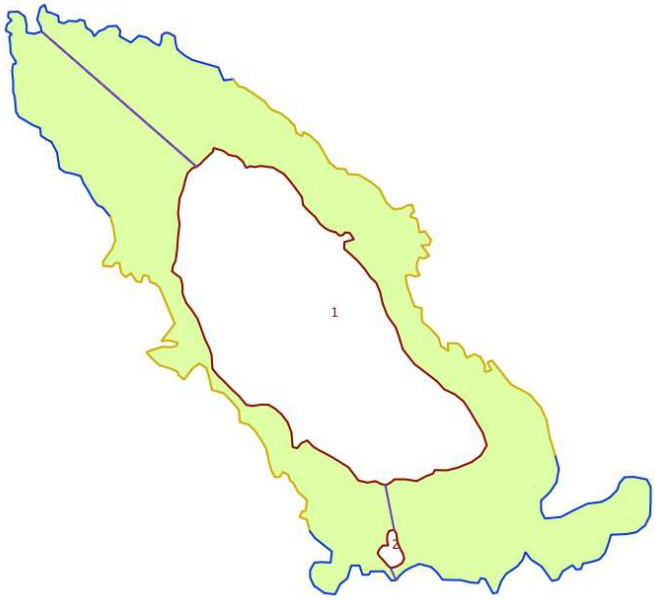


Figura A.35: Dimensión: 140x140 Funcional: $0.5 * S_\omega + 0.5 * (\textit{Área-Ortogonalidad})$

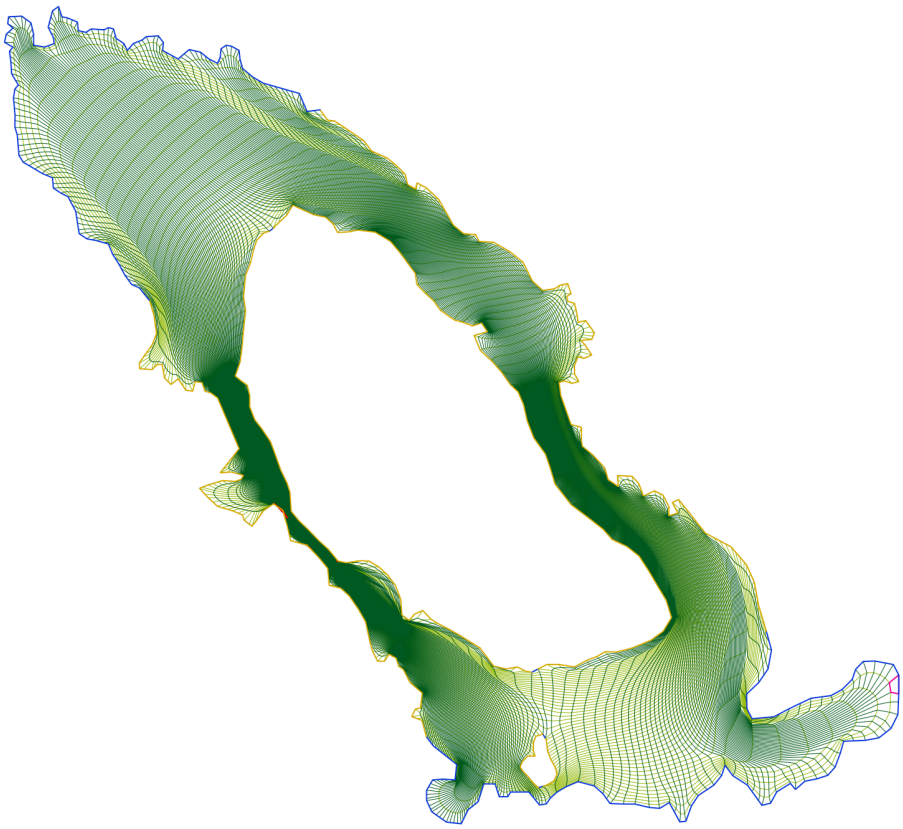
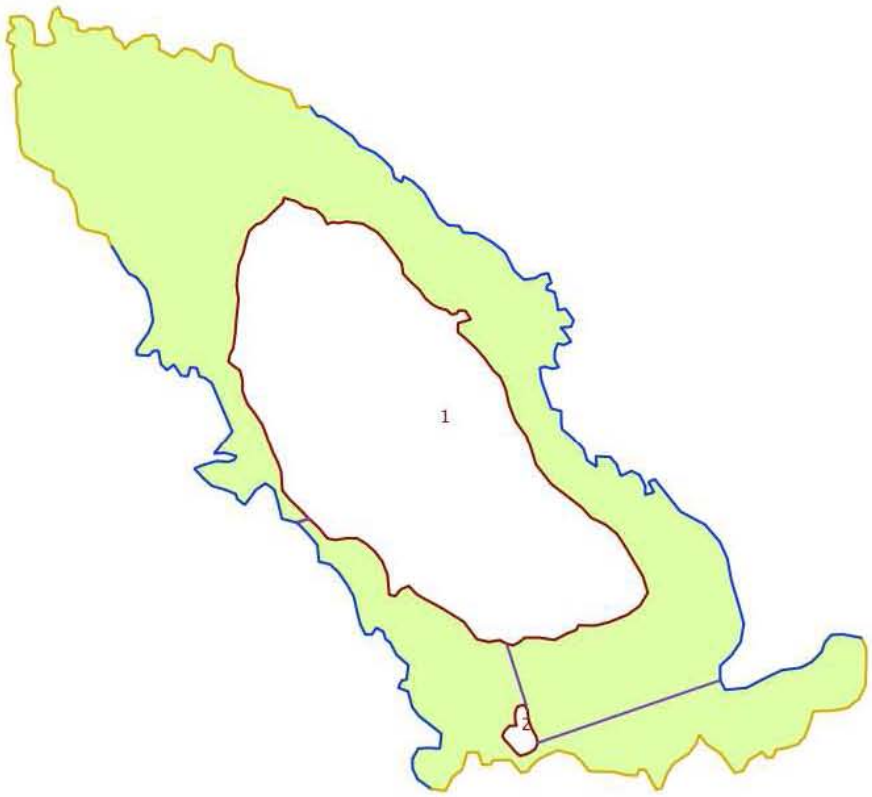


Figura A.36: Dimensión: 140x140 Funcional: $0.5 * H_{\omega} + 0.5 * (\textit{Área-Ortogonalidad})$


1
2

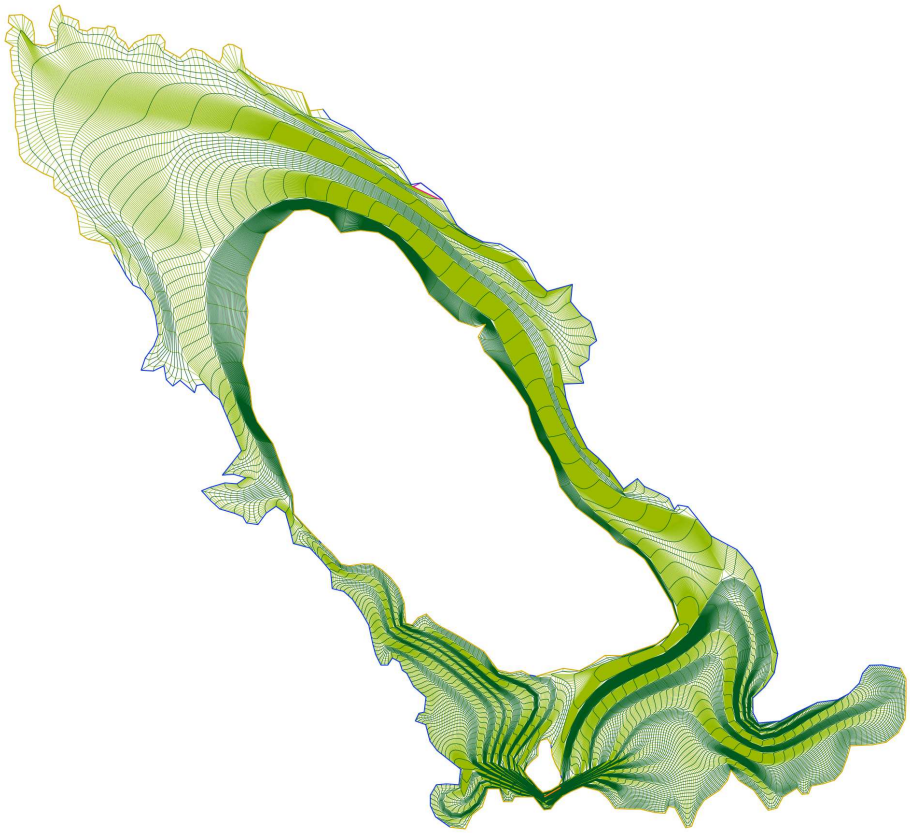


Figura A.37: Dimensión: 140x142 Funcional: $0.5 * S_\omega + 0.5 * (\textit{Área-Ortogonalidad})$

Apéndice B

Formatos de archivos

B.1. Contornos

El formato de los contornos nos da la siguiente información:

- N: el número de puntos del contorno externo (Ω_{ext}).
- $ibflag$: nos indica si tiene o no definida la frontera (1 si esta definida, 0 si no).
- nbi: el número de puntos de la i-ésima frontera,
- $xi \quad yi$: coordenadas (xi, yi) del i-ésimo punto del contorno exterior.
- NH: número de agujeros.
- hi: número de puntos del i-ésimo agujero.
- $ixj \quad iyj$: coordenadas (ixj, iyj) del j-ésimo punto del i-ésimo agujero.
- K: $2(NH + 1)$ número de vértices de la poligonal que divide a la región Ω_{ext} (llamados puntos especiales).
- Spi: índice del i-ésimo vértice de la poligonal que divide a la región Ω_{ext}.
- Cpi: índice del contorno donde se encuentra el i-ésimo vértice de la poligonal que divide a la región Ω_{ext} (El índice empieza desde cero).

Esto organizado como se ve en la figura B.1.

Con estos datos tenemos determinado el conjunto Ω, el algoritmo preciso que hace la escritura y lectura de este formato se encuentra en los archivos *contio.h* y *contio.cpp*.

La figura B.2 muestra un ejemplo con dos agujeros, y a continuación el archivo **CON** asociado a dicho contorno.

```
N ibflag nb1 nb2 nb3 nb4

x1     y1
x2     y2
.      .
.      .
.      .
xN     yN

NH
h1 h2 ... hNH
1x1    1y1
1x2    1y2
.      .
.      .
.      .
1xh1   1yh1
2x1    2y1
.      .
.      .
.      .
2xh2   2xh2
.      .
.      .
.      .
NHx1   NHy1
.      .
.      .
.      .
NHxhNH NHyhNH

Sp1    Cp1
Sp2    Cp2
.      .
.      .
.      .
SpK    CpK
```

Figura B.1: Formato de los contornos **CON**

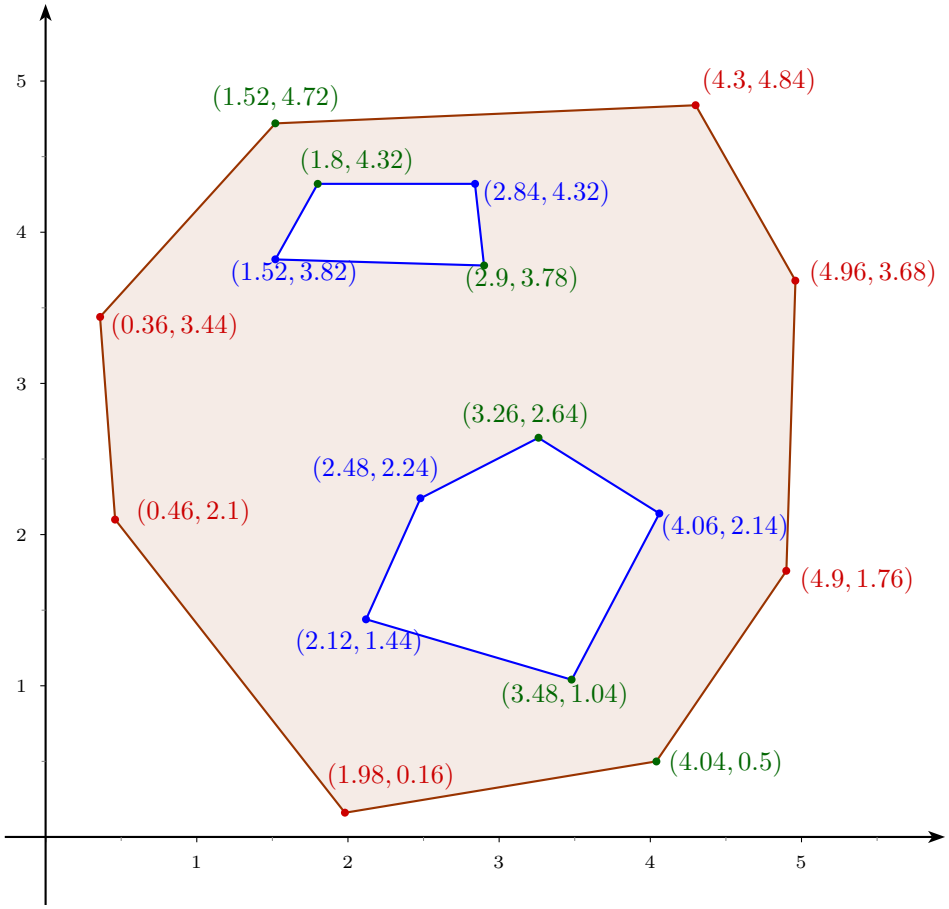


Figura B.2: Contorno con dos agujeros

```
 9     1     3     3     3     3
4.300000    4.840000
1.520000    4.720000
0.860000    3.480000
0.460000    2.100000
1.980000    0.160000
4.040000    0.500000
4.900000    1.760000
4.960000    3.680000
4.300000    4.840000
 2
 6    5
2.480000    2.240000
2.120000    1.440000
3.480000    1.040000
4.060000    2.140000
3.260000    2.640000
2.480000    2.240000
1.800000    4.320000
1.520000    3.820000
2.900000    3.780000
2.840000    4.320000
1.800000    4.320000
 1    -1
 0     1
 2     1
 4     0
 2     0
 5    -1
```

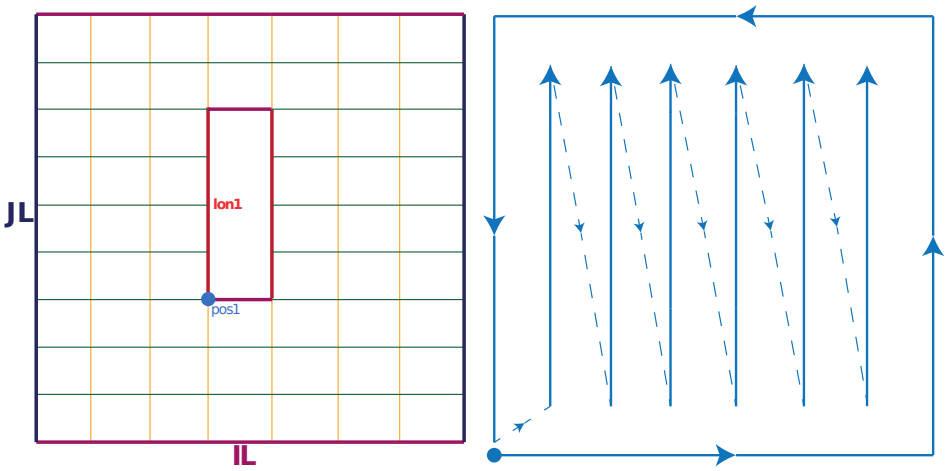
Figura B.3: Archivo *CON* asociado a la figura B.2

B.2. Mallas

El formato del archivo para las mallas contiene la siguiente información:

- IL: número de columnas de la matriz que representa a la región (malla).
- JL: número de filas de a matriz que representa a la región (malla).
- *name*: el nombre de la región (ya se usa).
- $xi \quad yi$: coordenadas (xi, yi) del i-ésimo punto de la malla.
- nc=0 número de celdas no convexas (ya no se usa).
- NCI=0 número de celdas inactivas (en general si hay agujeros no hay celdas inactivas).
- NH: número de agujeros.
- *posi*: índice del arreglo de la esquina superior izquierda del i-ésimo agujero en la matriz (los índices comenzado en cero).
- *loni*: longitud vertical del agujero i-ésimo.

El formato red piensa la matriz como un arreglo unidimensional recorrida como se ve en la figura B.4 y donde los índices pares son las coordenadas x, y los índices impares las coordenadas y de cada punto de la matriz (esto pues los índices en C++ comienzan en cero, si no, es al revés). Este formato de archivo se visualiza como se puede ver en la figura B.5.



(a) Partes del formato RED (b) Recorrido de lectura de RED

Figura B.4: Formato de lectura de los archivos RED

```
IL      JL
NAME
x1      y1      x2      y2      ...     x7      y7
x8      y8      .       .       .
.       .
.       .
.       .
xn      yn      .       .       .       xILJL   yILJL

0

0

NH
pos1    ...     posNH
lon1    ...     lonNH
```

Figura B.5: Formato de las malla **RED**

Siguiendo el ejemplo de la figura B.2, la siguiente figura muestra la configuración que se utilizó para generar la malla e inmediatamente después el archivo resultante.

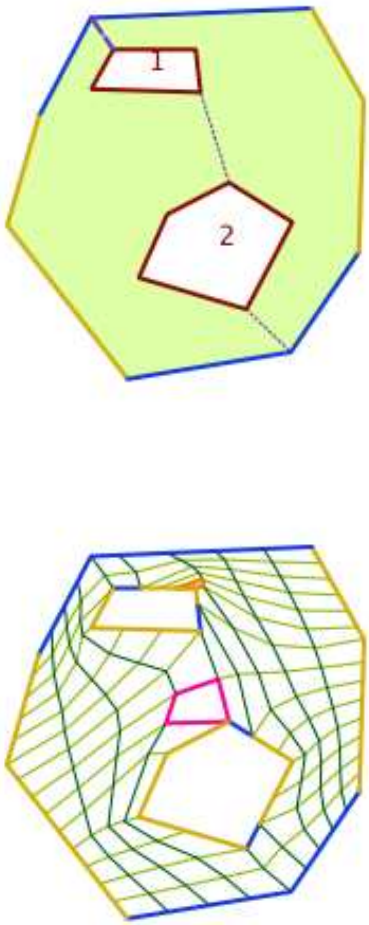


Figura B.6: Dimensión: 10x15

$m_1 = 5 \quad m_2 = 5 \quad m = 10$
$l_1 = 3 \quad l_2 = 3 \quad l_3 = 3$
$h_1 = 5 \quad h_2 = 5 \quad n = 15$

```
10 15
ejemplo2h.red
   1.980000    0.160000    2.495000    0.245000    3.010000    0.330000    3.525000
   0.415000    4.040000    0.500000    4.236892    0.788469    4.402669    1.031352
   4.568446    1.274235    4.734223    1.517117    4.900000    1.760000    4.907263
   1.992426    4.914527    2.224852    4.921790    2.457278    4.929053    2.689704
   4.936317    2.922130    4.943580    3.154556    4.950843    3.386982    4.960000
   3.680000    4.874982    3.829425    4.759986    4.031540    4.644989    4.233655
   4.529993    4.435770    4.414997    4.637885    4.300000    4.840000    3.692233
   4.813766    3.084466    4.787531    2.476700    4.761296    1.868933    4.735062
   1.520000    4.720000    1.355000    4.410000    1.190000    4.100000    1.025000
   3.790000    0.860000    3.480000    0.782420    3.212349    0.704840    2.944698
   0.627260    2.677047    0.549680    2.409395    0.460000    2.100000    0.605062
   1.914855    0.776929    1.695498    0.948796    1.476141    1.120664    1.256784
   1.292531    1.037428    1.464398    0.818071    1.636265    0.598714    1.808133
   0.379357    2.250968    0.476793    1.999432    0.705531    1.748443    0.940217
   1.524592    1.208246    1.438117    1.522185    1.361566    1.802474    1.298006
   2.057670    1.239466    2.297010    1.187115    2.528486    1.156985    2.772748
   1.101742    3.002713    1.055195    3.238783    1.035140    3.492542    2.718011
   0.576389    2.395758    0.809468    2.039679    1.039012    1.739127    1.328040
   1.801769    1.844075    1.843750    2.200728    1.887598    2.523263    1.845485
   2.785016    1.703297    2.979343    1.531035    3.170137    1.337309    3.388747
   1.247238    3.606794    1.216636    3.849757    3.207155    0.682726    2.857462
   0.924260    2.349383    1.138582    1.940170    1.385292    2.246622    2.239330
   2.464304    2.618435    2.587130    2.989119    2.469030    3.274739    2.120446
   3.378387    1.756948    3.493699    1.414505    3.730657    1.384569    3.902355
   1.395497    4.162715    3.690868    0.796240    3.480000    1.040000    2.719354
   1.263720    2.120000    1.440000    2.480000    2.240000    3.260000    2.640000
   3.123901    3.174193    2.900000    3.780000    2.411795    3.794151    1.923589
   3.808302    1.520000    3.820000    1.800000    4.320000    1.695160    4.480650
   3.927314    1.063916    3.642897    1.348943    3.816451    1.678097    4.060000
   2.140000    3.871715    2.257678    3.556170    2.454894    3.371145    3.235361
   2.861431    4.127121    2.840000    4.320000    2.591662    4.320000    2.370460
   4.320000    2.149258    4.320000    2.086859    4.524080    4.204126    1.285179
   4.113581    1.556151    4.169846    1.890549    4.215735    2.222245    4.027630
   2.498548    3.784806    2.789070    3.626982    3.251911    3.309170    3.823326
   2.932398    4.346020    2.924789    4.430375    2.881639    4.513453    2.794739
   4.592894    2.652470    4.671250    4.451140    1.530211    4.411033    1.809555
   4.418603    2.109139    4.395416    2.398400    4.265766    2.677264    4.106337
   2.956144    3.967892    3.268391    3.804878    3.604554    3.605739    3.945532
   3.452442    4.215944    3.345072    4.411173    3.272080    4.558147    3.198975
   4.679360    4.673973    1.774179    4.658938    2.038058    4.654678    2.303693
   4.628870    2.563552    4.567944    2.817220    4.495029    3.068815    4.421528
   3.320066    4.338916    3.565235    4.236888    3.803599    4.130171    4.040231
   4.019410    4.262711    3.907267    4.466709    3.799688    4.650631
0
0
2
184 172
5 5
```

Figura B.7: Archivo *RED* asociado a la figura B.6

Bibliografía

[1] Chávez González A., Cortés-Medina A., and Tinoco-Ruiz J.G. Testing the quality of grids generated using the direct approach. *Applied Numerical Mathematics*, (40):191–206, 2002.

[2] Chávez González A., Cortés-MedinaA., and Tinoco-Ruiz J.G. A direct finite-difference scheme for solving pdes over general two-dimensional regions. *Applied Numerical Mathematics*, (40):219–233, 2002.

[3] S. N. Atluri. *The Meshless Method (MLPG) for Domain and BIE Discretization*. Tech Science Press, 2004.

[4] P. Barrera and J. G. Tinoco. Area control in generating smooth and vonces grids over general plane regions. *Journal of Computational and Applied Mathematics*, 1999.

[5] P. Barrera and J.G. Tinoco. Smooth and convex grid generation over general plane regions. *Mathematics and Computers in Simulation*, 1998.

[6] P. Barrera-Sánchez, F.J. Domínguez-Mota, G.F. González-Flores, and J. G. Tinoco. Generating quality structured convex grids on irregular regions. *Electronic Transactions on Numerical Analysis*, 34:76–89, 2009.

[7] MA Burlington. Jaguar land rover achieves goal of sophisticated engineering with exa powerflow 4.1. `http://www.prweb.com/releases/cfd/simulation/prweb1891174.htm`, January 2009.

[8] César Carreón. Un módulo para el tratamiento de contornos en el sistema unamalla. Tesis de licenciatura, UNAM, 2008.

[9] J. E. Castillo. *Variational Grid Generation*. PhD thesis, University of New Mexico, 1987.

[10] J. E. Castillo. Mathematical aspects of numerical grid generation. *SIAM*, 1991.

[11] Darren Engwirda. Unstructured 2d triangular mesh generation in matlab. `http://www.advancedmcode.org/unstructured-2d-triangular-mesh-generation-in-matlab.html`, July 2010.

[12] Guilmer F. González Flores. Generación de mallas en regiones planas irregulares. Master's thesis, UNAM, 1994.

[13] Damrong Guoy and Jeff Erickson. Automatic blocking scheme for structured meshing in 2d multiphase flow simulation. In *13th Annual International Meshing Roundtable*, pages 121–132, 2004.

[14] James M. Hyman, Shengtai Li, Patrick Knupp, and Mikhail Shashkov. An algorithm for aligning a quadrilateral grid with internal boundaries. *Journal of Computational Physics*, (163):133–149, 2000.

[15] P. Knupp and S. Steinberg. *Fundamentals of grid generation*. CRC Press, 1992.

[16] Yu. V. Likhanova, V.D. Liseikin, D.V. Patrakhin, and I.A. Vaseva. Generation of block structured smooth grids. *Verychislitelsftsnnerye Tehnologii*, 11(4), 2006.

[17] Shashkov M. *Conservative finite difference met- hods on general grids.* Symbolic and Numeric Computation Series. CRC Press, 1996.

[18] J. Nocedal and J. Wright. *Numerical optimization*. Springer Series in Operations Research, 1999.

[19] Nokia. Qt. `http://qt.nokia.com/`, August 2011.

[20] Nvidia. Physx. `http://developer.nvidia.com/object/physx.html`, February 2009.

[21] P. J. Roache and S. Steinberg. Variational grid generation. *Numerical Methods for P.D.E.s*, 2:71–96, 1986.

[22] B. K. Soni, J.F. Thompson, and P. Weatherhill. *Handbook of Grid Generation*. CRC Press, 1999.