eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN INGENIERÍA MECÁNICA

TRABAJO FIN DE MÁSTER

Procedimiento Para La Generación De Modelos Geométricos Con Topologías De "Infill" Óptimas En Fabricación Aditiva

Estudiante José Antonio Postigo Martín
Director Dr. Rubén Ansola Loyola
Departamento Ingeniería Mecánica
Curso académico 2021/2022

Bilbao, 5 de julio de 2022

Resumen

En este trabajo se describirá el proceso de detección, trazado de contornos y exportación, automatizados, de un modelo obtenido mediante una optimización topológica a un archivo CAD, con el objetivo de usarlo para fabricar las piezas mediante fabricación aditiva.

Este proceso consiste en idear un método capaz de interpretar los resultados de optimización topológica mediante un conjunto de algoritmos programados en lenguaje *Python*, y tomando como referencia métodos tradicionales de procesamiento de imágenes. Los algoritmos se dividirán en una serie de procesos que comenzarán por filtrar los datos para obtener una imagen binaria, distinguir cada uno de los agujeros o dominios vacíos, extraer los límites de cada contorno y aplicar las técnicas necesarias de suavizado y simplificación de los contornos. Estos algoritmos, aunque tomando su base en otros de distinta finalidad, es un método propio y específico para soluciones de optimización topológica, bidimensional y con elementos cuadrados.

Por último, aprovechando que el software FreeCAD integra el lenguaje Python por defecto para crear piezas mediante código, se emplearán los datos obtenidos de los procesos anteriores para generar el modelo CAD correspondiente.

Palabras claves

Optimización Topológica, MEF (Método de los elementos finitos), "*Infill*", CAD (*computer-aided design*), Vectorización, FA (Fabricación Aditiva), Píxel.

Abstract

In this work, the process to automated detection, contour tracing and export of a model obtained by topology optimization to a CAD file, will be describe with the objective of using it to manufacture parts through additive manufacturing.

This process consists of devising a method able to interpret the results of topology optimization using a set of algorithms programmed in *Python* language, and taking traditional image processing methods as a reference. The algorithms will be divided into a series of processes that will begin by filtering the data to obtain a binary image, distinguishing each of the holes or empty domains, extracting the limits of each contour and applying the necessary techniques for smoothing and simplifying the contours. These algorithms, although based on other algorithms with different purposes, is a specific method for two-dimensional topology optimization solutions with square elements.

Finally, taking advantage of the fact that the FreeCAD software integrates the Python language by default to create parts using code, the data obtained from the previous processes will be used to generate the corresponding CAD model.

Key Words

Topology optimization, FEM (Finite element method), "Infill", CAD (*computer-aided design*), Vectorization, AD (Additive manufacturing), Pixel.

Laburpena

Lan honetan, CAD fitxategi batera optimizazio topologikoaren bidez lortutako modelo bat detektatzeko, inguruak trazatzeko eta esportatzeko prozesu automatizatua deskribatuko da. Piezak fabrikazio gehigarriaren bidez fabrikatzeko helburuarekin.

Prozesu hau optimizazio topologikoaren emaitzak *Python* lengoaian programatutako algoritmo-multzo baten bidez interpretatzeko gai den metodo bat asmatzean datza, eta irudiak prozesatzeko metodo tradizionalak erreferentziatzat hartuta. Algoritmoak hainbat prozesutan banatuko dira. Prozesu horiek datuak filtratzen hasiko dira, irudi bitar bat lortzeko, zulo edo domeinu huts bakoitza bereizteko, inguru bakoitzaren mugak ateratzeko eta inguruak leuntzeko eta sinplifikatzeko beharrezko teknikak aplikatzeko. Algoritmo horiek, oinarria helburu desberdineko beste batzuetan hartuta ere, optimizazio topologikoko, bidimentsionaleko eta elementu karratuko soluzioetarako metodo propio eta espezifikoa dira.

Azkenik, FreeCAD softwareak kode bidez piezak sortzeko Python lengoaia lehenetsia integratzen duela aprobetxatuz, aurreko prozesuetatik lortutako datuak erabiliko dira dagokion CAD eredua sortzeko.

Gako-hitzak

Optimizazio Topologikoa, EFM (Elementu finituen metodoa), "*Infill*", CAD (*computer-aided design*), Bektorizazioa, FG (Fabrikazio Gehigarria), Pixel.

Acrónimos y Abreviaturas

AM: *Additive Manufacturing* (fabricación aditiva)

β**:** Factor de escarpado

$\mathbb{B}_{\mathcal{R}}$**:** Dominio de un radio $\mathcal{R}$

c**:** Función objetivo

CAD: *Computer Aided Desing* Dieño asistido por ordenador

$\mathbb{D}$**:** Conjunto de elementos

D_e**:** Factpr de escala OCM

DLP: Procesamiento digital de la luz

E**:** Módulo de Young

FDM: *Fused Deposition Modelling*

FEM: *Finite element method* (Método de los elementos finitos)

$\boldsymbol{f}$**:** Vector de fuerzas externas

$\boldsymbol{K}$**:** Matriz de rigidez

$\boldsymbol{k}$**:** Núcleo del filtro de convolución

$\boldsymbol{K}_f$**:** Matriz de rigidez filtro PDE

λ**:** Multiplicador de Lagrange

MMA: *Method of moving asympotes*

N_e**:** Elementos dentro de una circunferencia de radio $\mathcal{R}$

$\boldsymbol{N}_e(\boldsymbol{x})$**:** Vector de funciones de interpolación

ν**:** Factor de amortiguamiento

OCM: Criterios de optimalidad

OT: Optimización Topológica

Ω**:** Dominio

Ω_m**:** Dominio sólido

Ω_v**:** Dominio vacío

ω**:** Peso del filtro

p**:** Penalización método SIMP

PDE: *Partial differential equation* (Ecuación diferencial en derivadas parciales)

$\mathcal{R}$**:** Radio de filtrado

$reloj$**:** Dirección del operador de trazado

ρ**:** Variable de diseño

$\boldsymbol{\rho}$**:** Vector de variables de diseño

$\tilde{\rho}$**:** Variable de diseño filtrada

***SIMP*:** Material Sólido Isótropo con Penalización

SLA: Estereolitografía

SLS : Selective laser Sintering

SLP: Métodos secuenciales de programación lineal

$\boldsymbol{t}$**:** Vector de fuerzas aplicadas en el contorno

TFM: Trabajo Fin de máster

μ**:** Posición escalón

$\boldsymbol{u}$**:** Vector de desplazamientos

$\boldsymbol{x}$**:** Punto del Dominio

: Píxel izquierdo de operador

: Píxel central de operador

➔: Píxel derecho de operador

➚: Píxel actual con $reloj = 1$

Índice

Índice de figuras

Índice de Tablas

Parte I

Memoria

Memoria

Contenido

Introducción

Si se observan las tendencias políticas y económicas de los países desarrollados en la actualidad, cada vez se naturalizan más términos como eficiencia energética, aprovechamiento de recursos materiales o desarrollo sostenible. Dentro de estas tendencias de socioeconómicas, la optimización puede ser, y de hecho, empieza a ser una herramienta crucial. Debido al crecimiento del volumen de investigaciones, la interconexión entre ramas científicas diferentes y la enorme evolución de la capacidad de cálculo de las computadoras; la optimización comienza a ser utilizada en una gran variedad de áreas de conocimientos, ya que es posible resolver problemas de un gran volumen de datos y unas restricciones preimpuestas cumpliendo siempre con las ideas de eficiencia y aprovechamiento de recursos.

Ejemplos de todo lo anterior pueden encontrarse, por ejemplo, en la predicción de la demanda y optimización de la producción de los sistemas de generación de energía eléctrica, la planificación de la producción de las grandes empresas o el tema que ocupa este trabajo: la optimización estructural.

La idea de optimización en la ingeniería y la arquitectura, aun sin ser necesariamente nombrada como tal, ha estado presente a lo largo de la historia. Un ejemplo podría ser la arquitectura medieval y su búsqueda de la grandeza y perfección de las estructuras, conceptos que a menudo iban asociados al estatus político o religioso de quien las encargaba o incluso eran medida de fervor y sacrificio religioso. Esto indica que de manera histórica, se ha desafiado a los ingenieros a buscar diseños óptimos basados en criterios objetivos o subjetivos. La metodología de diseño estaba en parte basada en la originalidad, conocimientos y experiencia del ingeniero, optando normalmente por hacer pequeñas variaciones a diseños preexistentes, hasta que se cumplieran los requerimientos estructurales y/o artísticos.

En la actualidad esto empieza a no tener que ser así. La optimización ahora puede entenderse como un proceso algorítmico basado en conocimientos físicos y matemáticos, que buscan la mejor solución para unos requerimientos concretos. Esto hace que la idea de diseño basado en la experiencia y la intuición puedan ceder espacio a métodos de diseño racionales, fundamentados en su mayoría en el *método de los elementos finitos (FEM)*.

Este cambio de metodología de diseño, no solo permite complementar el conocimiento tácito que el ingeniero posee para diseñar la pieza, si no que pueden traspasar los límites de la intuición, otorgando a las piezas una forma geométrica muy compleja que en muchos casos, resulta inabordable por la mayoría de métodos tradicionales de fabricación. En este punto, la fabricación aditiva adquiere una gran importancia; pues su flexibilidad para generar geometrías complicadas, permite que la forma de las piezas optimizadas no sea un problema para que estas sean fabricadas.

Existen diversos métodos de optimización estructural, pero de entre todos ellos, el que ocupa este proyecto es el que se conoce como «Optimización de topología».

Optimización de topología

La optimización topológica se puede definir como un problema de optimización estructural, cuyo objetivo es encontrar la mejor distribución de material en un dominio de diseño, para unas restricciones y condiciones de contorno determinadas. La resolución de estos problemas se fundamenta en el método de los elementos finitos. En la optimización topológica, la distribución de material se establece en base a un vector de variables de diseño definidas como valor característico del elemento.

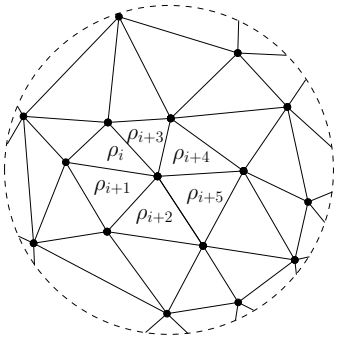


Figura 1.1: Ejemplo de dominio de diseño.

Para encontrar esa distribución de material óptima, se parte del dominio de diseño discretizado como el de la Figura 1.1 y a cada elemento de este dominio se le asigna una variable de diseño ρ_i, que tomará valores binarios 0 o 1.

$$\boldsymbol{\rho} = \begin{Bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_i \\ \vdots \\ \rho_n \end{Bmatrix}.$$

Este vector de variables de diseño entrará en un proceso iterativo que asignará a cada variable el valor correspondiente para minimizar la "compliance" (c), la cual se evalúa usando FEM.

De manera general, el problema de optimización topológica puede definirse de la siguiente manera:

$$\left.\begin{array}{rll} \min\limits_{\rho}: & c\left(\boldsymbol{\rho}\right) = \boldsymbol{u}^t \boldsymbol{K} \boldsymbol{u} \\ \text{sujeto a}: & \frac{V(\boldsymbol{\rho})}{V_o} = \alpha \\ & \boldsymbol{K} \cdot \boldsymbol{u} = \boldsymbol{f} \\ & \rho \in [0,1] \end{array}\right\}, \tag{1.1}$$

donde c es la función objetivo, $\boldsymbol{\rho}$ el vector de variables de diseño, $V(\boldsymbol{\rho})$ el volumen de material, V_o el volumen de partida, α el porcentaje de reducción de volumen, $\boldsymbol{u}$ el vector de desplazamientos, $\boldsymbol{K}$ la matriz de rigidez y $\boldsymbol{f}$ el vector de fuerzas externas.

Tras resolver este problema de optimización, los valores del vector de variables de diseño definirán la distribución de material, la cual puede ser representada como una

imagen rasterizada[1] como puede observarse en la Figura 1.2 para una viga empotrada - libre. Puede observarse que las formas obtenidas por este método, pueden ser muy

Figura 1.2: Ejemplo de viga empotrada - libre obtenida a través de OT.

complejas, tanto, que los métodos de fabricación convencional pueden no ser los más adecuados para fabricarlas, si no imposible. Aún así, la optimización topológica puede ser utilizada de diversas maneras:

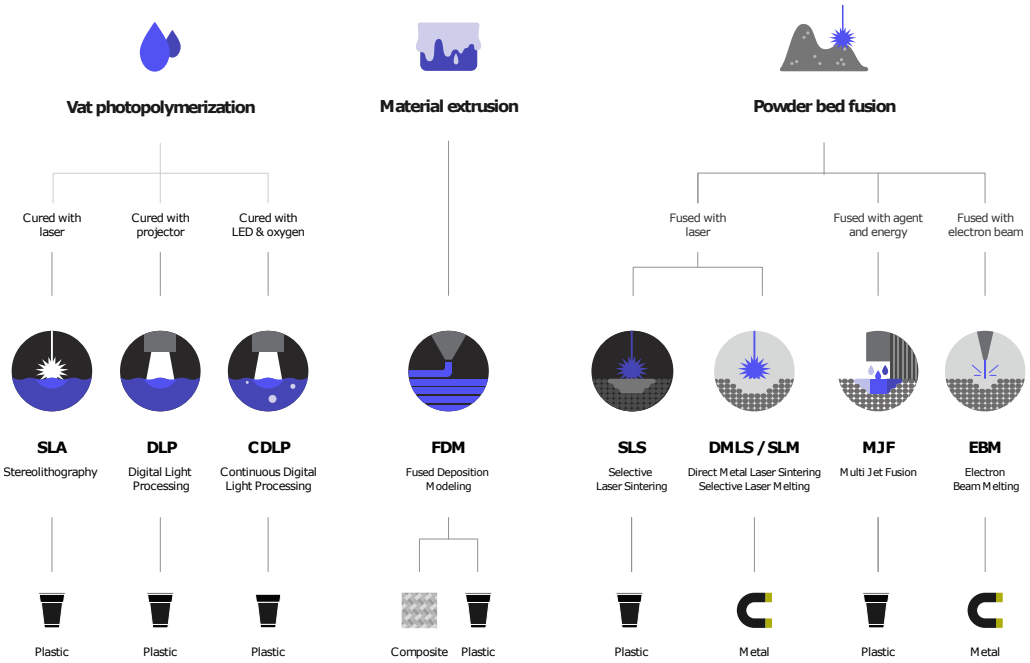
- Fabricando las piezas de manera directa utilizando métodos de fabricación no convencional, como fabricación aditiva.
- Usando los resultados como guía de dónde colocar el material, siendo ya conscientes del método de fabricación a emplear.

Fabricación Aditiva

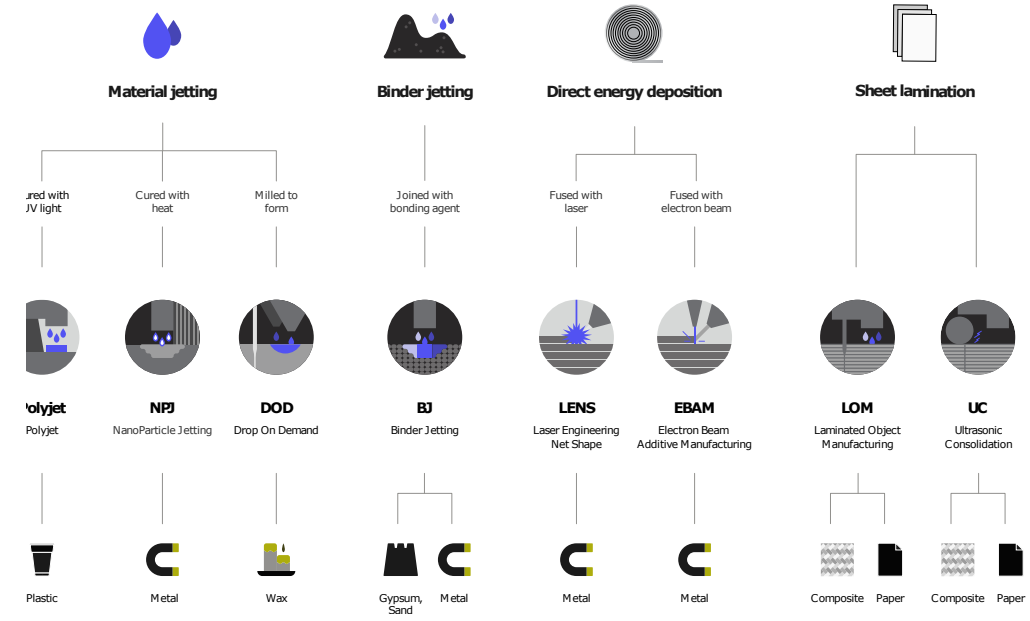
Según la norma UNE-EN ISO/ASTM 52900:2022 la fabricación aditiva puede definirse de la siguiente manera:

La fabricación aditiva (FA) es el término general que designa aquellas tecnologías que crean objetos físicos mediante adición sucesiva de material para crear objetos físicos según lo especificado por los datos de un modelo 3D. Estas tecnologías se utilizan actualmente para diversas aplicaciones en la industria de la ingeniería, así como en otras áreas de la sociedad, tales como medicina, educación, arquitectura, cartografía, juguetes y ocio.

[1] «rasterizada» indica que la imagen está formada por una serie de cuadros digitales que forman un mosaico. A cada cuadro digital se le asigna un valor numérico que puede relacionarse con el color de ese cuadro. Al colorear cada cuadro como corresponda, se dibuja la imagen.



(a)



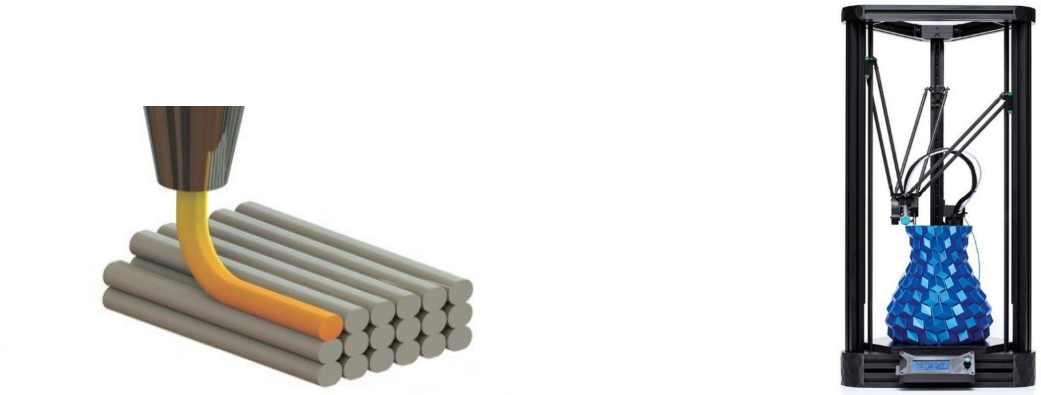
(b)

Figura 1.3: Tecnologías de fabricación aditiva [30].

Actualmente hay muchos tipos de máquinas que podrían entrar en la definición de fabricación aditiva, ya que hay una gran cantidad de empresas que aportan diversas

soluciones tecnológicas al mercado. En la Figura 1.3 se muestra una clasificación de las diferentes tecnologías que pueden encontrarse actualmente, de la cual se extrae la siguiente clasificación general:

- **Proyección de aglutinante; BJT (*binder jetting*)**: Este proceso consiste en la deposición de un aglutinanre sobre un lecho de polvo de manera selectiva, uniendo los materiales entre sí.
- **Deposición de energía focalizada; DED (*direct energy deposition*)**: Proceso que consiste en aportar material sólid, y a medida que este se deposita se funde aportando energía térmica focalizada en ese punto. Esto generalmente se hace mediante arco eléctrico o láser.
- **Extrusión de material; MEX (*material extrusion*)**: Este tipo de tecnología es la más popular a día de hoy, gracias a la muy conocida Fused Deposition Modelling (FDM), que se basa en depositar sobre la base de impresión, el material fundido capa a capa formando la pieza. Este material depositado se introduce en el fusor en forma de filamento o polvo. Lo más común es que se utilice para fabricar piezas de materiales poliméricos, aunque existen desarrollos para su uso con metales y material biológico (ver Figura 1.4).



(a) (b)

Figura 1.4: Impresoras FDM.

- **Proyección de material; MJT (*material jetting*)**: Proceso por el cual se depositan de manera selectiva gotas de materia prima. Este proceso es similar a la impresión tradicional en la que se depositan gotas de diferente color sobre un papel, la diferencia es que generalmente, el material utilizado es un polímero fotoreactivo que se solidifica mediante luz ultravioleta.

- **Fusión de lecho de polvo; PBF (*powder bed fusion*)**: Esta tipología trata de unir las partículas de polvo que se encuentran en la zona de impresión fundiéndolas mediante la aportación de energía térmica. El material base en polvo puede ser metal o polímero. Ejemplos de este tipo de tecnologías son: SLM (Selective laser melting), EBM (Electron beam melting), SLS (Selective Laser Sintering).
- **Laminado de hojas; SHL (*sheet lamination*)**: En este proceso, el material predispuesto en forma de láminas se une formando la pieza.
- **Fotopolimerización en tanque o cuba; VPP (*vat photopolymerization*)**: Se basa en solidificar un fotopolímero líquido que se encuentra en la zona de impresión mediante polimerización selectiva. De esta categoría, las tecnologías más famosas son SLA (Estereolitografía) y DLP (procesamiento digital de la luz). Ambas se basan en curar una resina líquida usando luz ultravioleta. La principal diferencia entre estas dos es que la primera utiliza un láser para curar el líquido, y la segunda proyecta una imagen 2D de la sección a imprimir (ver Figura 1.5).

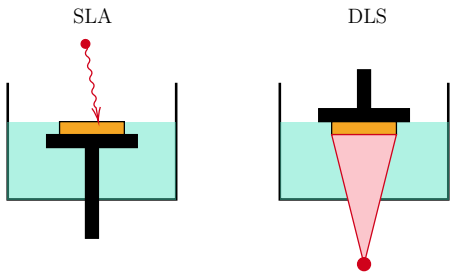


Figura 1.5: Comparativa entre DLS y SLA

Optimización de Topologías *Infill*

El término «*Infill*» es muy utilizado en fabricación aditiva y se refiere al formato de relleno poroso y regular de las celosías internas ("*lattice*"), que se utiliza para aligerar las piezas. Este formato de relleno puede adoptar diferentes formas geométricas y patrones, como los que se muestran en la Figura 1.6.

Esta manera de aligerar las piezas, tiene una gran influencia en las características mecánicas de la pieza. Generalmente, al usar rellenos con un bajo porcentaje de material, la resistencia de la pieza disminuye mucho. En consecuencia, al intentar fusionar las ideas de optimización topológica e *infill*, se busca encontrar el patrón de relleno óptimo para una pieza determinada.

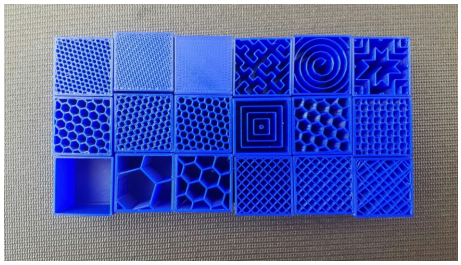


Figura 1.6: Ejemplos de diferentes tipologías de estructuras «*Infill*» [33].

En el artículo de Wu *et al.* [23], se propone una modificación del algoritmo convencional de optimización topológica, cuyo objetivo es calcular el problema de manera «local», haciendo que la distribución de masa adquiera una forma porosa. Los detalles de esta modificación se expondrán más adelante, pero el resultado obtenido por este método puede verse, por ejemplo en la Figura 1.7.

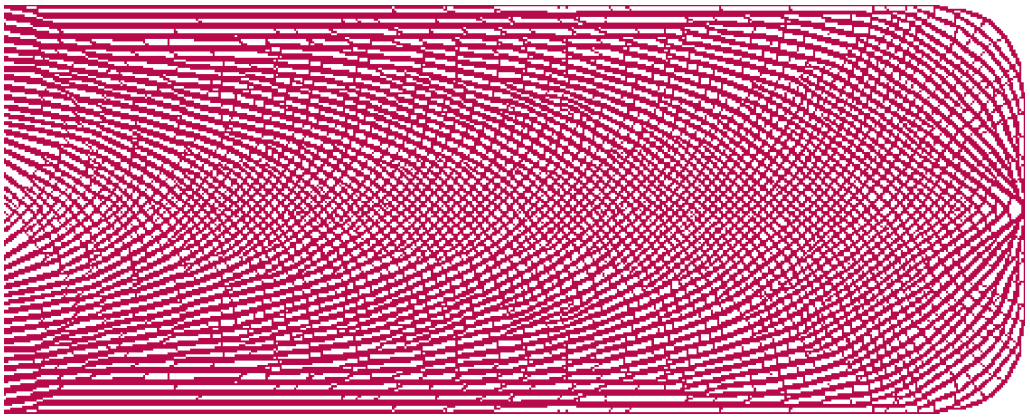


Figura 1.7: Ejemplo de viga *Infill* empotrada - libre.

Se observa como la solución presenta patrones de pequeño tamaño, que son más adecuados para aplicarlo al relleno de una pieza sin cambiar por completo su distribución de material. Esto, además, tiene como ventaja que la pieza sigue siendo aceptable si el estado de las solicitaciones mecánicas cambia, como por ejemplo, un cambio de dirección o de punto de aplicación.

Contexto

Si se comprueban los resultados que generan los algoritmos de optimización topológica (ver Figura 2.1(b)), se puede observar que estos forman un vector o matriz de valores comprendidos entre 0 y 1. Las imágenes en escala de color que se muestran,

no son mas que una forma visual de simbolizar lo que estos resultados representan, asociando a cada elemento un color de una escala (□ → 0; ■ → 1), dependiendo del valor asociado a cada elemento finito.

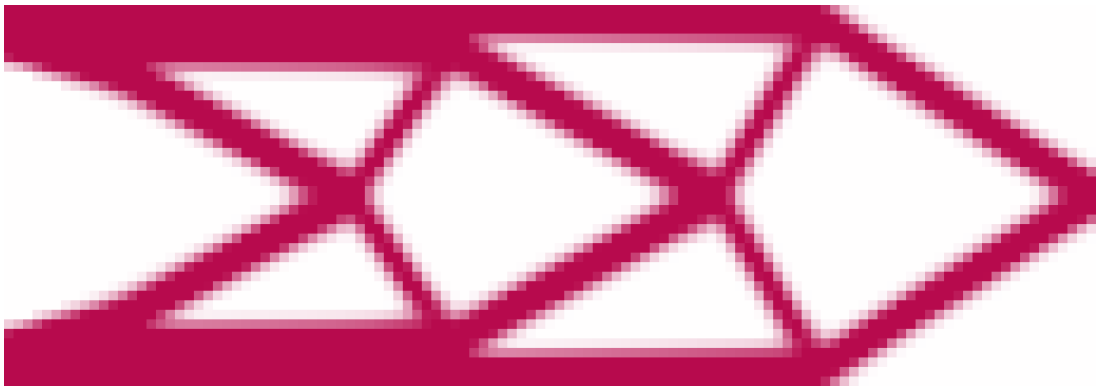


Figura 1.8: Ejemplo de resultado de optimización.

El problema que ocupa a este proyecto parte de la necesidad de materializar estos resultados a través de un proceso de fabricación. Esto implica que a partir de ese «vector» de resultados, debe extraerse un archivo que pueda ser usado para fabricar la pieza. El proceso a priori podría resultar sencillo, sobretodo si se piensa en ejemplos como el de la Figura 2.1(b). Podría usarse la imagen a color del resultado, y con un software CAD cualquiera, dibujar las líneas de los contornos hasta crear la pieza; pero esto tiene algunos inconvenientes. En primer lugar, y esta es la causa principal por la que se propuso hacer este proyecto, son los resultados tipo *Infill* como el de la Figura 1.7, archivo cuenta con más de 3000 agujeros que si tuvieran que ser dibujados por el usuario del software CAD "a mano alzada", podría tomar un tiempo inasumible.

Por otro lado hay que preguntarse, ¿cómo de sistemático puede llegar a ser el usuario a la hora de "dibujar" manualmente estos resultados? ¿Qué debe ser interpretado como parte de un agujero y qué no? ¿Cómo se puede asegurar manualmente que dos líneas diferentes sigan el mismo criterio para trazarse? Todas estas preguntas podrían parecer a priori fáciles de responder, pero como se comprobará a lo largo del proyecto, responderlas puede ser complicado. Además, dependiendo de la cantidad de elementos del problema de optimización y de cómo se responda a cada una de las cuestiones, el resultado tras la interpretación de resultados podría ser muy diferente.

Por último, aunque en este proyecto los algoritmos de optimización se utilizan a nivel investigación (no se parte de un algoritmo preprogramado por una marca comercial). Si se centra la atención en un usuario que los usa como diseñador, tener que dibujar manualmente los resultados podría ser bastante incómodo, sobretodo si

se tiene que reiterar en un diseño para mejorarlo. Su labor en este caso sería mucho más productiva, si se reduce su trabajo a hacer pequeños retoques sobre la pieza. De ahí la necesidad de disponer de un método veloz, que a partir de los resultados de optimización, construya un modelo CAD que el usuario pueda modificar fácilmente si fuera necesario.

Objetivos y alcance del trabajo

Para definir los objetivos de este proyecto, es necesario en primer lugar, observar cuales son las limitaciones del mismo. Generalmente se ha trabajado con algunos de los algoritmos de optimización que se usan en investigación, los cuales tienen unas limitaciones importantes:

1. Trabajan con elementos finitos bidimensionales, así que la interpretación estará restringida a problemas planos.

2. Los elementos finitos que discretizan el dominio, son todos del mismo tamaño y de forma cuadrada. Esto permite interpretar el vector de variables de diseño como una matriz de valores, donde la posición del elemento finito asociado, vendrá determinada por la posición del valor en la matriz.

Atendiendo a lo planteado, se propondrá un método para abordar la interpretación de los resultados, basado en algoritmos de procesamiento de imágenes, que obtenga un conjunto de listas con las posiciones de los puntos del dominio situados en los bordes materiales de la pieza. El método debe obtener una representación lo más fiel posible a la visualización de la matriz de partida y hacerlo de manera rápida. Además, estas coordenadas se exportarán a un software CAD para automatizar el proceso de generación de geometrías, con la idea de poder usar fabricación aditiva para obtener una pieza física.

Beneficios que aporta el trabajo

Para abordar esta cuestión, es necesario observar primero, bajo qué contexto se desarrolla la optimización topológica. Pues debido a la "novedad" de esta metodología de diseño, podría ser difícil tener una visión amplia de los beneficios potenciales que tiene, y por supuesto, sería más difícil entender el contexto en el que se desarrolla el trabajo y los beneficios que aporta.

En primer lugar, la creciente aplicación de la optimización de topología en los diferentes software CAD/CAM, podría considerarse coincidente o adherido a un cambio radical en el modo de producción, en lo que se conoce como «cuarta revolución industrial» o «industria 4.0». Autores como Daaboul *et al.* [19], Jiang *et al.* [22] o Bianchi [25], describen cómo, bajo esta nueva "revolución", el modo de producción está cambiando de un sistema llamado "de producción flexible", a un modelo de "personalización en masa". Esto indica que la tendencia de las industrias desarrolladas es de mantener una producción a gran escala, como venía haciéndose tradicionalmente, pero a su vez, que esta sea personalizada a los gustos y preferencias del consumidor. Es en esta línea donde la optimización topológica podría encontrar su lugar en la industria, ya que puede reducir drásticamente los plazos de desarrollo del producto, lo que se traduce en una reducción de los costes. El proceso automatizado suele dar lugar a piezas de mejor rendimiento en mucho menos tiempo del que se necesitaría con los métodos de diseño tradicionales. La optimización topológica podría entenderse, no como un sustituto del diseñador, si no como una extensión de su capacidad creativa, que le ayuda a elaborar diseños muy complejos de manera muy rápida y eficiente.

Por otra parte, en septiembre de 2015 la Asamblea General de las Naciones Unidas, establece un compromiso hacia la sostenibilidad económica, social y ambiental de los 193 Estados miembros, conocida como «Agenda 2030». La Agenda plantea 17 Objetivos con 169 metas que abarcan las esferas económica, social y ambiental (ver Figura 1.9).

De entre los 17 objetivos, para la finalidad de este apartado se pueden destacar dos de ellos, el 9: Industria, innovación e infraestructura y el 12: Producción y consumo responsables; de los cuales, estas metas podrían ser las más relevantes en lo que concierne a la OT:

9.4 *De aquí a 2030, modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la* ***adopción de tecnologías y procesos industriales limpios y ambientalmente racionales****, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas*

12.2 *De aquí a 2030, lograr la gestión sostenible y el* ***uso eficiente de los recursos naturales***

12.5 *De aquí a 2030,* ***reducir considerablemente la generación de desechos mediante actividades de prevención****, reducción, reciclado y reutilización*

Mediante optimización topológica se puede determinar la mejor distribución de material posible, para una pieza bajo un estado de carga concreto. Esto encaja muy bien con algunos de los objetivos planteados por la Agenda 2030, ya que "determinar

`https://www.un.org/sustainabledevelopment/es/`

Figura 1.9: Objetivos de desarrollo sostenible.

la mejor distribución de material" es equivalente a "hacer un aprovechamiento más eficiente del mismo", con sus correspondientes beneficios en materia económica y de sostenibilidad. En negrita se resaltan las ideas que mejor podrían combinar con la adopción de la optimización topológica como metodología de diseño. Además, la optimización topológica en cierto modo, tiene muy buena sinergia con la fabricación aditiva, permitiendo así fabricar piezas con unos desechos muy moderados o incluso nulos. Esta relación simbiótica se debe a que por un lado, la AM es muy flexible en cuanto a la complejidad geométrica que admite, mientras que la OT, mediante restricciones, puede generar piezas que se adapten a las limitaciones de fabricación que la AM pudiera tener; eliminando por completo el uso de soportes y en consecuencia, la generación de desechos. A esto se le debe añadir que la optimización topológica en esencia está llamada a convertirse en una herramienta complementaria a la creatividad del diseñador, haciendo que este no requiera repetir de manera reiterativa pruebas de diseño para verificar que se cumplan los requisitos estructurales, y además, restando peso a la necesidad de poseer un conocimiento intuitivo, resultado de una amplia experiencia práctica. Esto no solo implica una mejora de la productividad del diseñador, si no que facilita la posibilidad de crear diseños mucho más complejos e innovadores con un mayor valor añadido (ODS 8).

Por último, una vez presentados de manera resumida, la importancia que podría

tener la optimización topológica en general, cabe preguntarse qué aporta el presente trabajo. Como bien se ha explicado a la hora de contextualizar este proyecto, una dificultad que podría considerarse en cierto modo un defecto de esta metodología de diseño, es que los resultados de optimización topológica no se expresan de un modo "comprensible" por un software CAD, y en consecuencia, deben ser "traducidos" a un formato "comprensible". Por tanto, la intención aquí es automatizar la generación de archivos CAD a partir de la solución del proceso de optimización, cosa que no es ni mucho menos inmediata. El algoritmo propuesto para hacer esto, reduce considerablemente el tiempo de interpretación de resultados y facilita el proceso de diseño y/o fabricación. Por otro lado, limita la interacción del diseñador con el resultado de optimización a la hora de realizar la interpretación, haciendo que los archivos CAD generados sean una exégesis completamente sistemática de los resultados. Además, el modelo CAD puede usarse en el software correspondiente, para modificarlo o incorporarlo junto a otros diseños, cosa que no sería posible de otro modo.

Selección/Descripción de la solución propuesta

El algoritmo propuesto como solución de este trabajo, está basado en el procesamiento de imágenes en mapa de bits. Este se divide en las etapas mostradas en la Figura 1.10. Se comienza con el resultado de la optimización topológica, el cual es un vector que contiene las variables de diseño con valores contenidos entre 0 y 1.

Primero, se aplica un filtrado para convertir esos valores a términos binarios[2], para ello simplemente los elementos cuyo valor sea mayor a 0,5 se cambia a valor 1. A continuación, se aplica un segundo filtro que ayuda a distinguir cada agujero, para ello, los valores asociados a los elementos de un agujero, se cambian por un valor $n = 10, 20, 30, 40, ...$

Ubicado cada agujero, se extraen los puntos límite de cada uno de ellos, y se guardan por separado. Como estos puntos presentan muchas irregularidades, se aplica un promediado para suavizar las líneas y redondear las esquinas.

Tras esto, los contornos presentan un número de datos innecesario, que solo hace complicar la posterior representación en formato CAD. Por esto, se aplica un algoritmo de simplificación, que ayude a representar únicamente la información necesaria.

Por último, estos puntos extraídos, se exportan a FreeCAD para representar la

[2]Binario en este contexto quiere decir que un valor que es continuo $[0, 1]$, se va a cambiar por un valor discreto que puede ser $\{0, 1\}$

pieza final.

Análisis de alternativas

En el desarrollo de este proyecto se han considerado diversos métodos alternativos para afrontar el problema que lo ocupa. Muchos de estos métodos se han llevado a la práctica, solo con el objetivo de compararlos con la solución adoptada.

Método de ordenación de puntos del contorno

Para generar el trazado de los contornos, se ha usado un algoritmo que se "autoguía" sobre los límites entre los dominios sólidos y vacíos; colocando los puntos de manera ordenada conforme avanza. Pero esta no fue la primera idea considerada para hacer esto.

La primera idea para generar los contornos, fue guardar de manera desordenada las coordenadas de los elementos límite. Esto se puede hacer de manera bastante sencilla con un filtro, que sume a cada elemento, el valor de los elementos que le rodean. Los elementos de regiones sólidas tendrán un valor de 9 tras el filtrado, porque todos los elementos que le rodean tienen valor 1; mientras que las regiones vacías tendrán valor 0. Aquellos elementos con valor $0 < v < 9$, formarían parte del contorno.

Obtenidas esas coordenadas, se calcula el centro de masas de ese conjunto de puntos, que se utiliza para hacer un cambio a coordenadas polares de cada uno de ellos. Ordenando los ángulos de menor a mayor, se obtendría un conjunto más o menos ordenado de datos.

Este conjunto de datos no es un trazo en si, mas bien, es una nube de los puntos que rodean el contorno. Para convertir esta "nube de puntos" a un trazado único, se usó una aproximación por mínimos cuadrados, para adaptar una sucesión de polinomios cúbicos a ese conjunto de coordenadas.

Aunque esta alternativa fue programada de diferentes maneras, resulta muy poco robusta como para adoptarla como método final. Esto es debido a que usar las coordenadas polares para ordenar los puntos del contorno, puede causar errores en agujeros muy alargados. Además de que en general, los resultados eran bastante inestables.

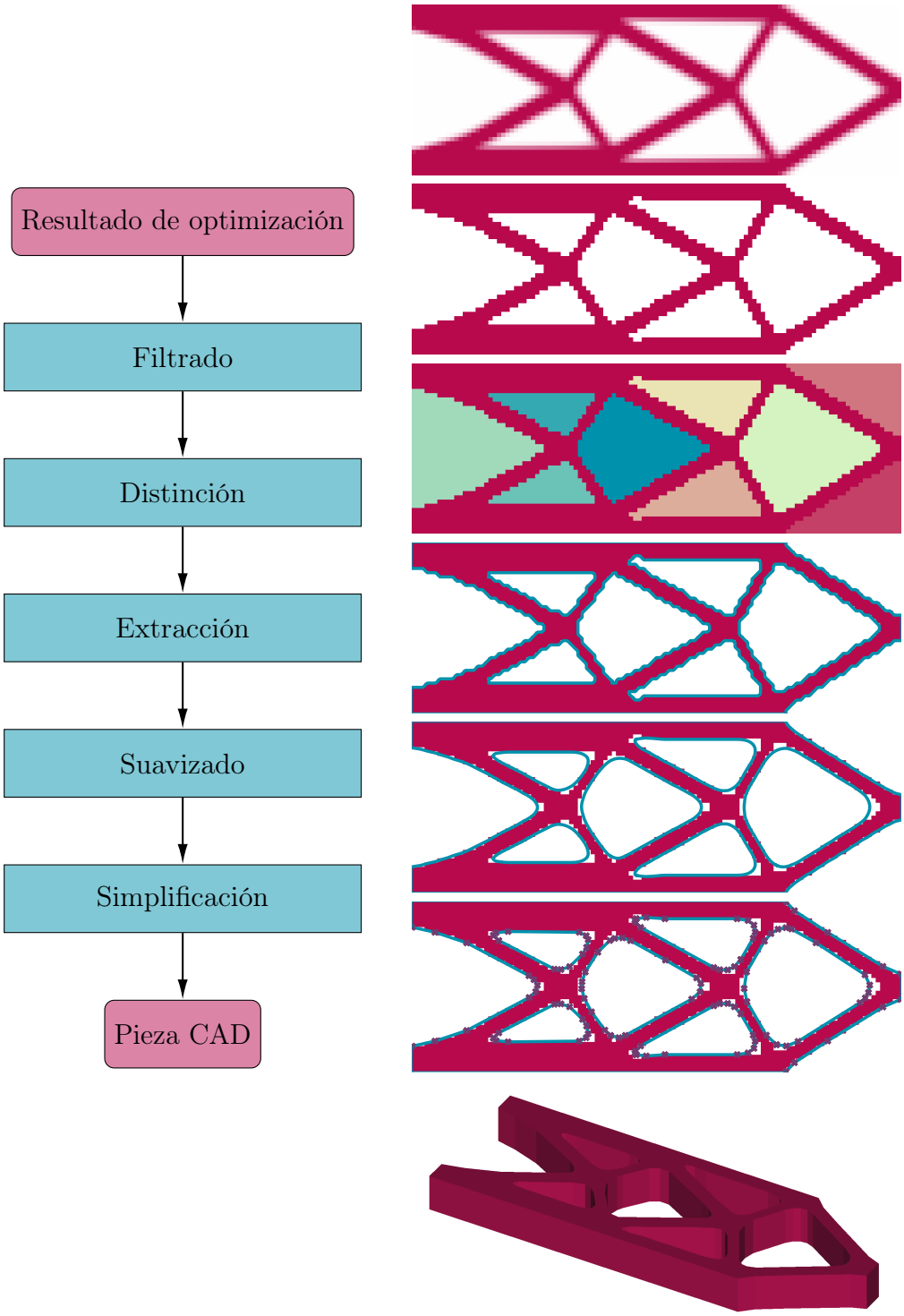


Figura 1.10: Etapas del algoritmo para exportar resultados a CAD.

Métodos de suavizado

El método de suavizado elegido está basado en una media móvil. Pero también se consideró utilizar curvas de spline cúbicas para generar los contornos.

Estas se descartaron por varios motivos:

- Frente a un algoritmo basado en promediar la línea, generar un método robusto que aproxime el contorno mediante curvas de spline, podría ser una tarea muy compleja. Insistir en el uso de este método no sería rentable si se tiene otro, que resulta igualmente efectivo, y con una programación mucho más sencilla.
- Exportar las curvas de spline a FreeCAD podría hacerse de dos maneras diferentes:
    1. Exportando directamente los puntos de la curva, siendo equivalente a lo conseguido mediante el promediado de la línea.
    2. Exportando los nudos del spline. Esta opción puede ser interesante, pero se tendría que revisar en profundidad la documentación de FreeCAD para hacer esto adecuadamente, ya que dependiendo del software, la manera de definir estos nudos por parte del usuario puede variar.

Métodos de trazado

Los método de trazado de contornos, se explican en profundidad en el Capítulo 3. Ahí se expone que estos utilizan un operador que evalúa los elementos circundantes. En base a los valores de los elementos sobre los que se sitúa el operador, se toman unas decisiones de avance u otras.

En el proyecto se ha creado un operador propio para hacer esto, pero también se ha programado el mismo algoritmo usando el clásico operador del método de Theo Pavlidis. Aunque ambos métodos resultan igual de rápidos, mediante el operador usado en este proyecto, resulta más sencillo dar un tratamiento adecuado a contornos pequeños.

Capítulo 1

Análisis del estado del arte

Contenido

1.1. Formulación del problema de optimización

Como ya se ha mencionado, la optimización topológica trata de encontrar la mejor distribución de material en un dominio Ω, para unas condiciones de contorno determinadas (ver Figura 1.1). Para ello, se discretiza el dominio Ω mediante elementos finitos y a cada elemento, se le asigna una variable de diseño ρ_i que toma valores binarios (0 o 1). Con esto se pretende describir las regiones del dominio que son sólidas Ω_m y las regiones vacías Ω_v.

Para determinar las zonas con y sin material, se define la función $\rho(\boldsymbol{x})$:

$$\begin{cases} \rho(\boldsymbol{x}) = 1 & \forall \boldsymbol{x} \in \Omega_m, \\ \rho(\boldsymbol{x}) = 0 & \forall \boldsymbol{x} \in \Omega_v. \end{cases} \tag{1.1}$$

Considerando que el material es isotrópico, lineal y que el sólido se ve solicitado bajo un único estado de cargas; el problema de optimización se formula en base a

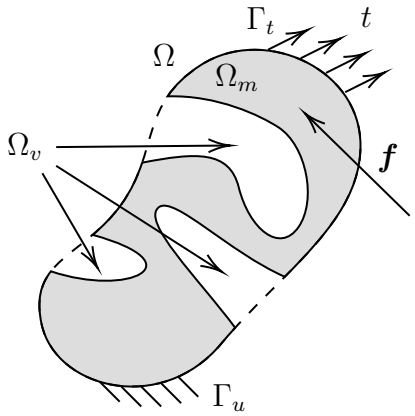


Figura 1.1: Dominio de diseño.

minimizar el trabajo de las cargas externas. Esta función se define de la siguiente manera:

$$c(\boldsymbol{x}) = \int_{\Omega_m} \boldsymbol{f}^t \boldsymbol{u} d\Omega + \int_{\Gamma_t} \boldsymbol{t}^t \boldsymbol{u} d\Gamma_t = \int_{\Omega} \rho(\boldsymbol{x}) \boldsymbol{f}^t \boldsymbol{u} d\Omega + \int_{\Gamma_t} \boldsymbol{t}^t \boldsymbol{u} d\Gamma_t, \tag{1.2}$$

donde $\boldsymbol{f}$ es el vector de cargas externas, $\boldsymbol{u}$ el vector de desplazamientos y $\boldsymbol{t}$ el vector de cargas superficiales aplicadas en el contorno Γ_t. Además, se considera que las condiciones de contorno impuestas sobre los desplazamientos Γ_u, son homogéneas.

El problema de optimización se escribe como sigue:

$$\left.\begin{array}{ll} \underset{\rho(\boldsymbol{x})\in\{0,1\}}{\text{mín}} : & c\left(\boldsymbol{u}\right) \\ & \\ \text{sujeto a :} & \int_\Omega \rho(\boldsymbol{x}) d\Omega \leq V_{max} \\ & \boldsymbol{u} \rightarrow \text{Solución del problema elástico - lineal.} \end{array}\right\} \tag{1.3}$$

Para plantear un algoritmo de resolución de este problema de optimización, el dominio se discretiza por elementos finitos, y la variable $\rho(\boldsymbol{x})$, se define como un vector de variables de diseño $\boldsymbol{\rho}$ que contiene el valor ρ_i asociado a cada elemento finito. Por tanto, de cara a la resolución numérica del problema de optimización topológica, este se reformula de la siguiente manera:

$$\left.\begin{array}{ll} \underset{\rho}{\text{mín}} : & c\left(\boldsymbol{u}\right) = \boldsymbol{u}^t \boldsymbol{K} \boldsymbol{u} \\ & \\ \text{sujeto a :} & \frac{V(\boldsymbol{\rho})}{V_o} = \alpha \\ & \boldsymbol{K} \cdot \boldsymbol{u} = \boldsymbol{f} \\ & \rho \in \{0,1\} \end{array}\right\}, \tag{1.4}$$

donde c es la función objetivo, $\boldsymbol{\rho}$ el vector de variables de diseño, $V(\boldsymbol{\rho})$ el volumen de material, V_o el volumen de partida, α el porcentaje de reducción de volumen, $\boldsymbol{u}$ el vector de desplazamientos, $\boldsymbol{K}$ la matriz de rigidez y $\boldsymbol{f}$ el vector de fuerzas externas.

Aunque este planteamiento permita resolver el problema de manera computacional, presenta algunas dificultades importantes a tener en cuenta:

- El problema está mal condicionado y la existencia de solución no está asegurada, implicando que la resolución pueda ser inestable. Esta inestabilidad puede presentarse de varias maneras diferentes, una de ellas, es que a medida que aumenta el tamaño del modelo computacional, tiende a aumentar el número de agujeros, es decir, aparece una dependencia del mallado. Esto, por supuesto, es un fenómeno no deseado porque de un aumento de la discretización, no resultaría una solución más detallada.

  Otra inestabilidad típica es lo que se conoce como *The checkerboard problem* (Figura 1.3), donde se aprecia como la solución adopta un formato de tablero de ajedrez, alternando elementos sólidos y vacíos. Este fenómeno se debe a la naturaleza de la modelización por elementos finitos, que tiende a sobrestimar la rigidez de este tipo de patrones, cuando lo cierto es que esta alternancia entre elementos sólidos y vacíos no es interpretable físicamente.

  Para abordar estos problemas se suelen proponer dos alternativas:

  - A la primera alternativa se le conoce como *métodos de homogeneización*. Estos métodos se basan en el concepto de *relajación*, que trata de agrandar el dominio de posibles valores de las variables de diseño. Para ello se define un material compuesto poroso, formado por celdas periódicas con agujeros microscópicos, de manera que, calculando el tamaño y orientación de estos agujeros, se resolvería el problema de distribución óptima de material. Autores como Bendsøe y Kikuchi [3], trabajaron en esta alternativa.
  - La segunda alternativa se basa en imponer restricciones a las variables de diseño, limitando los posibles valores que estos puedan tomar. Dentro de este método se encuentran los métodos *SIMP*, desarrollada por Bendsøe [4].

- Otra dificultad a tener en cuenta es que la variable de diseño ρ_i es binaria. Debido a que los algoritmos como *SIMP*, son métodos no heurísticos, es necesario que esta variable pueda tomar valores comprendidos entre 0 y 1 (variable continua). Para resolver esto se pueden adoptar enfoques de penalización de valores intermedios o bien, relajar el espacio de diseño introduciendo un material poroso.

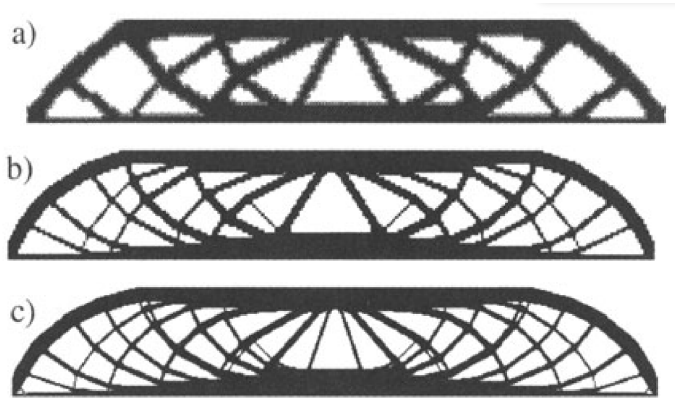


Figura 1.2: Muestra de la dependencia que los resultados de optimización para una viga MBB, donde en a) se muestra el problema teórico, en b) el resultado con 400 elementos y en c) con 6400 elementos [15].

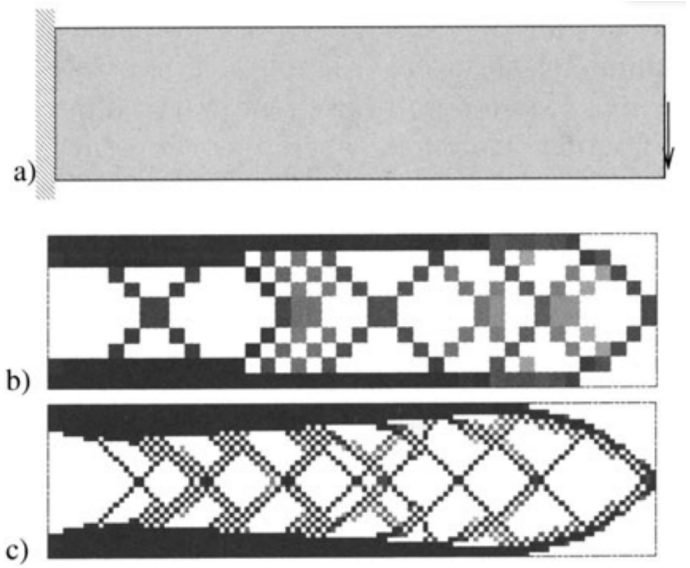


Figura 1.3: Muestra del *The checkerboard problem* en una viga empotrada - libre, en función de la discretización del dominio. Con 2700 elementos en a), 4800 en b) y 17200 en c) [15].

Hoy en día también se desarrollan enfoques basados en variable binaria, como los trabajos de Beckers [10]. Otros autores como Hajela y Lee [7] proponen métodos de optimización heurística como los algoritmos genéticos.

1.1.1. Método *SIMP* (Material Sólido Isótropo con Penalización)

Este método de optimización propuesto por Bendsøe [4], se plantea como alternativa a los *métodos de homogeneización*. Se basa en relajar la variable de diseño permitiéndole adoptar valores intermedios, siendo ahora una variable continua. Pero como se busca que en la solución final el resultado tienda a una imagen binaria, los valores intermedios se penalizan, para evitar que aparezcan grandes zonas con $\rho_i \in (0, 1)$.

La penalización se introduce usando una función de interpolación que evalúa las propiedades del material como densidad relativa del material elevada a una potencia p y multiplicada por las propiedades del material sólido isotrópico:

$$E_e(\rho_e) = E_{min} + (E_0 - E_{min}) \cdot \rho_e^p, \tag{1.5}$$

donde valores de $p > 1$, provocan que las densidades intermedias sean ineficientes y vayan desplazándose hacia el valor nulo, castigando el valor del módulo de Young correspondiente.

La ley de variación el módulo de Young planteada en la Ecuación 1.5, puede ser discutible y de hecho, ha sido objeto de debate. Esto se debe a que esta variación de la rigidez en función de la densidad no tiene un sentido físico, solo responde a requerimientos computacionales. En el artículo [11] se discute este problema, comparando esta formulación con diferentes modelos microestructurales. Se concluye afirmando que el modelo planteado en la Ecuación 1.5 puede ser válido siempre y cuando se ignoren los resultados de densidad intermedia, por tanto se puede optar simplemente por no considerar estos elementos.

A pesar de lo expuesto en el párrafo anterior, el hecho de tener elementos de densidad intermedia sigue siendo una problemática a resolver por el método *SIMP*, sobretodo a la hora de interpretar los resultados. Se han propuesto múltiples modificaciones al método, en las que se introduce lo que se conoce como *projection methods*. Estos "métodos de proyección" tratan de eliminar esas zonas de transición "grises" abordándolo desde diferentes perspectivas. Sigmund [17] hace una comparativa de estos métodos y propone una alternativa basada en operadores de morfología. Por otro lado, Guest *et al.* [16], proponen un fitro basado en la función escalón de

Heaviside. A día de hoy, el más utilizado es el que propone Wang *et al.* [18], basado en la "proyección de Heaviside", aunque con una nueva formulación:

$$\bar{\rho}_i = \frac{\tanh(\beta\mu) + \tanh(\beta(\rho_i - \mu))}{\tanh(\beta\mu) + \tanh(\beta(1 - \mu))}, \tag{1.6}$$

esta ecuación contiene dos parámetros de ajuste: $\beta \in [1, \infty)$ y $\mu \in [0, 1]$. Como se observa en la Figura 1.4, «β» puede considerarse como factor de escarpado, debido que al aumentar su valor, el filtro pasa a parecerse más a un escalón de Heaviside teórico. Por otro lado, «μ» controla el lugar donde se dará este escalón.

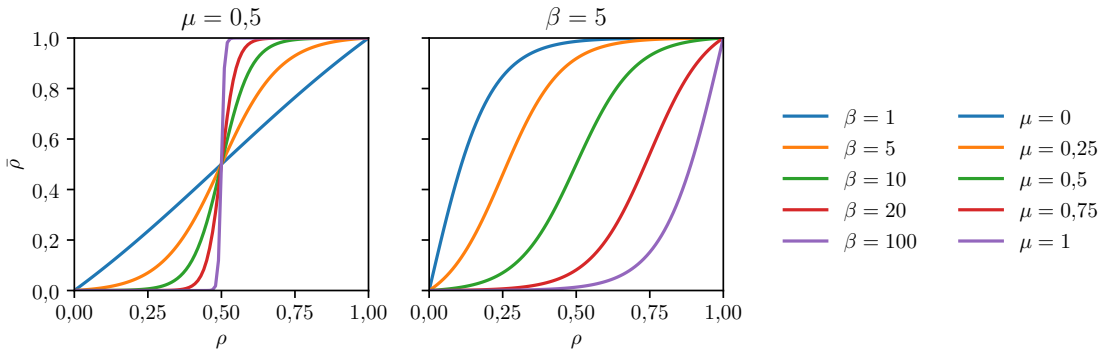


Figura 1.4: Parámetros de ajuste de la Ecuación 1.6.

El uso de este filtro se hace de manera gradual, se comienza con un valor de «$\beta = 1$», y a medida que se va alcanzando la solución, este parámetro se incrementa. Esto hace que en las primeras etapas de trabajo del algoritmo, las variables de diseño tengan un mayor número de valores posibles a tomar y tan pronto como se vaya alcanzando la solución final, se aumenta el valor de «β», haciendo que la restricción sea cada vez más estricta.

1.1.2. Implementación del algoritmo de optimización topológica

El algoritmo de optimización topológica se resume en la Figura 1.5, donde el primer paso de este proceso es la definición del problema, en el que se especifican las condiciones de contorno y se inicializan las variables de diseño. Tras esto, se inicia el proceso iterativo, que acaba cuando se cumple con el criterio de convergencia.

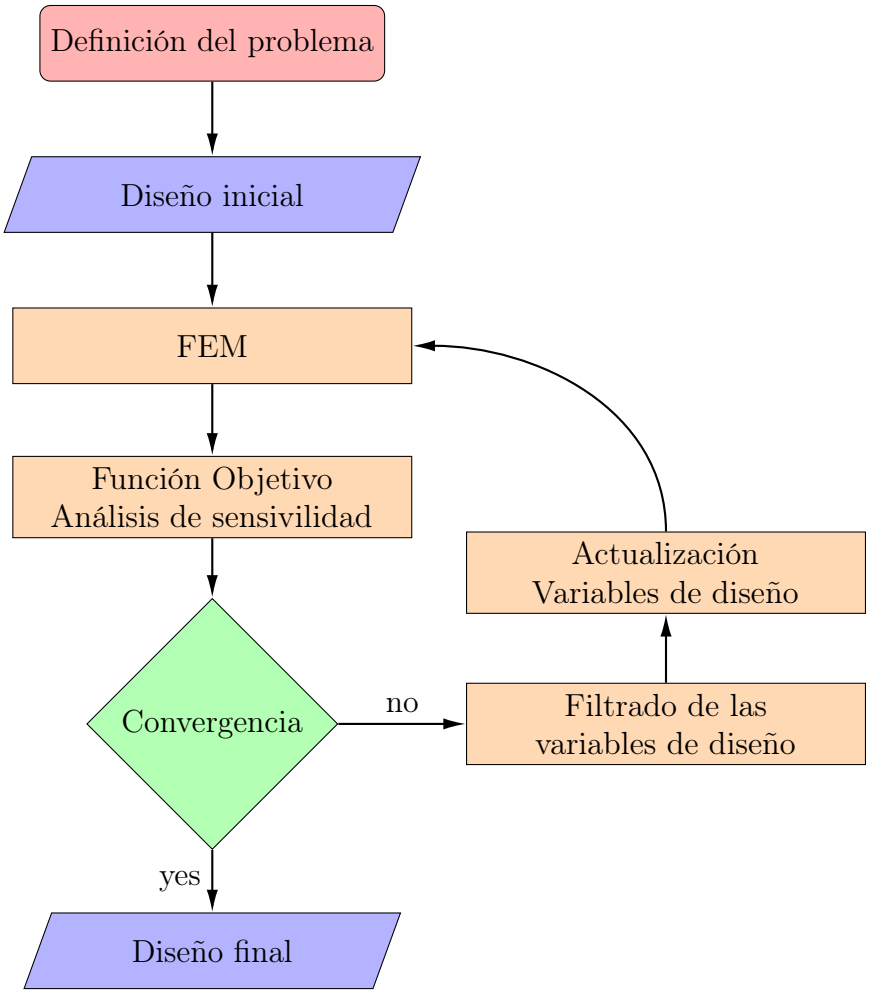


Figura 1.5: Diagrama de flujo del algoritmo de optimización topológica.

Análisis de sensibilidad

El análisis de sensibilidad generalmente trata de determinar qué efectos tendrá sobre la solución óptima el cambio de los parámetros del modelo.

En lo referido a optimización topológica, se define como el cálculo de las variaciones de la respuesta de la estructura a las cargas aplicadas, originada por modificaciones en alguna de las variables que configuran el diseño de esa estructura. Conociendo esta relación, se puede decidir como variar las variables de diseño para que la solución converja al valor óptimo. Normalmente se adopta un enfoque basado en gradiente, por el cual, se derivan la función objetivo y las diferentes restricciones en función de las variables de diseño.

Según Choi y Kim [32], el análisis de sensibilidad puede clasificarse según tres enfoques diferentes: Enfoque de aproximación, discreto y continuo.

- En el enfoque de aproximación, las sensibilidades se obtienen mediante el método de las diferencias finitas centrales o progresivas.
- El enfoque discreto toma como variables de diseño los términos de la matriz que gobierne el problema (si es un problema elástico, tomará los valores de la matriz de rigidez).
    - Si los términos de las matrices se derivan de manera explícita con respecto a las variables de diseño, se clasifica como método analítico.
    - Si lo anterior se realizan usando el método de las diferencias finitas, se le categoriza como método semianalítico.
- El enfoque continuo es aquel en el que la sensibilidad se resuelve en variable continua.
    - Si el dominio es continuo y las ecuaciones de sensibilidad se modelizan como problema continuo, se le denomina enfoque continuo - continuo.
    - Si la diferenciación se resuelve en el dominio continuo, pero se discretiza, se le clasifica como enfoque continuo-discreto

En los algoritmos de optimización topológica utilizados para realizar el presente trabajo, se utiliza un enfoque discreto analítico. El problema se discretiza primero mediante elementos finitos y en base a dicha discretización se estudia la sensibilidad con respecto a las variables de diseño.

Filtrado

Mediante este proceso se resuelven dos problemáticas de los algoritmos de optimización topológica: El llamado *The checkerboard problem* y la dependencia del mallado.

Los primeros algoritmos de optimización topológica como por ejemplo Sigmund [13], utilizaban un enfoque basado en el procesamiento de imágenes. Este enfoque trata de utilizar la convolución para filtrar las variables de diseño:

$$\tilde{\rho}(\boldsymbol{x}) = (F * \rho)(\boldsymbol{x}) = \int_{\mathbb{B}_{\mathcal{R}}} F(\boldsymbol{x} - \boldsymbol{y})\rho(\boldsymbol{y})\, d\boldsymbol{y}, \tag{1.7}$$

donde $\mathbb{B}_{\mathcal{R}}$ es una esfera en $3D$ o un círculo en $2D$, con centro en $\boldsymbol{x}$ y radio $\mathcal{R}$. Esta operación se hace de manera discreta a través de un operador matricial llamado *kernel*:

$$\mathbf{k} = \frac{1}{n^2}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}. \tag{1.8}$$

Definiendo la matriz $\boldsymbol{P} \in \mathcal{M}_{n\times n}(\mathbb{B}_{\mathcal{R}})$, donde cada posición de $\boldsymbol{P}$, contiene el valor de la variable a filtrar correspondiente a un elemento incluido en $\mathbb{B}_{\mathcal{R}}$:

$$\tilde{\rho}(\boldsymbol{x}) = \sum_{i=1}^{n}\sum_{j=1}^{n} P_{ij} \cdot k_{ij}. \tag{1.9}$$

Esta operación se entiende de manera más clara junto a la Figura 1.6, donde se observa que dependiendo del radio $\mathcal{R}$, se escogen un conjunto de elementos $\mathbb{B}_{\mathcal{R}}$.

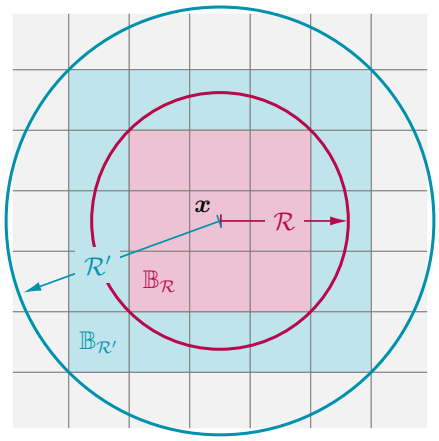


Figura 1.6: Funcionamiento de la operación de convolución aplicado en optimización topológica.

La matriz (1.8) puede definirse de muchas manera diferentes dependiendo del tipo de operación[1]. La opción mostrada es la que usualmente se emplea para los algoritmos de optimización topológica, conocido como filtro lineal, aunque en ocasiones se empleen filtros gausianos.

Sobre la base de lo explicado anteriormente, se asientan las diferentes operaciones de filtrado usadas en optimización topológica. De entre estas se pueden destacar dos

[1]En Kinser [27] se detallan los diferentes operadores y aplicaciones para procesamiento de imágenes.

por su amplio uso; filtros de densidad, propuestos por Bruns y Tortorelli [12] y filtros de sensibilidad, propuestos por Sigmund [9].

Los **filtros de densidad** trabajan directamente modificando la densidad del elemento, definiendo la variable de diseño filtrada, que sustituye a la variable original en el proceso de optimización. Esta densidad filtrada se define como;

$$\tilde{\rho}_e = \frac{\sum_{i \in N_e} \omega(\boldsymbol{x_i}) \nu_i \rho_i}{\sum_{i \in N_e} \omega(\boldsymbol{x_i}) \nu_i}, \tag{1.10}$$

donde:

- $N_e = \{i \mid ||\boldsymbol{x}_i - \boldsymbol{x}_e|| \leq \mathcal{R}\}$,
- $\omega(\boldsymbol{x}_i) = \mathcal{R} - ||\boldsymbol{x}_i - \boldsymbol{x}_e||$ (filtro lineal),
- $\omega(\boldsymbol{x}_i) = e^{-\frac{1}{2}\left(\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_e\|}{\mathcal{R}/3}\right)^2}$ (filtro gausiano),
- e designa el elemento sobre el que se aplica el filtro,
- ν_i es el volumen del elemento.

Cuando este filtro se aplica, a las densidades originales ρ_i se les llama variables de diseño, y a la densidad filtrada $\tilde{\rho}_i$, densidades físicas, resaltando el hecho de que ρ_i deja de tener sentido físico. En consecuencia, cuando se presente el resultado del proceso de optimización, la solución será $\tilde{\rho}_i$, las densidades físicas. Por otro lado, esta redefinición de las densidades del elemento, complica levemente el proceso de análisis de sensibilidad, debido a que las derivadas ya no solo dependen de las variables de diseño, si no también del proceso de filtrado. Entonces, teniendo en cuenta (1.10), el proceso de derivación se escribirá de la siguiente manera, de acuerdo a la regla de la cadena:

$$\frac{\partial \psi}{\partial \rho_j} = \sum_{e \in N_j} \frac{\partial \psi}{\partial \tilde{\rho}_e} \frac{\partial \tilde{\rho}_e}{\partial \rho_j} = \sum_{e \in N_j} \frac{\omega(\boldsymbol{x}_i)\nu_i}{\sum_{i \in N_e} \omega(\boldsymbol{x}_j)\nu_j} \frac{\partial \psi}{\partial \tilde{\rho}_e}, \tag{1.11}$$

donde ψ representa: la función objetivo c o el volumen de material V.

Los **Filtros de sensibilidad** trabajan de manera algo diferente, pues en vez de definir una nueva variable de diseño, opta por modificar la sensibilidad directamente:

$$\widehat{\frac{\partial c}{\partial \rho_e}} = \frac{1}{\text{máx}(\gamma, \rho_e) \sum_{i \in N_e} \omega(\boldsymbol{x}_i)\nu_i} \sum_{i \in N_e} \omega(\boldsymbol{x}_i)\nu_i \rho_i \frac{\partial c}{\partial \rho_i}, \tag{1.12}$$

donde el término γ normalmente tiene valor 10^{-3} y se utiliza para definir un valor mínimo que impida la división por cero.

Filtros PDE

Además del enfoque basado en convolución, existe una alternativa propuesta por Lazarov y Sigmund [20], que cambia por completo la concepción del proceso de filtrado y viene a solventar algunos defectos del filtro de convolución. Estos defectos son la distinción de dominios y la problematica con dominios no convexos. Para ejemplificar esto debe mirarse la Figura 1.7, donde se obserca como al aplicar el filtro de convolución, este atribuiría al elemento, el valor de los elementos del conjunto de elementos $\mathbb{D}_a$ y $\mathbb{D}_b$, pero este último conjunto no cumple realmente las condiciones conectivas con el elemento al que se le aplica al filtro, y no deberían tenerse en cuenta.

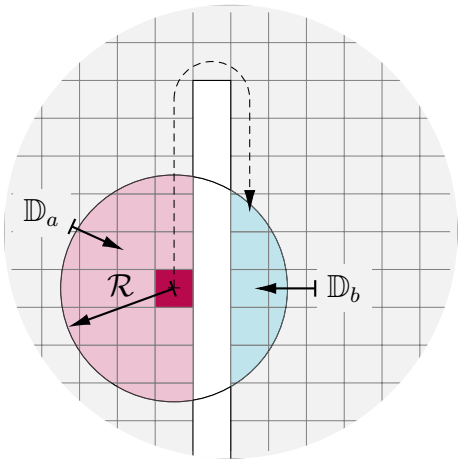


Figura 1.7: Ejemplo de filtro aplicado a dominio no convexo.

Debido a lo anterior y también debido al alto coste computacional que tiene inicialmente el método a medida que crece el radio, se plantea una alternativa basada en un concepto diferente, los filtros PDE. Se parte de que el valor de la variable filtrada, es solución de una ecuación diferencial de Helmholtz:

$$-r^2\nabla^2\tilde{\rho}(\boldsymbol{x}) + \tilde{\rho}(\boldsymbol{x}) = \rho(\boldsymbol{x}), \tag{1.13}$$

con condiciones de frontera de Neumann homogéneas,

$$\frac{\partial\tilde{\rho}(\boldsymbol{x})}{\partial\boldsymbol{n}}, \tag{1.14}$$

siendo $\boldsymbol{n}$ un vector unitario normal al dominio. Este enfoque acepta una resolución mediante el método de los elementos finitos, haciendo:

$$\tilde{\rho}(\boldsymbol{x}) = \boldsymbol{N}_e(\boldsymbol{x})\tilde{\boldsymbol{\rho}}_e, \tag{1.15}$$

donde $\boldsymbol{N}_e(\boldsymbol{x})$ es el vector de funciones de interpolación y $\tilde{\boldsymbol{\rho}}_e$, el vector de densidades filtradas en los nudos del elemento. Aplicando el método de los residuos ponderados sobre (1.13):

$$\boldsymbol{K}_f \tilde{\boldsymbol{\rho}} = \boldsymbol{p}_f, \tag{1.16}$$

donde $\boldsymbol{K}_f = \sum_{i \in \mathbb{N}_e} \boldsymbol{K}_{f,i}$, $\boldsymbol{p}_f = \sum_{i \in \mathbb{N}_e} \boldsymbol{p}_{f,i}$ y $\sum$ denota la operación de expansión y ensamblaje. Las matrices y vectores de cada elemento vienen dados por:

$$\boldsymbol{K}_{f,i} = \int_{\Omega_i} \left[\nabla \boldsymbol{N}_e^T \boldsymbol{K}_d \nabla \boldsymbol{N}_e + \boldsymbol{N}_e^T \boldsymbol{N}_e \right] d\Omega, \tag{1.17}$$

$$\boldsymbol{p}_{f,i} = \int_{\Omega_i} \boldsymbol{N}_e^T \rho \, d\Omega = \rho_e \int_{\Omega_i} \boldsymbol{N}_e^T \, d\Omega. \tag{1.18}$$

Este sistema de ecuaciones tiene unas características importantes. La matriz $\boldsymbol{K}_f$ es siempre simétrica y definida positiva, admitiendo factorización de Cholesky y siendo el *método del gradiente conjugado*, el algoritmo iterativo más efectivo para resolver el sistema de ecuaciones. Por lo tanto, el método del gradiente conjugado podrá usarse para problemas con un gran número de grados de libertad, mientras que la factorización de la matriz, puede ser más conveniente en problemas con menor número de grados de libertad, pues solamente debe factorizarse una vez ($\boldsymbol{K}_f$ no depende de la distribución de material).

Esta técnica de filtrado, tal y como se discute en [20], presenta algunas ventajas respecto a los filtros basados en convolución. En primer lugar, resuelven algunos problemas de estos últimos, como la distinción entre dominios diferentes, la distancia entre elementos en dominios convexos y el alto coste para implementar computación paralela en el proceso de filtrado. Por otra parte, permite una implementación muy sencilla en los códigos de optimización, debido a que puede emplearse el código de resolución del problema de elementos finitos. Por último, se puede emplear un filtro anisotrópico de manera sencilla, añadiendo solamente algunos parámetros extra de control, sin incrementar de manera significativa el coste computacional.

Actualización de las variables de diseño

Tras resolver las sensibilidades y filtrar las variables de diseño, se requiere un proceso matemático o computacional capaz de dar nuevos valores a las variables de diseño. Existen diferentes algoritmos para resolver esto, pudiendo ser categorizados en *métodos directos* y *métodos indirectos*, o procedimientos basados en *criterios de optimalidad* y *algoritmos de programación matemática.*, pero de entre todos los algoritmos, hay tres que suelen ser los más comunes: *criterios de optimalidad* (OCM)Bendsøe [6], *method of moving asympotes* (MMA) Svanberg [2] y *métodos*

secuenciales de programación lineal Dunning y Kim [21]. Los dos primeros serán los que se utilizarán en el presente trabajo.

Los criterios de optimalidad parten de convertir el problema de optimización restringida de la Ecuación (1.4), en uno no restringido, definiendo la siguiente función de Lagrange:

$$L(\boldsymbol{\rho}, \lambda) = c(\boldsymbol{\rho}) + \lambda(V(\boldsymbol{\rho}) - \alpha V_o), \tag{1.19}$$

aplicando las condiciones de primer orden de *Karush - Kuhn - Tucker*,

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{\rho}} = \dfrac{\partial c(\boldsymbol{\rho})}{\partial \boldsymbol{\rho}} + \lambda \dfrac{\partial V(\boldsymbol{\rho})}{\partial \boldsymbol{\rho}} = 0, \\ \\ \dfrac{\partial L}{\partial \lambda} = V(\boldsymbol{\rho}) - \alpha V_o = 0. \end{cases} \tag{1.20}$$

El OCM se compone de dos niveles reiterativos. En el nivel interno, las variables de diseño se actualizan buscando cumplir la primera restricción de (1.20), para un λ concreto. En el nivel externo, se actualiza el multiplicador de Lagrange λ con el fin de cumplir la restricción de volumen. Para ello se utiliza el algoritmo de la bisección, donde en cada iteración se ejecuta el nivel interno con un nuevo valor de λ. Para esto se define un «factor de escala» D_e en cada elemento:

$$D_e = -\frac{\dfrac{\partial c(\boldsymbol{\rho})}{\partial \rho_e}}{\lambda \dfrac{\partial V(\boldsymbol{\rho})}{\partial \rho_e}}, \tag{1.21}$$

cuando $D_e = 1$, se satisface la primera restricción de (1.20). La idea es que mediante este factor de escala, se establezcan los valores de cada variable de diseño mediante el siguiente esquema condicional:

$$\begin{cases} \text{máx}(0, \rho_e - m) & \text{Si} \quad \rho_e D_e^\eta \leq \text{máx}(0, \rho_e - m) \\ \text{mín}(1, \rho_e + m) & \text{Si} \quad \rho_e D_e^\eta \leq \text{mín}(1, \rho_e + m) \\ \rho_e D_e^\eta & \text{En caso contrario} \end{cases}, \tag{1.22}$$

donde m es un factor que limita el movimiento de las variables de diseño, y $\eta\,(= 1/2)$ es un factor de amortiguamiento.

1.1.3. Optimización topológica de geometrías infill

La generación de geometrías infill podría considerarse un caso particular de optimización topológica, donde se toman algunas pequeñas modificaciones al algoritmo

general. El punto clave del que se parte para este nuevo algoritmo es la modificación de la restricción volumétrica. Mientras que en la OT tradicional, se restringe la distribución de material a nivel global, para generar geometrías infill debe restringirse de manera local. Los principios básicos de este método fueron propuestos por Wu *et al.* [26], pero posteriormente se han hecho modificaciones, con diferentes objetivos, como las estructuras *Shell Infill*, Wu *et al.* [24].

El algoritmo en general, es muy similar al clásico algoritmo SIMP, únicamente cambiando la restricción volumétrica. Para ello, siendo ρ la densidad de un elemento y $\bar{\rho}$ la densidad promedio para un radio de influencia R, se busca que en todo momento, ese volumen promedio sea inferior a un valor preestablecido α, es decir:

$$\max_{\forall e}(\bar{\rho_e}) \leq \alpha. \tag{1.23}$$

Por supuesto, esta ecuación no puede aplicarse tal cual. Como se busca emplear un método basado en gradiente, se debe buscar una expresión diferenciable. Para ello se recurre a la norma p, la cual cuando $p \to \infty$, la función sea equivalente a la original:

$$\left(\frac{1}{n}\sum_{e}\bar{\rho_e}^{p}\right)^{\frac{1}{p}} \leq \alpha, \tag{1.24}$$

donde n es el número de elementos, y p normalmente se toma ($= 16$).

1.2. Posprocesado de resultados

El posproceso es una etapa que podría considerarse aparte del proceso de optimización. El objetivo es una vez obtenido los resultados de optimización, traducirlos a unos parámetros interpretables por el software de diseño o la máquina de fabricación. Esta tarea tiene cierta complejidad, por esto existen diversos enfoques de resolución.

Estos enfoques pueden dividirse en 3 categorías (ver Figura 1.8):

- *Iso-density contours methods*,
- Métodos de reconstrucción,
- Métodos de procesamiento de imágenes

Los *Iso-density contours methods*, se basan en pasar las densidades de cada elemento a una distribución de densidades en los nudos. A partir de aquí, se busca generar curvas de nivel de dicha solución, a través de curvas de Bézier. Esta representación con curvas de Bézier permite aplicar una optimización de forma a la

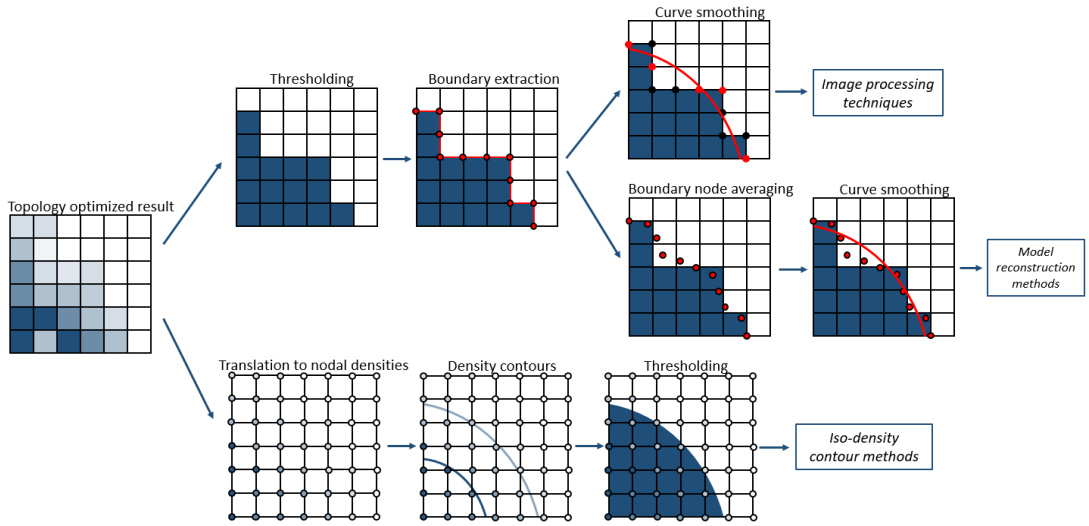


Figura 1.8: Enfoques adoptados para el posprocesado.

estructura para obtener una mayor definición de la misma. Una primera propuesta de este método fue la desarrollada por Maute y Ramm [8].

Por su parte, los métodos de reconstrucción, filtran las variables de diseño con el objetivo de obtener una representación binaria de la misma. Usando esta última, se extraen los límites de los contornos. A partir de estos puntos extraídos, se utiliza una aproximación de mínimos cuadrados sobre los nudos de un B-Spline. Esta alternativa se explica en mayor detalle en Tang y Chang [14]. Además, en Xia *et al.* [28], se propone una variante que busca mediante procesamiento de las variables de diseño, extraer una imagen esquelética de la estructura, en base a la cual identificar los elementos e iniciar una optimización de forma, para dar el tamaño necesario a dichos elementos.

Los métodos basados en procesamiento de imágenes, utilizan las técnicas tradicionales de vectorización de imágenes en mapa de bits, para interpretar los resultados de optimización topológica. Una de las primeras referencias de esta alternativa podría encontrarse en Papalambros y Chirehdast [5], que desarrolló un método de tres etapas:

1. Convertir variables de diseño a imagen binaria,
2. Extraer contornos,
3. Suavizar contornos usando B-splines y segmentos lineales.

Parte II

Metodología

Capítulo 2

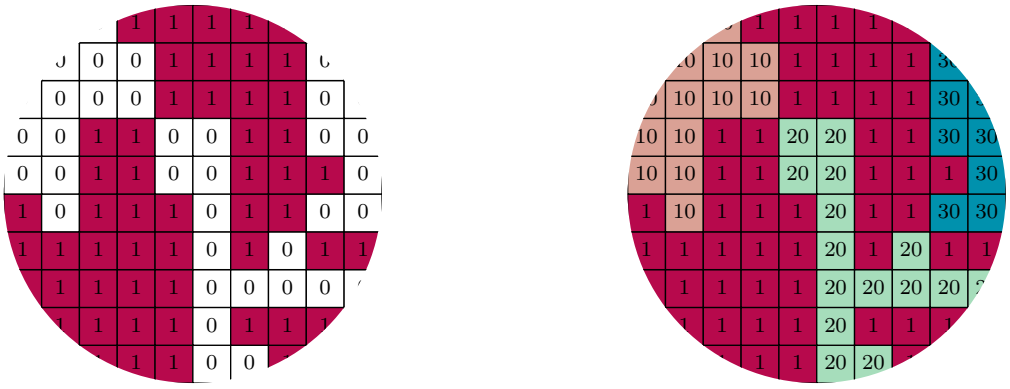
Distinción

Contenido

2.1. Introducción

El procedimiento nombrado como distinción, parte de una imagen binaria, donde los agujeros tienen valor 0 y las regiones sólidas valor 1, Figura 2.1(a). La idea es cambiar el valor asociado a los elementos de cada agujero, por uno que permita distinguirlos de manera sencilla, por ejemplo, cambiando los valores del primer agujero por 10, el segundo por 20 y el enésimo por $10\,n$, Figura 2.1(b).

A priori, este cambio podría resultar innecesario, pues como se verá en el Capítulo 3, el algoritmo usado para extraer los contornos, no exige distinguir cada uno de los agujeros previamente. De hecho, para geometrías relativamente sencillas, funciona de manera adecuada. En el Capítulo 3, se justifica el uso de este algoritmo.



(a) (b)

Figura 2.1: Ejemplo de imagen sin distinguir en (a) y distinguiendo cada agujero en (b).

2.2. Filtrado

Para que el algoritmo funcione adecuadamente, primero debe aplicarse un filtrado de las variables de diseño. De este modo se eliminan los valores intermedios y se genera una imagen completamente binaria. Para ello, se aplica una función que revisará cada elemento y aplicará la siguiente operación:

$$\tilde{\boldsymbol{\rho}} = \begin{cases} 0 & \text{si} \quad \rho_i < 0{,}5 \\ 1 & \text{si} \quad \rho_i \geq 0{,}5 \end{cases}, \tag{2.1}$$

Con esto se eliminarán los elementos de valor intermedio, consiguiendo una "imagen" de valores binarios.

2.3. Conectividad

Para comenzar el algoritmo primero hay que preguntarse qué será interpretado como agujero, ya que tener esto claro, es fundamental para crear un procedimiento de interpretación de los resultados. En general, agujero será, todo conjunto de elementos con densidad 0, que tengan conectividad 4 entre si. La conectividad puede definirse como la manera en la que un píxel se relaciona con sus vecinos. Para el alcance de este algoritmo solo será necesario conocer la conectividad 4 y 8, que son las permiten definir qué celdas usar y cuándo.

Dos píxeles tienen conectividad 4, cuando tienen una cara en común. Si únicamente comparten una esquina, entonces tienen conectividad 8 (ver Figura 2.2). Por lo

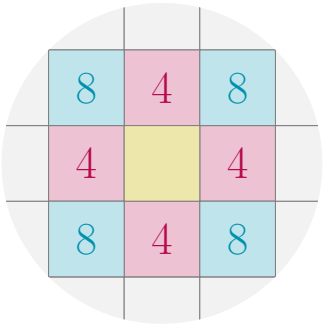


Figura 2.2: Ejemplo de tipos de conectividad entre elementos.

tanto, para una situación como la representada en la Figura 2.3, los elementos $\{(1,1)\,;\,(1,2)\,;\,(1,2)\,;\,(2,2)\}$ forman parte de un mismo agujero, mientras que los elementos $(3,1)$ y $(3,3)$ son parte de un agujero diferente.

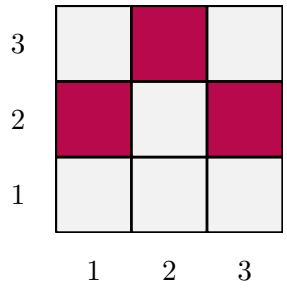


Figura 2.3: Ejemplo de diferenciación de agujeros.

2.4. Algoritmo de diferenciación de agujeros

El objetivo de este algoritmo es obtener píxeles de partida para el proceso posterior, por tanto, lo primero es encontrar elementos válidos para este propósito. Lo que se hace es buscar una posición del conjunto solución $\boldsymbol{\rho}$ con valor valor 0, esta se guarda, ya que forma parte del contorno, y se inicia el Algoritmo 1. Este algoritmo cambiará los valores de los elementos de ese agujero a un valor $\{10, 20, 30, ...\}$, de modo que cuando se reinicie la búsqueda de otro píxel válido, bastará con buscar otra posición con valor 0, ya que esta será automáticamente un elemento situado en otro contorno diferente del anterior. Como se han cambiado los valores referidos a los elementos del primer agujero, este no se detecta como tal en búsquedas futuras, y no se requieren condiciones adicionales para descartar su contorno en la búsqueda de nuevos puntos de partida.

ola

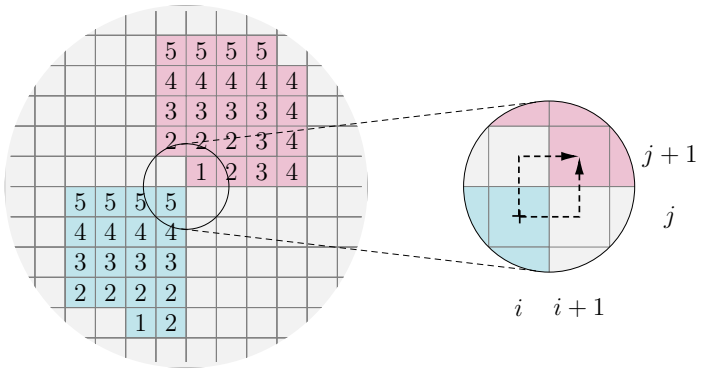


Figura 2.4: Descripción del funcionamiento del Algoritmo 1.

El Algoritmo 1, da valores a los elementos que forman el agujero y se describe gráficamente en la Figura 2.4, donde en cada elemento se indica el número de iteración del algoritmo. Se observa que, el proceso comienza tomando el primer elemento del agujero y comprobando cuales son los elementos colindantes que tengan conectividad 4 y 8. De entre esos elementos, se guardan en una lista los que tengan densidad 0, a estos se les cambia el valor a $\{10, 20, 30, ...\}$, según corresponda, y vuelve a iniciarse el Algoritmo 1 con esta última lista, hasta que esta se quede completamente vacía.

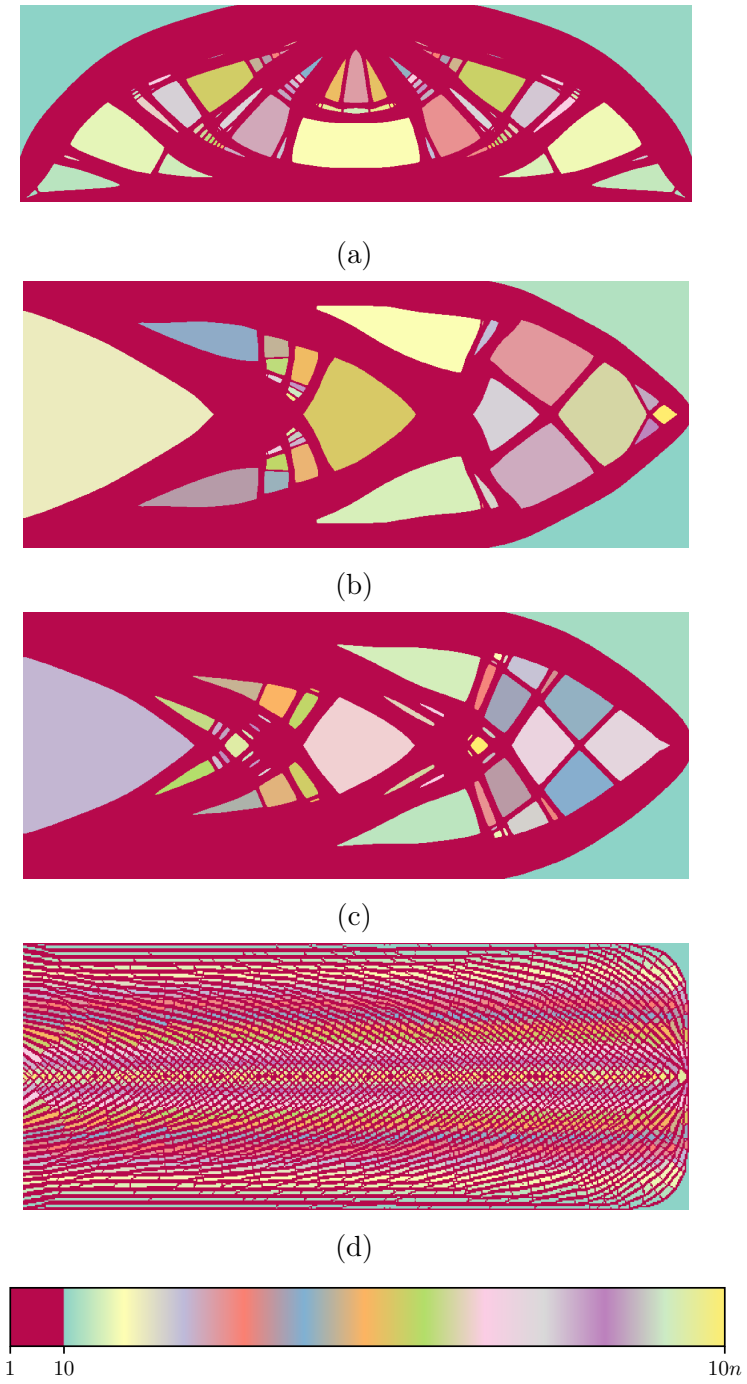
Debe prestarse atención a casos como el que se presenta en el detalle de la Figura 2.4, donde se muestra que para ser seleccionado como parte del mismo agujero, debe ser posible llegar al elemento a través de otro encadenando conectividades tipo 4. Por tanto, en el dibujo, como para ir del elemento (i, j) al elemento $(i + 1, j + 1)$, hay que pasar por los elementos $(i, j + 1)$ o $(i + 1, j)$, y estos tienen densidad 1; el elemento $(i + 1, j + 1)$ forma parte de otro agujero diferente.

Algoritmo 1 Función recursiva para identificar el agujero.

```
1: procedure AGUJERO(ρ, p[], Nel_x, Nel_y, id)
2:     pp[]                          ▷ lista vacía para acumular los nuevos puntos
3:     for l ∈ p do
4:         i ← l(0)
5:         j ← l(1)
6:         i_1 ← máx(i − 1, 0)
7:         j_1 ← máx(j − 1, 0)
8:         i_2 ← mín(i + 1, Nel_x)
9:         j_2 ← mín(j + 1, Nel_y)
10:        AGREGAR(ρ, i, j_1, i_1, j_1, i_2, j_1, pp, h)      ▷ Los 3 superiores
11:        AGREGAR(ρ, i, j_2, i_1, j_2, i_2, j_2, pp, h)      ▷ Los 3 inferiores
12:        AGREGAR(ρ, i_1, j, i_1, j_1, i_1, j_2, pp, h)      ▷ Los 3 del lado izquierdo
13:        AGREGAR(ρ, i_2, j, i_2, j_1, i_2, j_2, pp, h)      ▷ Los 3 del lado derecho
14:    end for
15:    if length(pp) > 0 then
16:        AGUJERO(ρ, p[], Nel_x, Nel_y, id)
17:    end if
18:    return ρ
19: end procedure
20:
21: procedure AGREGAR(ρ, i, j, i_1, j_1, i_2, j_2, pp[], h)
22:    if ρ(j, i) = 0 then:          ▷ Se comienza comprobando el elemento con
    conectividad 4.
23:        append (i, j) → pp
24:        ρ(j, i) = h
25:        if ρ(j_1, i_1) = 0 then:
26:            append (j_1, i_1) → pp
27:            ρ(j_1, i_1) = h
28:        end if
29:        if ρ(j_2, i_2) = 0 then:
30:            append (j_2, i_2) → pp
31:            ρ(j_2, i_2) = h
32:        end if
33:    end if
34: end procedure
```

2.5. Ejemplos

En la Figura 2.5 se muestran algunos ejemplos en los que se ha aplicado el algoritmo explicado previamente. Si se observa en primer lugar la leyenda de color (Figura 2.5(e)), y a continuación cualquiera de los ejemplos; los colores guardan simetría respecto al eje neutro de la pieza, siempre que esta tenga un eje de simetría horizontal. El motivo de esto es que para que se guarden los puntos de partida de manera que la pieza interpretada sea simétrica, en vez de hacer búsquedas en orden ascendente en el eje de ordenadas, se hacen de manera alternativa desde los extremos hasta el eje de simetría.



(a)

(b)

(c)

(d)

(e)

Figura 2.5: Ejemplos de casos complejos en los que se ha aplicado el proceso distinción.

Capítulo 3

Extracción de contornos

Contenido

3.1. Introducción

El algoritmo que se ha presentado en el Capítulo 2 da como resultado una lista con las posiciones de los elementos que inician el algoritmo recursivo que cambia el valor de sus píxeles vecinos. Esto es el punto de partida de la siguiente etapa del proceso: La detección de los contornos. Este trata de aplicar una serie de axiomas, que genere un conjunto de listas de coordenadas. Cada una de las listas se compondrá de la posición de los puntos frontera de cada agujero ordenados de manera que,

al representar estos puntos unidos mediante líneas, formen la imagen vectorial del agujero de partida.

En este capítulo se presentará el algoritmo usado para la generación de estas listas de coordenadas, el cual se inspira en el algoritmo de Pavlidis [1].

3.2. Algoritmo de Theo Pavlidis

La mayor dificultad de trazar los contornos de una imagen en mapa de bits, radica en saber qué información escoger. Esto se debe a que la interpretación que las personas hacen de la imagen, es un conocimiento tácito, basado en observar el conjunto de los elementos de la imagen. Sin embargo, al realizar el reconocimiento de los contornos de manera computacional, debe usarse un conocimiento explícito y limitado, ya que el número de axiomas necesarios para tomar las decisiones de interpretación, crecen de manera exponencial con el conjunto de datos escogido.

El algoritmo de Theo Pavlidis, utiliza un conjunto de datos muy limitado para tomar decisiones. Estos datos son los valores de los 3 elementos que hay frente al elemento actual, en una dirección y sentido a determinar. Así pues, a partir de estos 3 valores, se toman 2 decisiones diferentes según lo mostrado en la Figura 3.1, donde indica el pixel superior izquierdo, el pixel superior derecho y el píxel superior central. Por último, indica el píxel actual y la dirección en la que se escogen los 3 píxeles, dependiendo de la inclinación del símbolo: ,, y .

(a) (b)

Figura 3.1: Posibles casos del algoritmo de Pavlidis.

Este algoritmo se puede definir a derechas y a izquierdas, según el sentido de giro que quiera darse. En la Figura 3.1, se indica la versión a izquierdas, la versión a derechas sería la imagen simétrica a la misma.

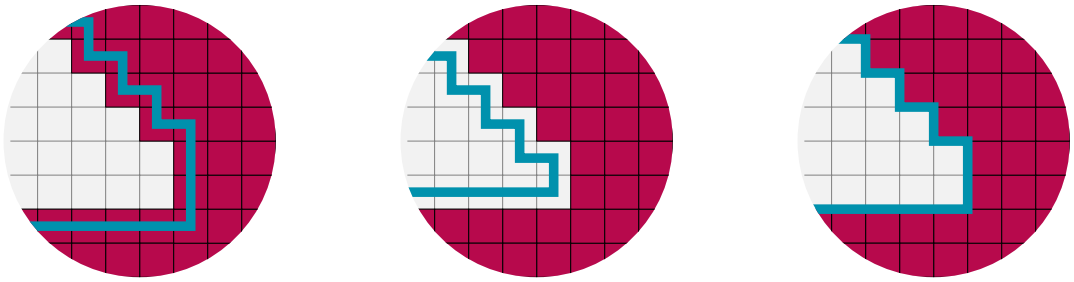
Se comienza con un píxel dentro del agujero o fuera, pero que esté situado en

la frontera entre dos regiones. Se miran los tres elementos superiores, inferiores o laterales, dependiendo de la dirección y sentido de búsqueda. Si se encuentra un caso como el de Figura 3.1(a), en la siguiente iteración se realizará la operación en el píxel superior izquierdo. En caso de que se encuentre el caso Figura 3.1(b), se escogerá el píxel superior central. Si no es ninguno de los anteriores, se girará sobre la misma posición hasta encontrar uno de estos dos casos. Por último, si se gira más de 3 veces sobre el mismo píxel o se llega a la posición de partida, se considera que el algoritmo ha terminado y se inicia una nueva búsqueda de otro píxel de partida.

3.3. Algoritmo de identificación de contornos

El algoritmo propuesto por Pavlidis [1], es un punto de partida interesante para abordar el problema, aunque este planteamiento se alterará levemente, para adaptarse a los requerimientos este proyecto.

En primer lugar, se debe definir qué información se debe atribuir a cada una de las listas que delimitan un contorno. Esto puede hacerse de múltiples maneras, como por ejemplo, guardando el centro de cada elemento por el que se pasa en cada iteración (Figura 3.2(a) y 3.2(b)), o bien, guardando las coordenadas límite entre ambos dominios (Figura 3.2(c)).



(a) (b) (c)

Figura 3.2: Ejemplos de interpretación del contorno.

Para este proyecto se opta por interpretar el contorno de la manera que aparece en la Figura 3.2(c), pero con una pequeña modificación. Se observa que la sucesión de dos coordenadas se da en ángulos $\{0°, 90°\}$. Teniendo en cuenta que el método de suavizado que se comentará más adelante, estará basado en promediados, cuando se esté guardando información de un borde inclinado, se tendrá un escalonamiento que podría generar líneas muy irregulares. Este efecto se observaría sobretodo en líneas inclinadas a 45°.

Para mejorar un poco este efecto, se puede hacer que el algoritmo guarde puntos sucedidos a una inclinación de 45° (ver Figura 3.3). Por supuesto, esto no elimina totalmente la irregularidad de líneas inclinadas, pero si que resta fuerza o gravedad al escalonamiento que se produce en estas.

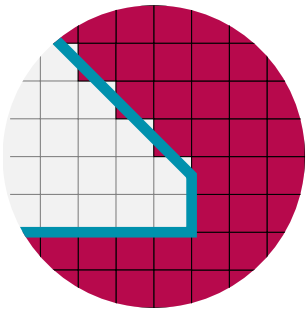


Figura 3.3: Contorno con inclinaciones de 45°.

Todo lo comentado se hará utilizando un operador similar al de Pavlidis [1], que se muestra en la Figura 3.4, donde los píxeles escogidos para tomar las decisiones de trazado son, el píxel frente al actual y los dos colindantes a estos. Estos dos últimos se situarán al lado derecho (como en la Figura 3.4), si se pretende trazar el agujero en sentido antihorario, y a la izquierda si el trazado es horario.

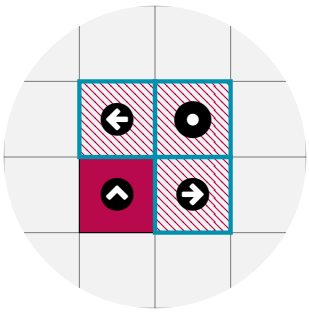


Figura 3.4: Operador del algoritmo de trazado de contornos.

3.3.1. Generalización de direcciones

El método que se utilizará, debe tomar decisiones en base a la información proporcionada por el operador, en una dirección y sentidos concretos (Figura 3.4). Para simplificar la programación de estos tres píxeles, se ha definido la función RELOJ, que se encarga de generalizar las 4 direcciónes de "observación".

En la Figura 3.5 se muestra gráficamente cómo funciona el Algoritmo 2. Se trata de que, dado un valor $0, 1, 2, 3$, se devuelva la posición de los píxeles que forman el

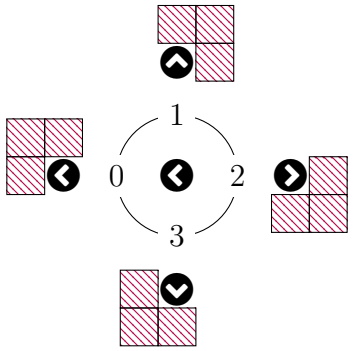


Figura 3.5: Ejemplo gráfico de la función RELOJ.

operador de esa dirección. Para cambiar de orientación, solamente se tendrá que hacer $reloj \pm 1$, y se realizará un giro de $\pm 90°$ relativo a la dirección anterior (en caso horario $\mp 90°$).

Por otra parte, esta generalización de las direcciones, también afectará a la enumeración de los puntos clave de un elemento, los cuales vendrán dados por el criterio de la Figura 3.6. Estos puntos clave son necesarios para guardar las posiciones adecuadas de los bordes entre dominios, de acuerdo a lo expuesto en la Figura 3.3.

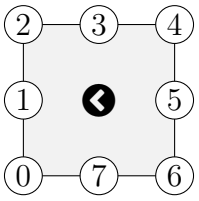


Figura 3.6: Puntos clave del elemento.

Algoritmo 2 Función recursiva para adimensionalizar el problema.

```
1: procedure RELOJ(reloj, i, j, Nel_x, Nel_y)
2:     i_1 = máx(i − 1, 0)
3:     i_2 = mín(i + 1, Nel_x − 1)
4:     j_1 = máx(j − 1, 0)
5:     j_2 = mín(j + 1, Nel_y − 1)
6:     if reloj == 0 then
7:         ik = [i_1, i_1, i_1] ▷ Listas para acumular las coordenadas de los tres píxeles
8:         jk = [j_1, j, j_2]
9:     else if reloj == 1 then
10:        ik = [i_1, i, i_2]
11:        jk = [j_2, j_2, j_2]
12:    else if reloj == 2 then
13:        ik = [i_2, i_2, i_2]
14:        jk = [j_2, j, j_1]
15:    else if reloj == 3 then
16:        ik = [i_2, i, i_1]
17:        jk = [j_1, j_1, j_1]
18:    else
19:        reloj = reloj  mód 4
20:        ik, jk, reloj = RELOJ(reloj, i, j, dim)
21:    end if
22:    return ik, jk, reloj
23: end procedure
```

3.3.2. Determinación de iteraciones

Tras generalizar las direcciones a un problema unidimensional, simplemente debe tomarse decisiones en base a los valores de 3 elementos. Como los valores que estos pueden tomar son binarios, se tiene que en total pueden encontrarse $2^3 = 8$ casos, que se enumeran en la Tabla 3.1. La tarea entonces será tomar decisiones en base a qué caso se encuentre en cada iteración.

El algoritmo se ha programado de manera que rodee internamente al dominio que se va a trazar. Este comienza por un elemento límite y se direcciona según $reloj = 2$.

Por otra parte, aunque lo que se expondrá pueda parecer únicamente válido para un sentido de giro antihorario, la simetría del problema, admite el uso del mismo algoritmo, siempre y cuando se definan para el sentido correcto en lo expuesto en las Figuras 3.4, 3.5 y 3.6.

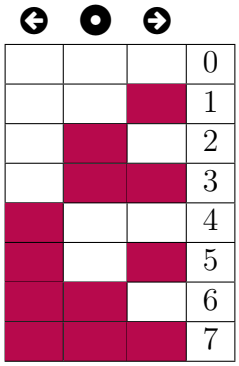


Tabla 3.1: Enumeración de los casos posibles

Caso 0

Es un caso en el que todos los elementos tienen valor cero. Se considera un caso irrelevante o inalcanzable, ya que solo podría darse en dos situaciones:

1. Debido a un error en el sentido de giro, el operador apunta hacia el interior del agujero.
2. Se ha avanzado a un píxel interior. Este caso debería ser inalcanzable, ya que la sucesión del resto de situaciones, se resuelve de manera que no se llegue a un caso como este.

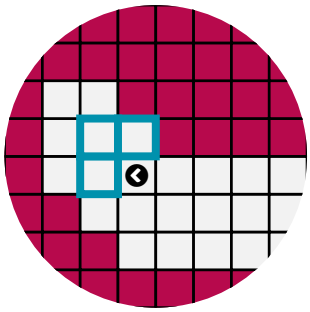


Figura 3.7: Ejemplo de caso 0.

Por lo tanto, la decisión más conveniente es hacer $reloj + 1$, intentando redirigir el operador hacia el borde, en busca de un conjunto de valores diferente.

Caso 1

Esta situación es de las más recurrentes, pues mediante la misma, se explicita el avance diagonal. Este avance permite, además de generar trazados inclinados a 45°,

hacer que el resto de inclinaciones intermedias $(0°, 45°, 90°)$, tengan un escalonamiento menos abrupto; facilitando tratamientos posteriores.

En este caso, se avanzará hacia el píxel ➔, y se guardará la posición {3} (ver Figura 3.8).

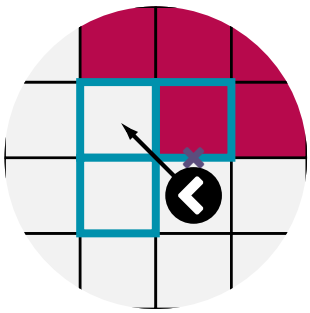


Figura 3.8: Ejemplo de caso 1.

Caso 2

Es una situación algo excepcional, que usualmente se da cuando, tras un avance diagonal que esté sucedido de otro rectilíneo, aparece un píxel saliente (ver Figura 3.9). En esta situación se hace $reloj + 1$, con intención de posicionarse en el elemento superior en la siguiente iteración.

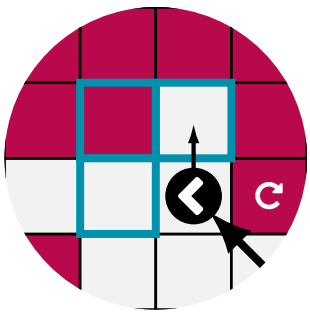


Figura 3.9: Ejemplo de caso 2.

Caso 3

Es otro caso de los más recurrentes, mediante la sucesión de casos 3, pueden generarse los trazados rectilíneos $\{0°, 90°\}$. Lo que se hace es agregar en la lista de coordenadas el punto {3} y pasar al elemento ➔ (ver Figura 3.10).

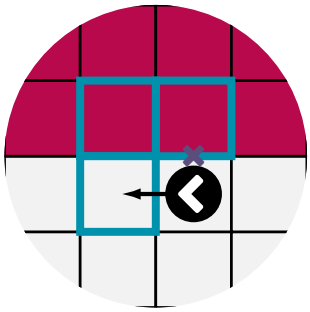


Figura 3.10: Ejemplo de caso 3.

Caso 4

Un ejemplo posible de esta situación se encuentra en la Figura 3.11, la cual se resuelve de manera idéntica al Caso 2.

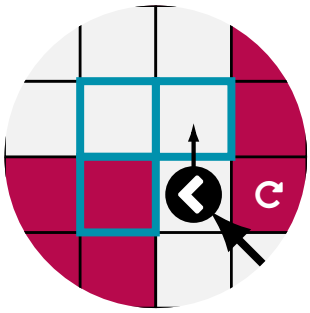


Figura 3.11: Ejemplo de caso 4.

Caso 5

En este caso, se encuentra que únicamente el píxel ⊙, tiene valor cero. Esto implica que la conectividad entre el conjunto de elementos al que pertenece el elemento actual y el conjunto del que forma parte el elemento ⊙, no es conectividad 4. Es decir, no pertenecen al mismo agujero.

Debido a lo anterior, se interpretará este caso, de la misma manera que se haría si todos los elementos del operador tuvieran valor 1 (ver Figura 3.12).

Caso 6

Esta situación se muestra en la Figura 3.11, la cual se resuelve de manera idéntica al Caso 2 y 4.

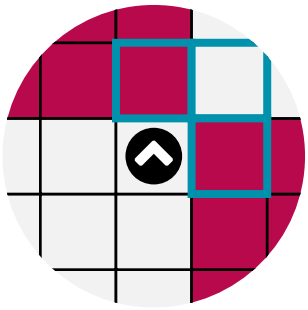


Figura 3.12: Ejemplo de caso 5.

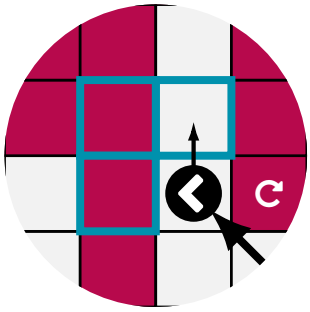


Figura 3.13: Ejemplo de caso 6.

Caso 7

Esta situación es de las más importantes en el algoritmo, debido a que permite trazar los huecos más pequeños, como los píxeles aislados.

El caso 7 general, se muestra en la Figura 3.14(a). En esta situación, normalmente se hace $reloj - 1$ y se añade a la lista de coordenadas el punto número 3, de acuerdo con la Figura 3.6. Pero, si se cumple la condición de que los dos casos anteriores son casos 7 o se acaba de empezar el trazado, se añade otra coordenada de acuerdo a la siguiente media de puntos,

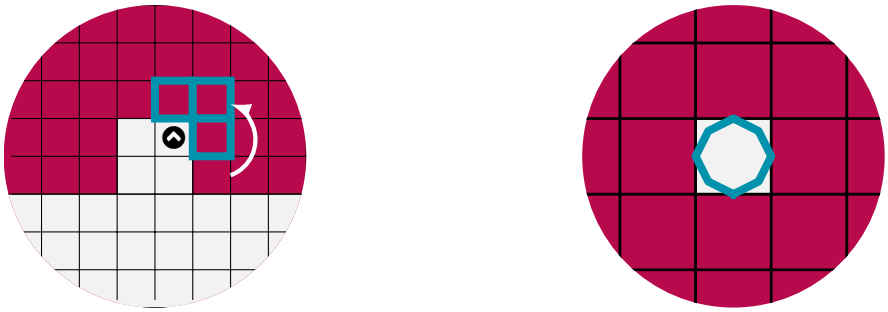
$$P = \frac{P_3 + P_4 + P_5}{3}, \tag{3.1}$$

donde P hace referencia a la coordenada x o y, y el subíndice $\{3, 4, 5\}$ a la enumeración de los puntos del elemento (ver Figura 3.6).

Esto permite interpretar de manera automática y sin tratamientos posteriores, los píxeles aislados como el de la Figura 3.14(b). Aquí, mediante la sucesión de casos 7, se tiende a redondear las esquinas, creando pequeños contornos ya suavizados.

En contornos de mayor tamaño, también podrían aparecer redondeos en las esquinas, pero esto no resulta problemático, ya que posteriormente se aplicarán

tratamientos sobre esas líneas.



(a) Caso 7 general (b) Píxel aislado

Figura 3.14: Ejemplos de caso 7

3.3.3. Elección del sentido de giro

Como ya se ha dicho, el algoritmo de trazado bajo los mismos criterios de funcionamiento, cambiando los parámetros iniciales vistos en las Figuras 3.4, 3.5 y 3.6; se puede trazar en sentido horario o antihorario.

Definir ambos sentidos de giro es necesario para garantizar, en la medida de lo posible, la simetría en piezas simétricas. Como normalmente se elige el sentido antihorario para los contornos situados en la mitad inferior, para los que se sitúen en la mitad superior deberá elegirse el sentido horario. De esta manera, se garantiza que contornos simétricos proporcionados por el proceso de optimización, mantengan la simetría tras trazar el contorno.

Por supuesto, para garantizar la simetría no solo es importante el sentido de giro, también es necesario que el punto de partida sea idéntico. Por este motivo, la búsqueda de estos puntos de partida se hace, desde los extremos inferior y superior, hasta el centro de la pieza, como se explicó en el Capítulo 2.

3.3.4. Tratamiento de los agujeros externos

Externos son aquellos agujeros colindantes con algún límite del dominio. Estos al ser contornos abiertos, presentan algunos requerimientos específicos para que sean trazados adecuadamente.

Al plantear la búsqueda de elementos de partida, haciendo un barrido fila a fila, podrían darse situaciones como las de la Figura 3.15 marca (a), donde el píxel inicial estaría en un punto intermedio del contorno, y al aplicar el algoritmo de trazado

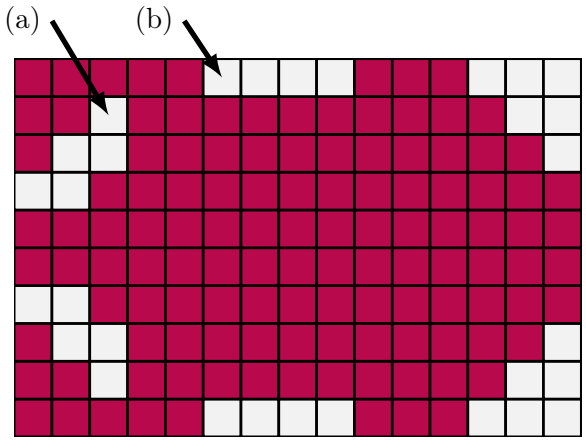


Figura 3.15: Ejemplos típicos de bordes externos.

no rodearía completamente al agujero. Además en la marca (b), si se comienza con el operador ajustado para un sentido horario, este apuntará hacia el exterior del dominio, no consiguiendo avanzar.

Por tanto, debido a la problemática que aguarda tras estos agujeros, es conveniente hacer una búsqueda anticipada de estos, de modo que se fuerce al algoritmo a encontrar un punto de partida beneficioso que evite los problemas anteriormente mencionados. Con esto en mente, a la hora de buscar los píxeles de partida se deben usar también algunas restricciones que garanticen un trazado correcto:

- Si el agujero es colindante con la pared inferior del dominio, debe trazarse en sentido horario.
- Si el agujero es colindante con la pared superior del dominio, se usa el sentido antihorario.
- El punto de partida debe estar situado en un límite del dominio.

3.3.5. Criterio de parada

Para el algoritmo trazador se establecen dos algoritmos de parada, uno para los agujeros internos, y otro para los externos.

Los agujeros internos tienen contornos cerrados que comienzan y acaban en el mismo punto. Por lo tanto, cuando se llegue al píxel inicial y el valor de *reloj* sea igual al inicial, se habrá llegado al final del trazo. Para asegurar que el contorno sea cerrado, si el punto inicial y final no son iguales, se agregara el punto de partida al final del vector.

En los agujeros externos, debido a la naturaleza del operador y de cómo se han programado los límites, cuando se topa con el final de contorno, empieza a reiterar sobre el mismo punto con el caso 3 de manera ininterrumpida. Solamente con detectar esta situación se puede dar por terminado el trazado.

3.3.6. Límites del dominio

La última parte del trazado de la pieza, corresponde a generar las líneas que conforman el dominio material de la misma. Estas son necesarias para cerrar el trazado en el programa CAD, permitiendo generar la imagen tridimensional de la estructura (ver Figura 3.16).

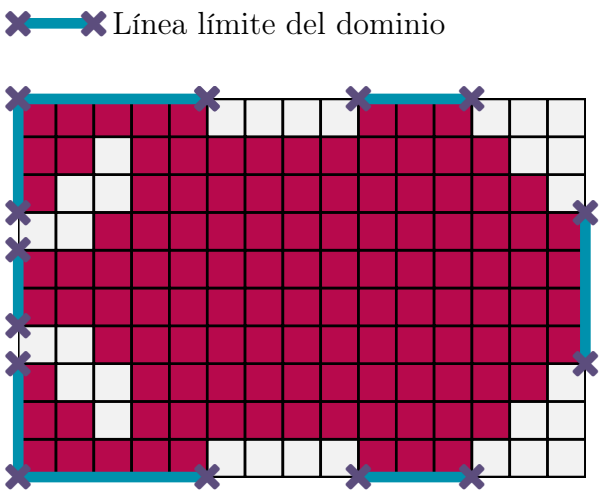


Figura 3.16: Representación de los límites del dominio.

Para guardar estas líneas, compuestas por la coordenada inicial y final, se crean 4 vectores con las coordenadas de los elementos de cada uno de los 4 contornos. A cada contorno se le define una dirección de avance de acuerdo a los sentidos del *reloj* (Figura 3.5):

- Pared inferior $reloj = 2$,
- Pared izquierda $reloj = 1$,
- Pared superior $reloj = 0$,
- Pared derecha $reloj = 3$.

Tras esto, para generar cada uno de las líneas, solamente hay que comenzar una búsqueda en los 4 vectores de coordenadas siguiendo las condiciones:

- Si se encuentra un píxel con valor 1, "se abre" un vector y se agrega el punto 4.
- "Abierto" un vector, este debe "cerrarse" bajo dos condiciones:
    1. Se encuentra un píxel de valor 0, donde se cierra el vector con el punto 4.
    2. Se llega a la última coordenada de la pared, donde se cierra con el punto 2.

3.4. Sobre la necesidad de distinguir los agujeros

Si se analiza detenidamente el algoritmo que se ha presentado, podría surgir la duda de por qué es necesario el algoritmo de distinción de agujeros presentado en el Capítulo 2, ya que el algoritmo de trazado por sí mismo es capaz de evaluar la conectividad, distinguiendo entre los diferentes agujeros. Es cierto que no es absolutamente necesario incorporar este paso intermedio para hacer funcionar el conjunto del algoritmo, pero al hacerlo, surge un problema: La búsqueda del píxel de partida.

Para esto se puede pensar en realizar búsquedas de píxeles que sean potenciales puntos de partida (aquellos elementos de valor cero que estén en el borde). Para ello se hacen consultas elemento a elemento en cada fila y cuando un píxel sea válido, se traza su contorno. Para no reiterar sobre un mismo contorno en el futuro, se guardan las posiciones por donde se "ha pasado" en una lista. Con esto, cuando se reinicie la búsqueda de elementos de partida, se comprueba que si una nueva posición potencialmente válida como punto de partida, se encuentra o no entre los elementos ya "visitados", y si no está en la lista, se inicia en esta posición un nuevo trazado.

Esta parece a priori, una manera sencilla de encontrar los inicios del trazado, pero la realidad es que su coste operativo es bastante alto. Al añadir los elementos visitados a un vector, y consultar si la posición actual existe en el mismo reiteradas veces para un mismo agujero; cuando el conjunto de datos es muy grande, la tarea de búsqueda empieza ser muy costosa, y se requiere un enfoque alternativo. Esto se hace especialmente importante en los resultados de *Infill*, donde dependiendo de la complejidad, se puede tener una densidad de agujeros altísima, que para superficies pequeñas pueden ser de miles de ellos, haciendo imposible utilizar este método de búsqueda de manera eficiente.

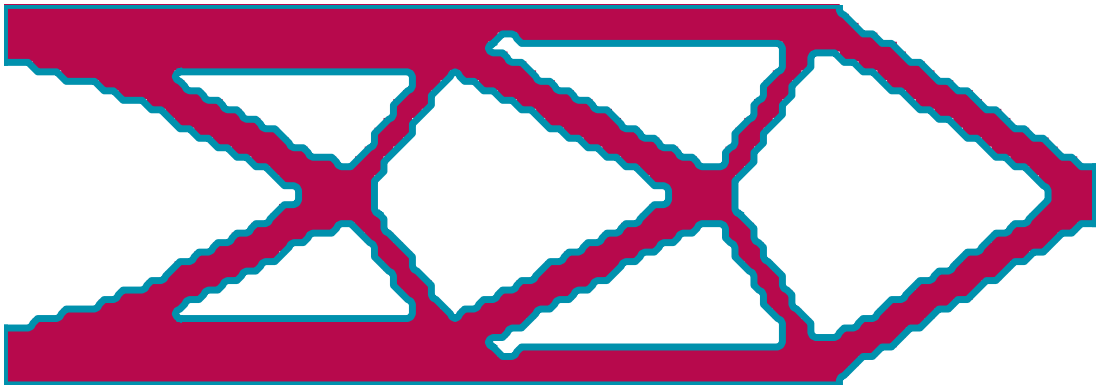
Por este motivo, el algoritmo de distinción de agujeros explicado en el Capítulo 2 resulta mucho más efectivo. La elección del píxel solo requiere que se cumpla la

condición de ser igual a 0, ya que al encontrarlo, los elementos de ese agujero cambiarán de valor automáticamente. Cuando se vuelva a iniciar la búsqueda, como los elementos del agujero anterior ya no tienen valor 0, el algoritmo los ignorará, llegando a otro agujero diferente. Estos píxeles serán automáticamente válidos, pues el primer píxel con valor 0 de un agujero, siempre estará en el borde del mismo.

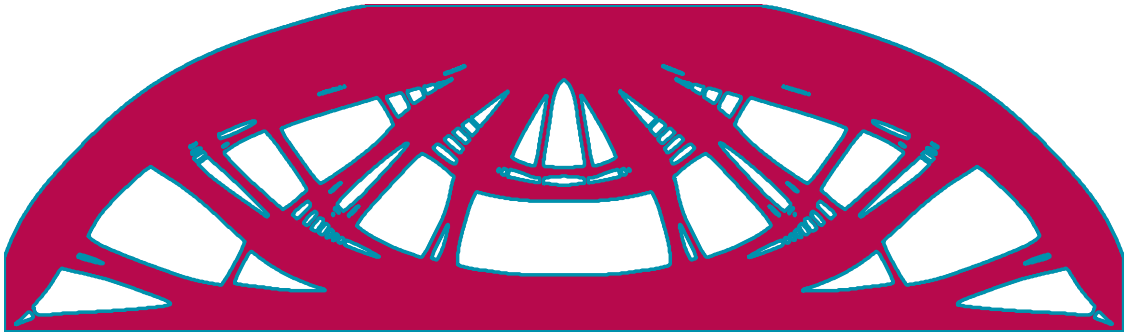
3.5. Ejemplos

En las Figuras 3.17 y 3.18, se muestran algunos ejemplos prácticos resueltos por el algoritmo. Se puede apreciar como este método proporciona imágenes completamente simétricas, siempre que los datos de entrada lo sean. El único problema es que las imágenes presentan líneas muy irregulares.

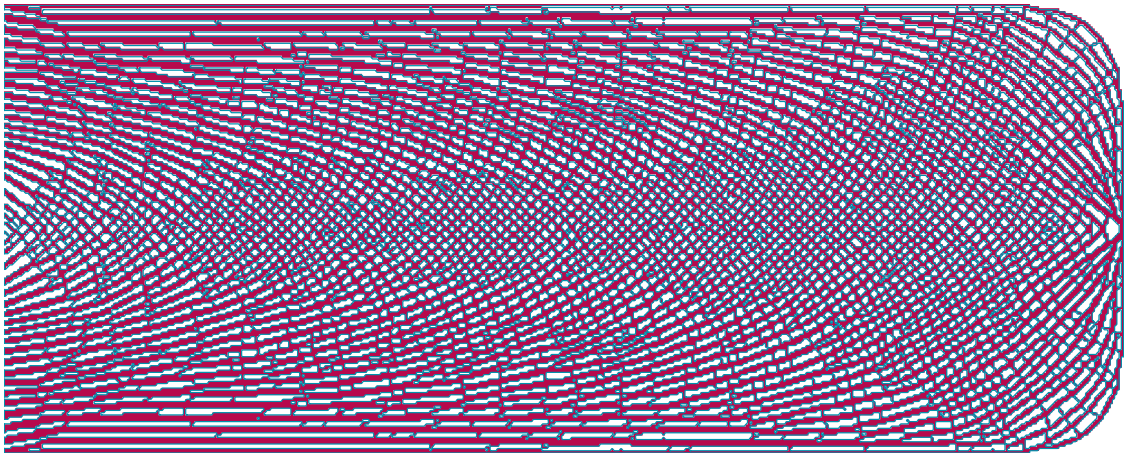
En este capítulo se ha presentado la segunda etapa de la interpretación de las soluciones de optimización topológica. Una vez se ha conseguido definir los contornos (aunque sea de manera irregular), se aplicarán tratamientos a los mismos suavizarlos y simplificarlos. Obteniendo una imagen más próxima a la interpretación intuitiva que se hace de la misma.



(a)

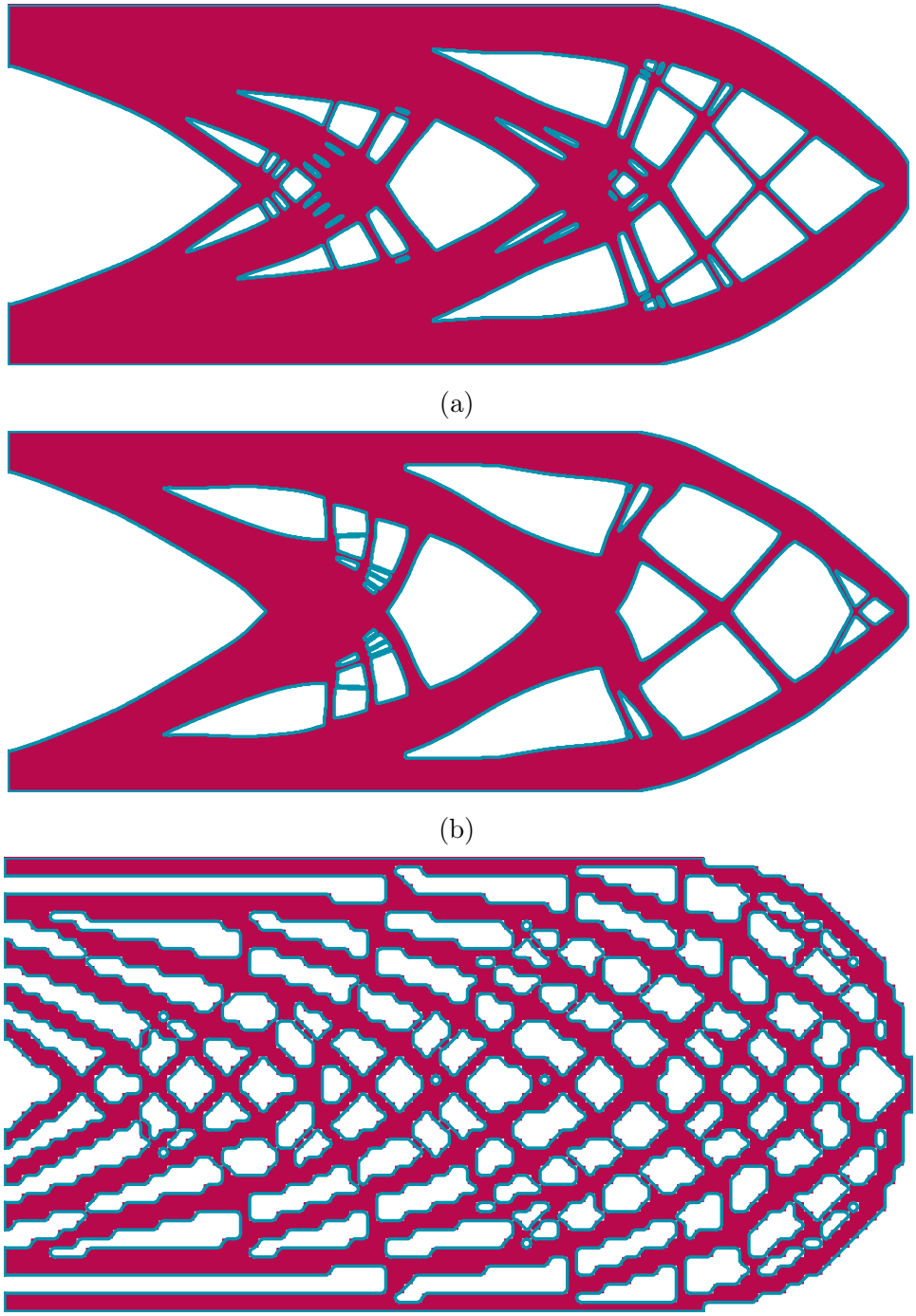


(b)



(c)

Figura 3.17: Muestra de casos reales resueltos por el algoritmo.



(a)

(b)

(c)

Figura 3.18: Muestra de casos reales resueltos por el algoritmo.

Capítulo 4

Tratamiento de línea

Contenido

4.1. Introducción

En los anteriores capítulos, se ha propuesto un método para convertir una imagen en forma de matriz de valores binarios a otra imagen vectorial dada por las coordenadas de los puntos adscritos al límite entre cada dominio. El problema de esta imagen es que las líneas pueden presentar fuertes irregularidades, además de tener información excesiva. Por este motivo, es necesario incorporar métodos que por un lado, suavicen las líneas y por otro las simplifiquen.

4.2. Suavizado

El método de suavizado se basa en un promediado móvil de los puntos. La idea es que si se define la posición del punto actual, como la media de los n valores

anteriores y posteriores, se podrá eliminar las inestabilidades de la línea y redondear las esquinas.

Algoritmo 3 Algoritmo para suavizar los contornos

procedure SUAVIZADO(x, y, $n[r]$)
  $x_1[length(x)]$
  $y_1[length(x)]$
  for $i = 0$ to $length(r)$ **do**
    for $j = 0 \rightarrow length(x)$ **do**
      $\sum_x \leftarrow 0$
      $\sum_y \leftarrow 0$
      $m \leftarrow 0$
      for $k = j - n(r) \rightarrow j + n(r)$ **do**
        $m \leftarrow m + 1$
        if $0 \leq k < length(x)$ **then**
          $\sum_x \leftarrow \sum_x + x(k)$
          $\sum_y \leftarrow \sum_y + y(k)$
        else if $k \geq length(x)$ **then**
          $\sum_x \leftarrow \sum_x + x(k - length(x))$
          $\sum_y \leftarrow \sum_y + y(k - length(x))$
        else
          $\sum_x \leftarrow \sum_x + x(k + length(x))$
          $\sum_y \leftarrow \sum_y + y(k + length(x))$
        end if
        $x_1(j) \leftarrow \frac{\sum_x}{m}$
        $y_1(j) \leftarrow \frac{\sum_y}{m}$
      end for
    end for
  end for
  return x_1, y_1
end procedure

Para hacer esto de manera computacional se ha usado el Algoritmo 3. Este se compone de dos parámetros de ajuste:

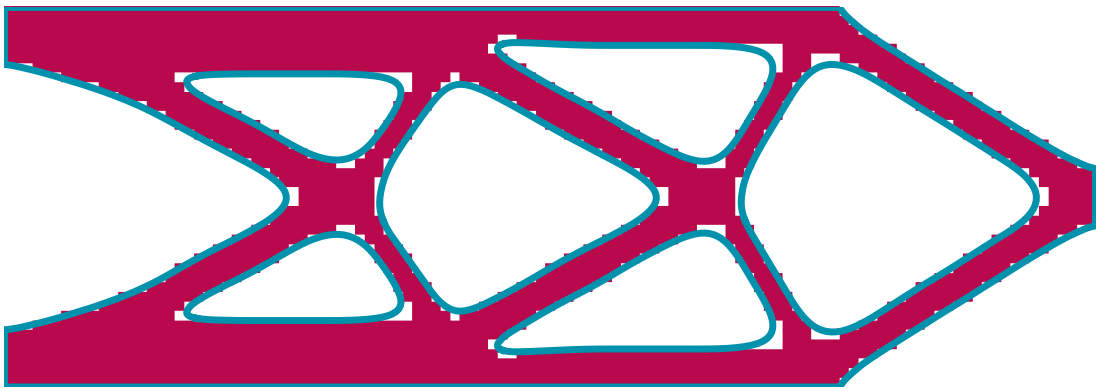
- n: Este parámetro controla el número de coordenadas escogidas antes y después de la actual. Es decir, controla en número de puntos a usar para hacer la media de coordenadas. Normalmente se usa un valor $\in [2, 8]$, dependiendo del tipo de problema. Para valores muy grandes, tiende a redondear en exceso las esquinas.

- r: Este parámetro controla el número de veces que se aplica el algoritmo de suavizado. Al repetir varias veces el proceso de suavizado sobre la línea, se realiza un suavizado mucho menos agresivo que elimina ciertas inestabilidades, sin redondear las esquinas.

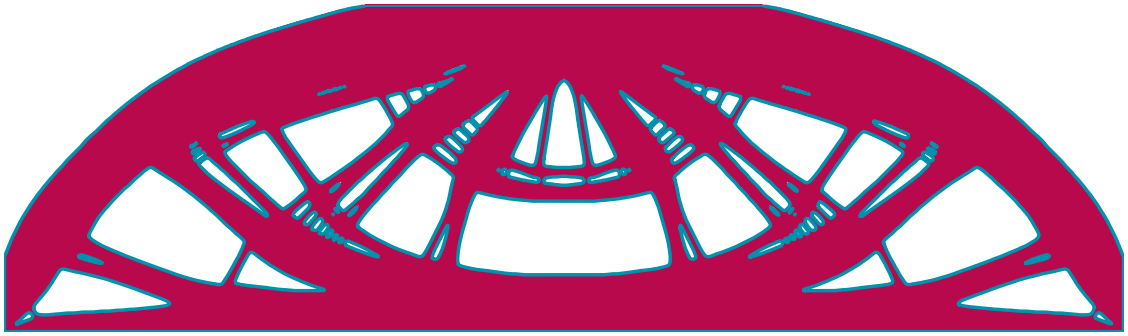
4.2.1. Ejemplos

En las Figuras 4.1 y 4.2 se muestran ejemplos donde se ha aplicado el suavizado de los contornos. Se puede observar que para casos con un pequeño número de elementos o agujeros pequeños, hay que ajustar con cuidado los parámetros del algoritmo para que no se altere en exceso la geometría de los contornos.

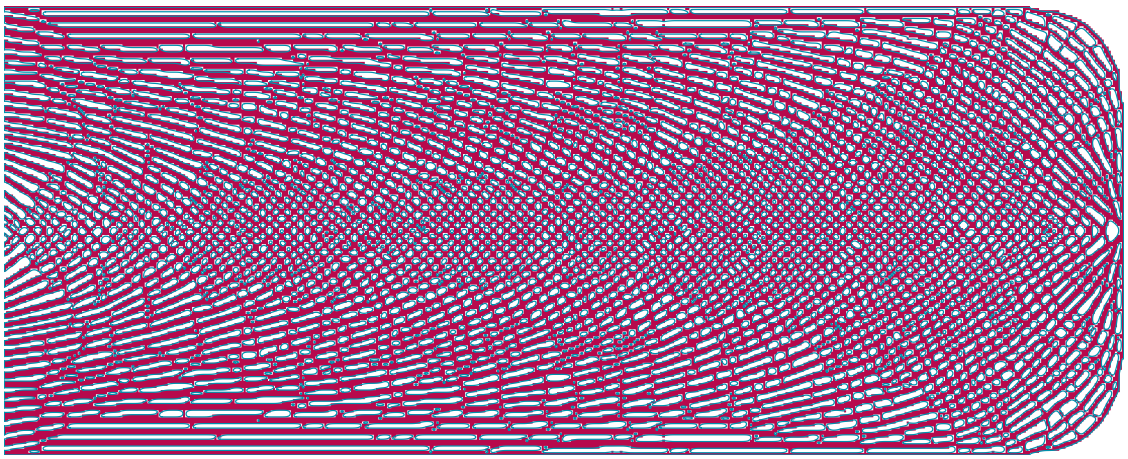
Además, como el algoritmo de trazado, redondea las esquinas de contornos muy pequeños, se pone una restricción de que si la curva tiene un número de puntos < 20, no se aplicará el suavizado.



(a) $r = 2, n = \{4, 5\}$

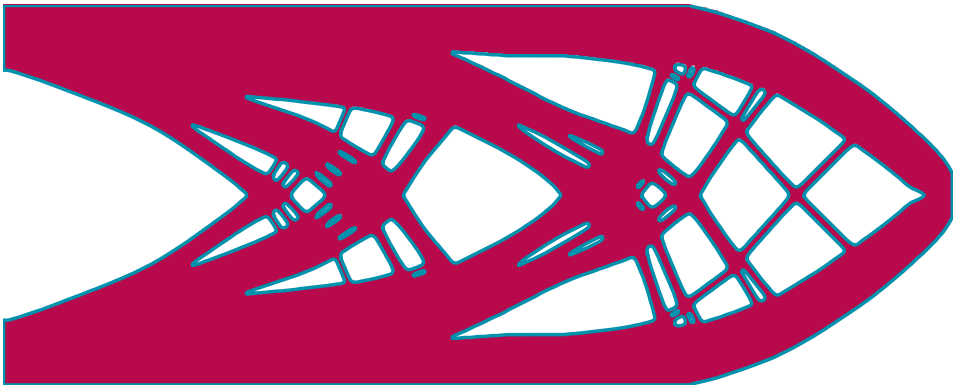


(b) $r = 1, n = \{5\}$

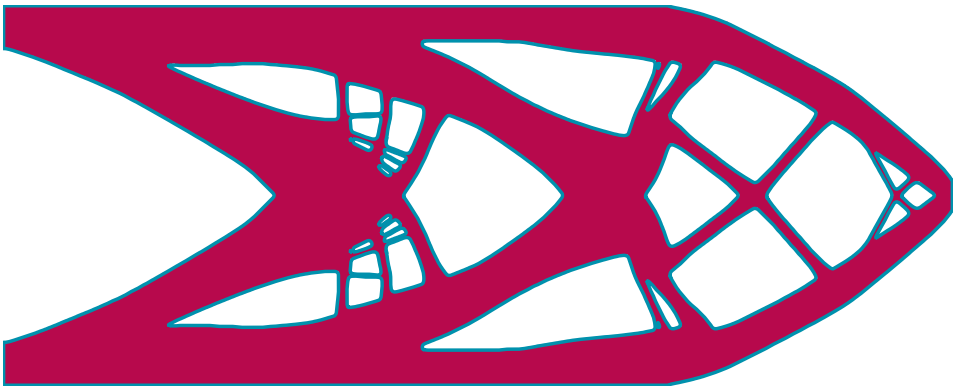


(c) $r = 2, n = \{2, 2\}$

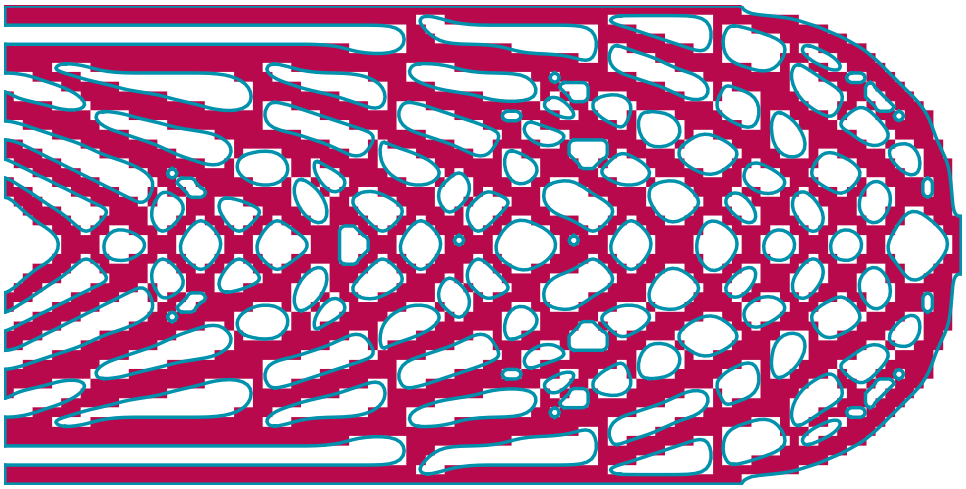
Figura 4.1: Muestra de casos reales resueltos por el algoritmo.



(a) $r = 1, n = \{5\}$



(b) $r = 1, n = \{5\}$



(c) $r = 2, n = \{2, 2\}$

Figura 4.2: Muestra de casos reales resueltos por el algoritmo.

4.3. Simplificación

Tras suavizar los trazados de cada contorno, se debe reducir la cantidad de información de cada uno de ellos. Cada contorno puede estar compuesto de cientos de puntos, muchos de ellos en trazos rectilíneos, aportando información redundante. Además, este excesivo número de coordenadas podría ser problemático a la hora de exportar los trazos al programa CAD y tenga que dibujarse la pieza.

Para hacer esto, se usa el algoritmo de Ramer–Douglas–Peucker (ver Algoritmo 4). Este algoritmo se describe gráficamente en la Figura 4.3, donde se observa que se comienza formando una línea entre los puntos extremos del trazado. Se mide la máxima distancia perpendicular a a línea que une los dos puntos anteriores, y si esta distancia es mayor que la tolerancia ϵ, se reinicia escogiendo el punto inicial y el intermedio, y el intermedio junto con el final. De esta manera, se genera un trazado que guarda los puntos más importantes para definir el contorno, e ignora los menos relevantes.

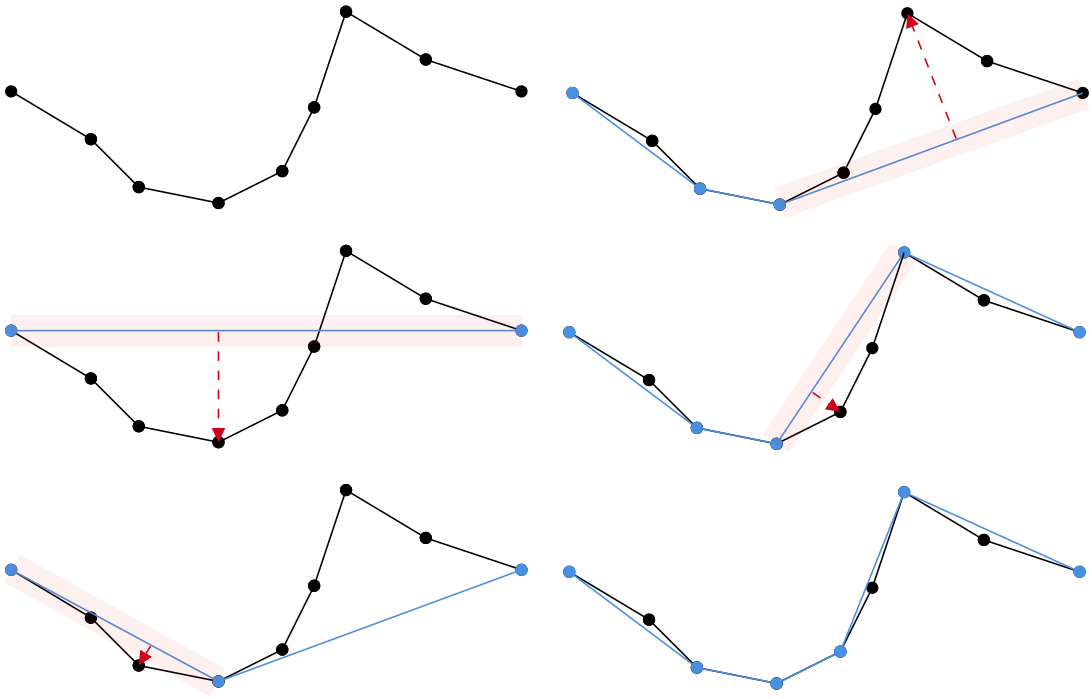


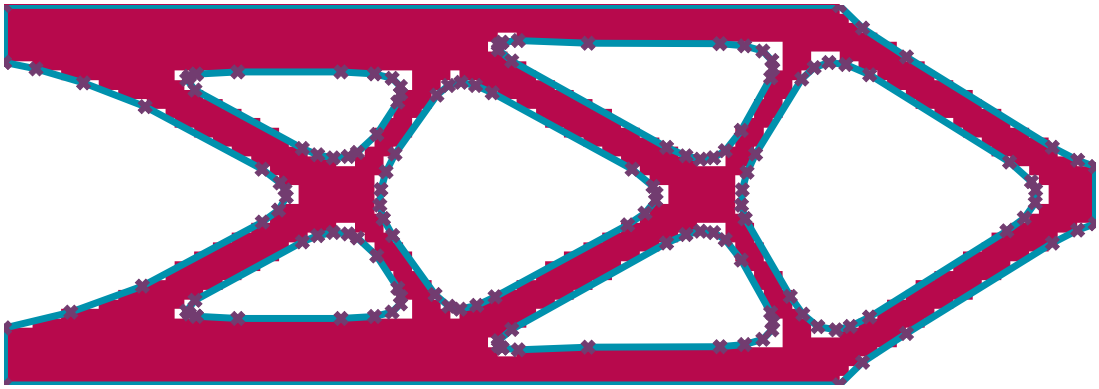
Figura 4.3: Ejemplo gráfico del algoritmo de Ramer–Douglas–Peucker.

Algoritmo 4 Algoritmo de Ramer–Douglas–Peucker

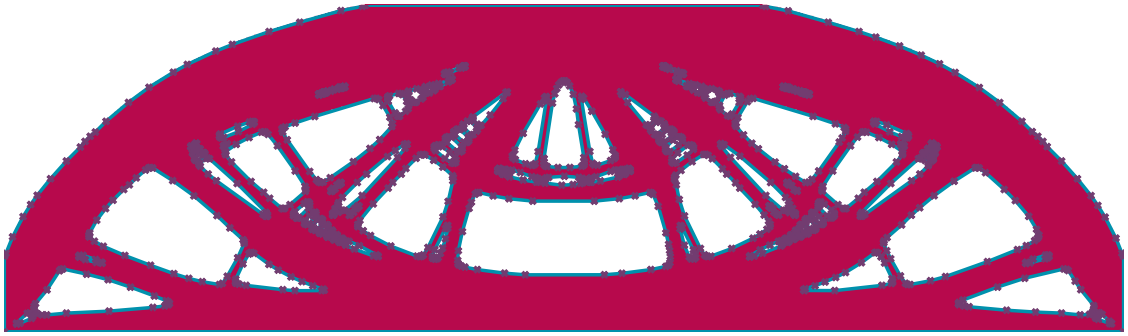
procedure DOUGLASPEUCKER($PointList[]$, ϵ)
  $d_{\text{máx}} \leftarrow 0$
  $index \leftarrow 0$
  $end \leftarrow length(PointList)$
  for $i = 2 \rightarrow end - 1$ **do**
    $dgets$ DistMin($PointList[i]$, $PointList[1]$, $PointList[end]$)
    if $d > d_{\text{máx}}$ **then**
      $d_{\text{máx}} \leftarrow d$
      $index \leftarrow i$
    end if
  end for
  if $d_{\text{máx}} > \epsilon$ **then**
    $R_1[] \leftarrow$ DouglasPeucker($PointList[1, ..., index]$, ϵ)
    $R_2[] \leftarrow$ DouglasPeucker($PointList[index, ..., end]$, ϵ)
    $R[] \leftarrow \{R_1, R_2\}$
  else
    $R[] \leftarrow \{PointList[1], PointList[end]\}$
  end if
  return R
end procedure
procedure DISTMIN(p, p_i, p_f)
  $a_1 \leftarrow p_f.x - p_i.x$
  $a_2 \leftarrow p_f.y - p_i.y$
  return $\dfrac{|a_2 \cdot p.x - a_1 \cdot p.y - p_i.x \cdot a_2 + p_i.y \cdot a_1|}{\sqrt{a_2^2 + a_1^2}}$
end procedure

4.3.1. Ejemplos

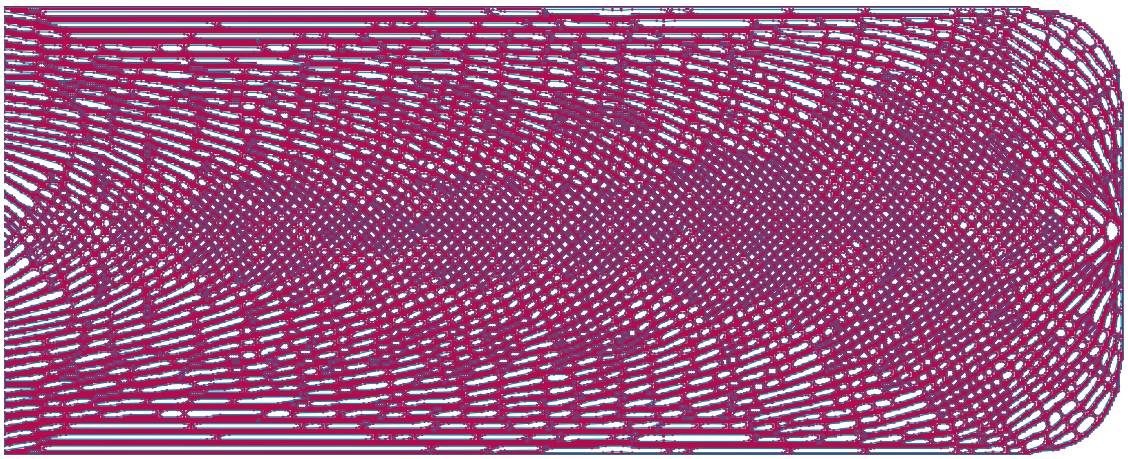
En las Figuras 4.4 y 4.5 se muestran ejemplos de imágenes con los trazos simplificados. Se observa que aparecen contornos compuestos de muchas caras que en una imagen CAD podría ser poco deseable, ya que se pierde la apariencia de continuidad. Pero esto no se ha considerado problemático para la impresión 3D, ya que debido a las tolerancias de la máquina, al tamaño de los agujeros y al bajo cambio de pendiente entre las diferentes caras, no se perdería la sensación de continuidad una vez fabricada la pieza.



(a) $\epsilon = 0{,}25$

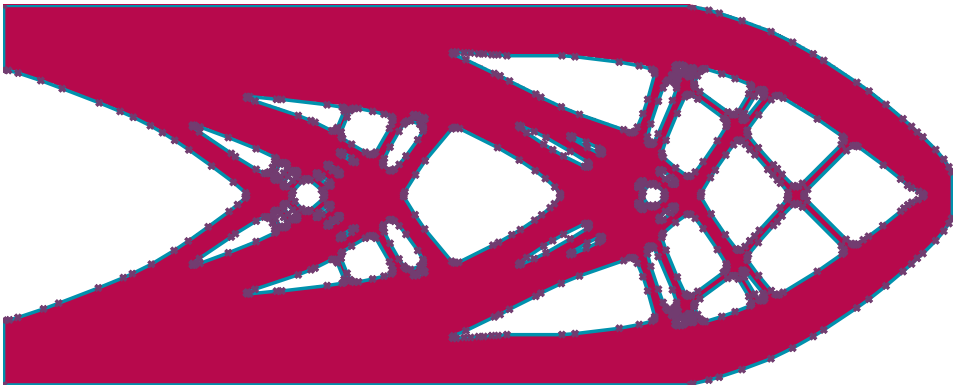


(b) $\epsilon = 0{,}25$

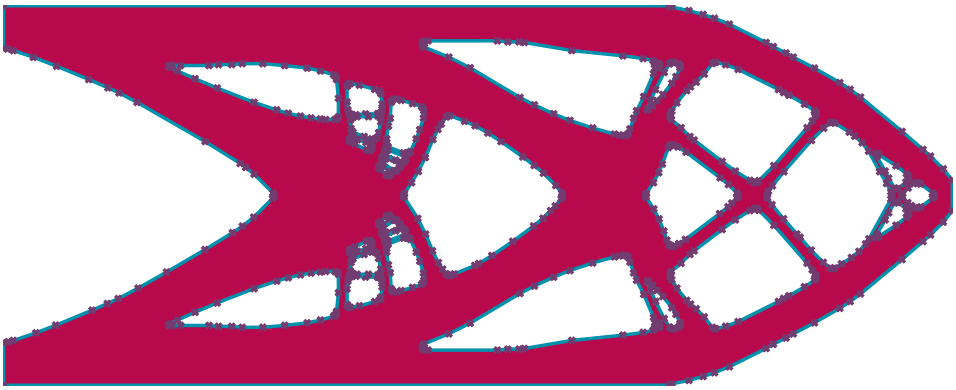


(c) $\epsilon = 0{,}25$

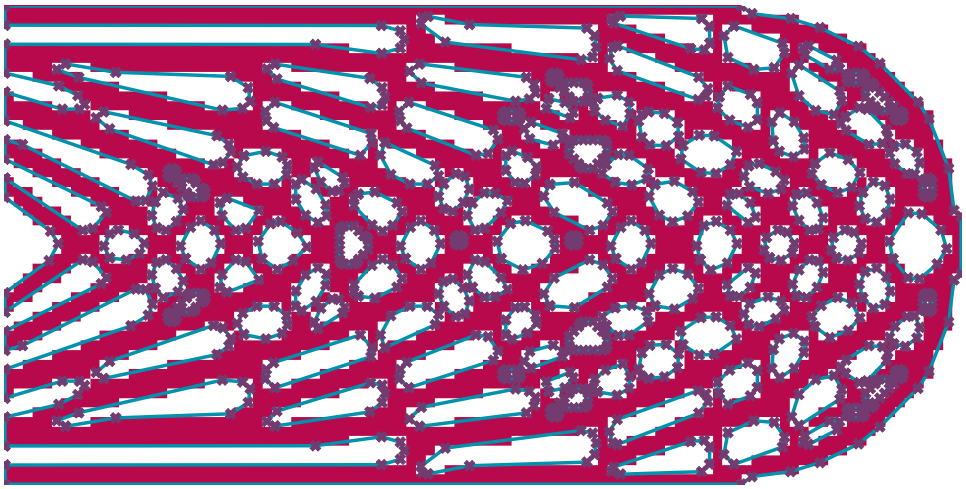
Figura 4.4: Muestra de casos reales resueltos por el algoritmo.



(a) $\epsilon = 0{,}25$



(b) $\epsilon = 0{,}25$



(c) $\epsilon = 0{,}25$

Figura 4.5: Muestra de casos reales resueltos por el algoritmo.

4.4. Generación de la pieza en formato CAD

Como se mencionó anteriormente, todo el proyecto se ha realizado usando el lenguaje de programación Python. Para crear las piezas en formato CAD, se han considerado principalmente dos softwares: Ansys SpaceClaim y FreeCAD. Ambos tienen compatibilidad con Python, siendo el entorno ideal para generar un Script que modele las piezas a partir del conjunto de listas que componen los contornos. De entre ambos softwares, se ha optado por trabajar con FreeCAD, por su carácter de software libre y por tener una mayor eficiencia generando las piezas mediante Script.

Para generar las piezas, los vectores referidos a los diferentes contornos se han separado en dos grupos, los que son externos y los internos. Esto se debe a que la idea es generar un croquis con todos los bordes externos para extruirlo en primer lugar, y después extrudir el corte de cada agujero por separado. Se ha hecho así porque generar un único croquis con todos los trazos directamente, resultaba muy costoso computacionalmente, ya que FreeCAD debe restringir todos los grados de libertad de cada línea que compone cada contorno.

Si se observa el árbol de operaciones en la Figura 4.6 y la Figura 4.7, puede comprobarse lo dicho anteriormente. Para formar la pieza, primero se extruye todo el contorno exterior. Tras esto, se extruyen por separado cada uno de los agujeros internos, y al final, se aplica una operación booleana para restar el sólido "exterior" a los sólidos correspondientes a los agujeros. El resultado de todo este proceso se muestra en las Figuras 4.8 y 4.9.

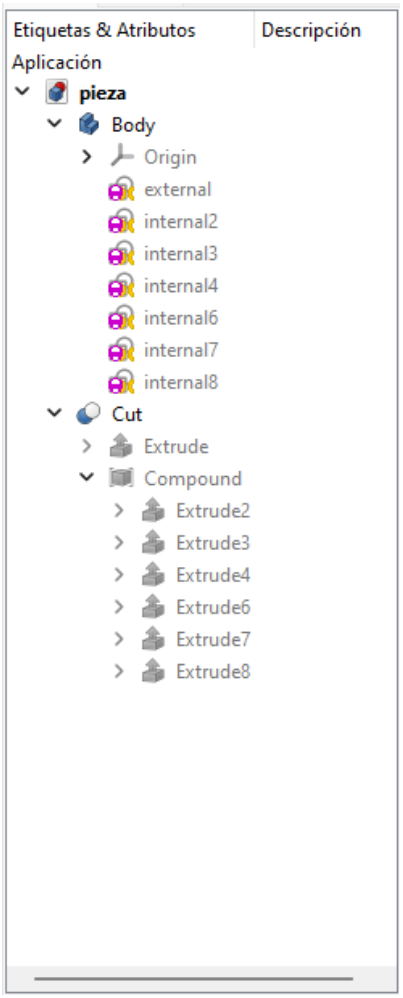


Figura 4.6: Árbol de operaciones de FreeCAD.

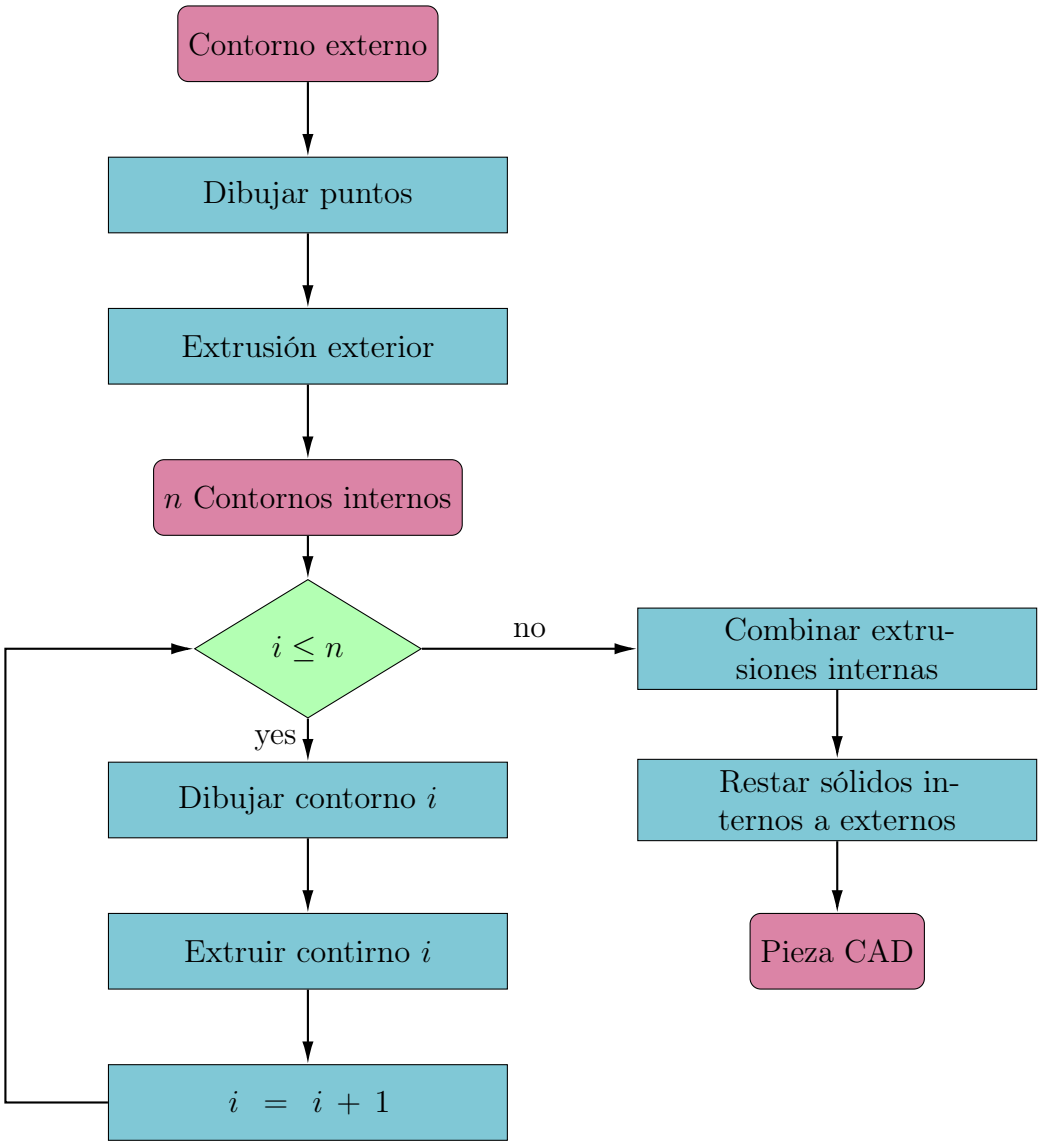
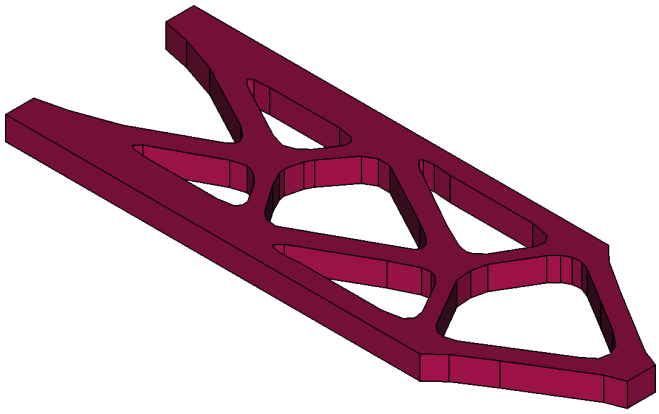
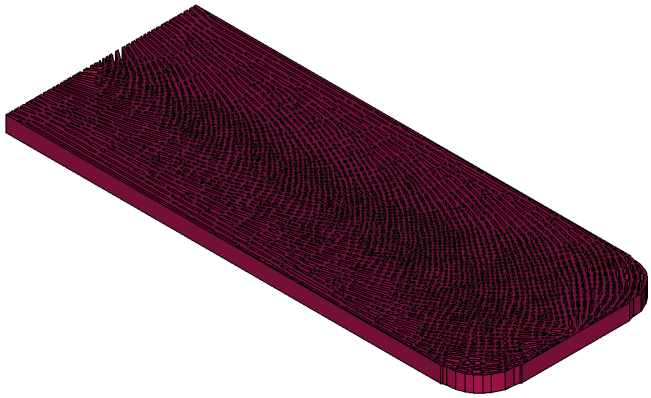


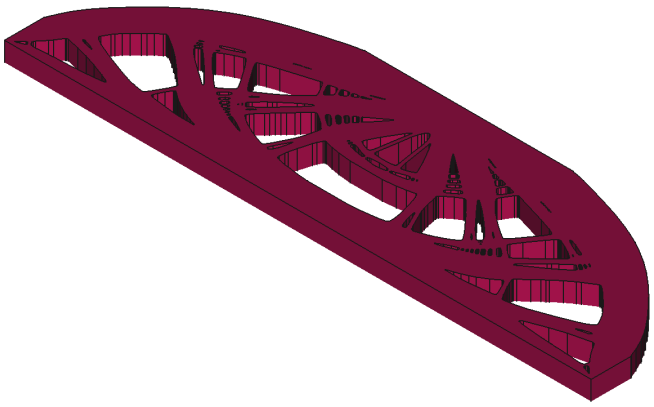
Figura 4.7: Diagrama de flujo de exportación a FreeCAD.



(a)

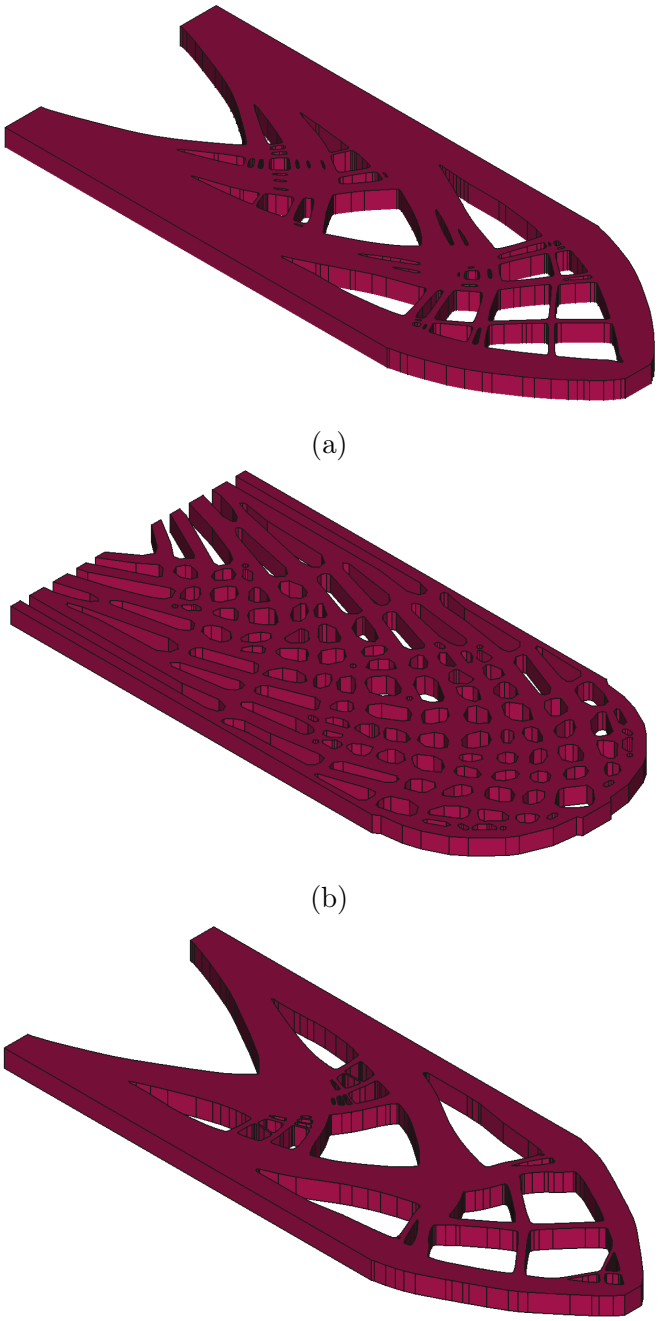


(b)



(c)

Figura 4.8: Muestra de casos reales resueltos por el algoritmo.



(a)

(b)

(c)

Figura 4.9: Muestra de casos reales resueltos por el algoritmo.

Parte III

Conclusiones y líneas futuras

Capítulo 5

Conclusiones

Contenido

5.1. Conclusiones

En este proyecto se ha propuesto un método para exportar los resultados de optimización topológica a archivos CAD, con el objetivo de fabricar estos resultados mediante fabricación aditiva.

Los Capítulos 2 y 3 se proponen los algoritmos fundamentales para extraer los contornos límites entre los diferentes dominios. Lo expuesto en el Capítulo 2 permite obtener de manera eficiente los puntos de partida para trazar cada contorno. Mientras que en el Capítulo 3 se explican los axiomas necesarios, para obtener los conjuntos de coordenadas ordenadas que dibujan el contorno.

Tras haber conseguido los conjuntos de coordenadas, se observan que estos se componen de un número datos muy grande y que presenta cierta irregularidad. En el Capítulo 4, se explica cómo se tratan estos conjuntos con el objetivo de que los trazos sean más regulares y simples. Para ello se utilizan dos algoritmos: el primero define la posición de cada coordenada, en función de las n coordenadas anteriores

y posteriores, haciendo que las líneas adquieran un aspecto mucho más regular. El segundo algoritmo, se trata del "Algoritmo de Ramer–Douglas–Peucker", que se trata de un proceso iterativo, que elimina la información redundante de los trazados, usando como criterio la máxima distancia perpendicular y una tolerancia ϵ. Los datos extraídos de todo este proceso, se llevan a FreeCAD para crear un modelo tridimensional que pueda usarse para exportar los resultados a una impresora 3D.

Todo se ha programado en el lenguaje de programación Python, creando una librería, donde, dando como entrada la matriz de resultados de optimización topológica, se obtenga como salida un conjunto de listas de puntos, que posteriormente, mediante un script de FreeCAD, generarán la pieza.

Si se observa la Figura 4.4 y Figura 4.5, donde se muestran los resultados finales de aplicación del algoritmo, podría objetarse que el ajuste de los trazados no es perfecto a la pieza original. No se debe olvidar que los métodos propuestos, son métodos de interpretación de resultados de manera sistemática, no son la solución real. Lo cerca o lo lejos que esté el trazado de parecerse a la solución real, dependerá de la definición de esta misma, o en otras palabras, de la cantidad de información que se posea, materializada en la densidad de elementos finitos. Así pues, comparando las Figuras 4.4(a), 4.5(a) y 4.5(c), con las Figuras 4.4(b), 4.4(c) y 4.5(b), cuando la densidad de elementos es muy alta, la solución de optimización topológica y el trazado de los contornos es prácticamente indistinguible.

Respecto a la velocidad de todos estos algoritmos, no se puede afirmar que sea más o menos eficiente, pues no se ha encontrado un similar con el que hacer comparaciones. Lo que sí se puede afirmar, es que la velocidad de generación de los trazados es satisfactoria, consiguiendo escribir los archivos de las soluciones más complejas (como la Figura 4.5(a)), en al menos 10 segundos, dependiendo de la potencia de la computadora. Además, debe recordarse que se ha usado un lenguaje de programación interpretado. Una traducción a un lenguaje compilado, como podría ser $C++$, mejoraría mucho la velocidad de generación de las piezas.

Por último, sobre el uso de FreeCAD para generar las piezas (Figura 4.8 y Figura 4.9), aunque en la mayoría de los casos funciona suficientemente rápido, para los resultados de infill con una gran densidad de agujeros (ver Figura 4.8(b)), puede tomar un tiempo excesivo, siendo muy dependiente de la capacidad de la computadora. Aún así, se ha observado que en general, todos los softwares tienen dificultades para trabajar con esta pieza, posiblemente a la enorme cantidad de caras que posee.

5.1.1. Piezas fabricadas

Para concluir los objetivos del trabajo, los modelos CAD generados, se han exportado a STL para su posterior fabricación mediante FA. Los resultados de esto pueden verse en la Figura 5.1.

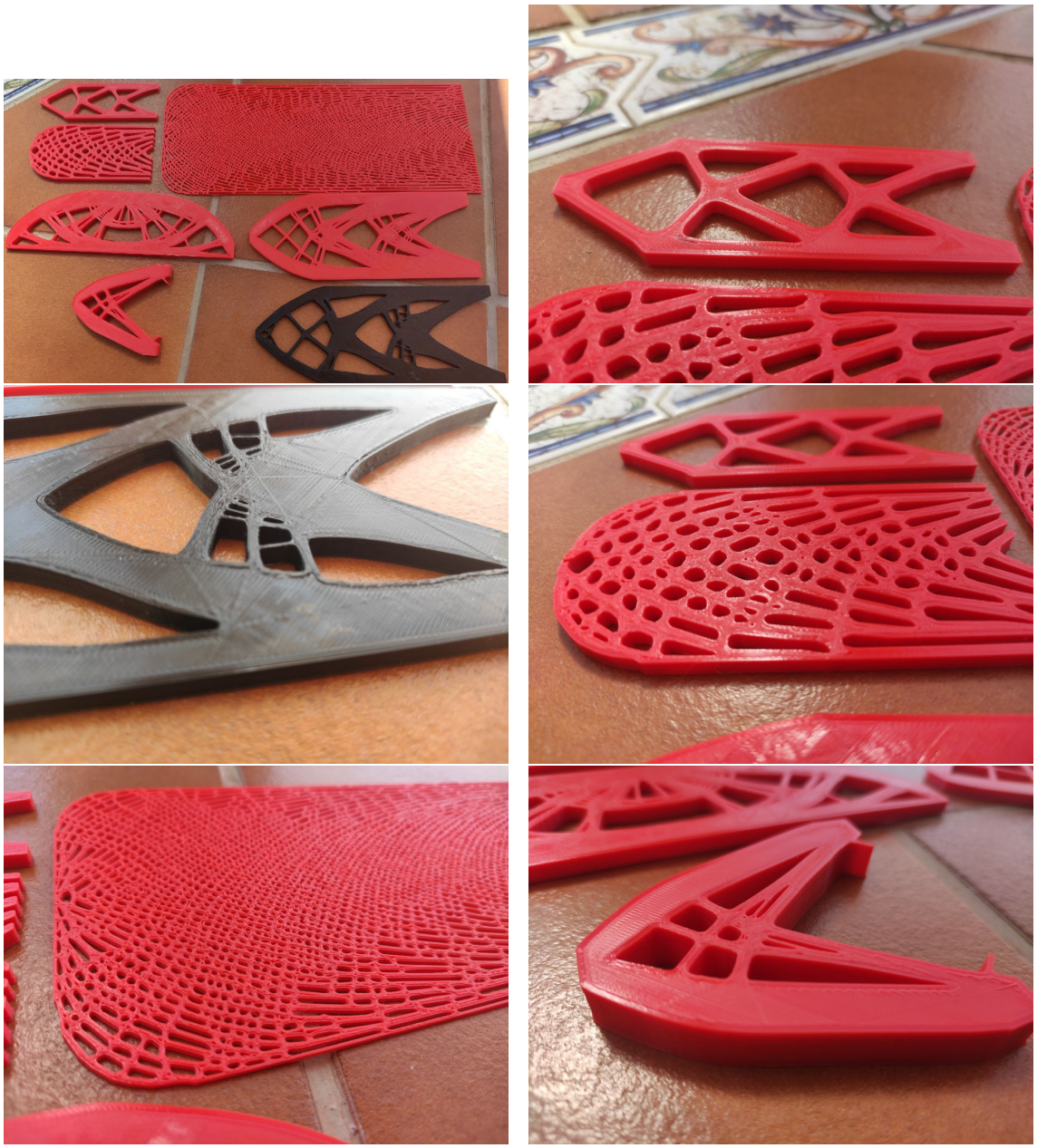


Figura 5.1: Piezas fabricadas mediante FA.

5.2. Líneas futuras

5.2.1. Exportar resultados a CAD

Uno de los mayores problemas encontrados tras la realización de este proyecto, es la generación de la pieza CAD, debido a que para piezas muy complejas puede tomar un tiempo excesivo. Además, los comandos de Script de FreeCAD, no parecen pensados para generar piezas de manera sencilla, resultando una programación algo complicada.

Existen algunas alternativas a FreeCAD que podrían resultar interesantes para futuros desarrollo: OpenSCAD [29] y CADquery [31]; ambos softwares CAD basados únicamente en "scripting". OpenSCAD utiliza un lenguaje propio, mientras que CADquery está basado en python.

5.2.2. Generación de modelos 3D

Una de los desarrollos naturales de este proyecto, es ampliar el algoritmo para generar modelos 3D. Esto sería generando los trazados de cortes a la pieza por diferentes planos, y uniendo los diferentes contornos para generar el sólido.

5.2.3. Generalización de elementos

Este proyecto se ha realizado bajo la suposición de que los elementos finitos son todos cuadrados y del mismo tamaño. Este algoritmo podría ampliarse también para hacerlo funcionar con cualquier tipo de elemento finito bidimensional

Parte IV

Anexos

Anexo A

Código

Contenido

A.1. Trazado de contornos

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math as m
import os
import pickle
import json
import string
import time



class pila:
    def __init__(self, lista: list, dim: int):
        self.lista = lista
        self.dim = dim

    def agregar(self, valor):
        self.lista.pop(-1)
        self.lista.insert(0, valor)

```

```
22 def agregar(x, i, j, i1, j1, i2, j2, pp, h):
23     if x[j][i] == 0:
24         pp.append((i, j))
25         x[j][i] = h
26         if x[j1][i1] == 0:
27             pp.append((i1, j1))
28             x[j1][i1] = h
29         if x[j2][i2] == 0:
30             pp.append((i2, j2))
31             x[j2][i2] = h
32 
33 def agujero(p, x, nelx, nely, h):
34     pp = []
35     for l in p:
36         i = l[0]
37         j = l[1]
38         i1 = int(np.maximum(i - 1, 0))
39         i2 = int(np.minimum(i + 1, nelx - 1))
40         j1 = int(np.maximum(j - 1, 0))
41         j2 = int(np.minimum(j + 1, nely - 1))
42         agregar(x, i, j1, i1, j1, i2, j1, pp, h)
43         agregar(x, i, j2, i1, j2, i2, j2, pp, h)
44         agregar(x, i1, j, i1, j1, i1, j2, pp, h)
45         agregar(x, i2, j, i2, j1, i2, j2, pp, h)
46 
47     if len(pp) > 0:
48         agujero(pp.copy(), x, nelx, nely, h)
49 
50 def cambio(x, nelx, nely):
51     h = []
52     a = x.copy()
53     a = a.tolist()
54     p = []
55     sentido = []
56     if a[0][0] == 0:
57 
58         for j in range(1,len(a)):
59             if a[j][0]==1:
60                 p.append((0, j-1))
61                 h.append((10 * len(h) + 10))
62                 a[j-1][0] = h[-1]
63                 if j-1 != 0:
64                     sentido.append(False)
65                 else:
66                     sentido.append(True)
67                 agujero([(0, j-1)], a, nelx, nely, h[-1])
68                 break
69     if a[len(a)-1][0] == 0:
70         for j in range(len(a)-1,-1,-1):
```

```
71              if a[j][0]==1:
72
73                  p.append((0, j+1))
74                  h.append((10 * len(h) + 10))
75                  a[j-1][0] = h[-1]
76                  if j+1 != 0:
77                      sentido.append(False)
78                  else:
79                      sentido.append(True)
80                  agujero([(0, j+1)], a, nelx, nely, h[-1])
81                  break
82
83
84      for i in range(nelx):
85          for j1, j2 in zip(range(0, int(nely / 2) + 1), range(nely -
        1, int(nely / 2) - 1, -1)):
86
87              if a[j1][i] == 0:
88                  p.append((i, j1))
89                  h.append((10 * len(h) + 10))
90                  a[j1][i] = h[-1]
91                  if i != nelx-1 and j1 != 0:
92                      sentido.append(False)
93                  else:
94                      sentido.append(True)
95                  agujero([(i, j1)], a, nelx, nely, h[-1])
96              if a[j2][i] == 0:
97                  p.append((i, j2))
98                  h.append((10 * len(h) + 10))
99                  a[j2][i] = h[-1]
100                 if i != nelx - 1 and j2 != nely-1:
101                     sentido.append(True)
102                 else:
103                     sentido.append(False)
104                 agujero([(i, j2)], a, nelx, nely, h[-1])
105     return p, sentido
106
107 def clock(horario: bool, reloj: int, i: int, j: int, dim: tuple):
108     i1 = int(np.maximum(i - 1, 0))
109     i2 = int(np.minimum(i + 1, dim[0] - 1))
110     j1 = int(np.maximum(j - 1, 0))
111     j2 = int(np.minimum(j + 1, dim[1] - 1))
112     i3 = i + 1
113     j3 = j + 1
114     ip = np.array([i, i, i, (i + i3) / 2, i3, i3, i3, (i + i3) /
        2])
115     jp = np.array([j, (j + j3) / 2, j3, j3, j3, (j + j3) / 2, j, j
        ])
116     a = 0
```

```
117     b = 0
118     c = 0
119     d = 0
120     if i == 0:
121         a = 1
122     elif i == dim[0] - 1:
123         c = 1
124     if j == 0:
125         b = 1
126     elif j == dim[1] - 1:
127         d = 1
128 
129     if not horario:
130         borde = a * 2 ** 3 + b * 2 ** 2 + c * 2 ** 1 + d
131         if reloj == 0:
132             ik = [i1, i1, i]
133             jk = [j, j2, j2]
134             ind = [0, 1, 2, 3, 4, 5, 6, 7]
135             i_ = ip[ind]
136             j_ = jp[ind]
137 
138         elif reloj == 1:
139             ik = [i, i2, i2]
140             jk = [j2, j2, j]
141             ind = [2, 3, 4, 5, 6, 7, 0, 1]
142             i_ = ip[ind]
143             j_ = jp[ind]
144 
145         elif reloj == 2:
146             ik = [i2, i2, i]
147             jk = [j, j1, j1]
148             ind = [4, 5, 6, 7, 0, 1, 2, 3]
149             i_ = ip[ind]
150             j_ = jp[ind]
151 
152         elif reloj == 3:
153             ik = [i, i1, i1]
154             jk = [j1, j1, j]
155             ind = [6, 7, 0, 1, 2, 3, 4, 5]
156             i_ = ip[ind]
157             j_ = jp[ind]
158 
159         elif reloj >= 4:
160             reloj = int(reloj - np.floor(reloj / 4) * 4)
161             ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i,
    j, dim)
162         elif reloj < 0:
163             reloj = int(reloj + 4)
```

```
164             ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i,
    j, dim)
165
166         return ik, jk, reloj, i_, j_, borde
167     else:
168
169         borde = a * 2 ** 3 + d * 2 ** 2 + c * 2 ** 1 + b
170
171         if reloj == 0:
172             ik = [i1, i1, i]
173             jk = [j, j1, j1]
174             ind = [2, 1, 0, 7, 6, 5, 4, 3]
175             i_ = ip[ind]
176             j_ = jp[ind]
177         elif reloj == 1:
178             ik = [i, i2, i2]
179             jk = [j1, j1, j]
180             ind = [0, 7, 6, 5, 4, 3, 2, 1]
181             i_ = ip[ind]
182             j_ = jp[ind]
183
184
185         elif reloj == 2:
186             ik = [i2, i2, i]
187             jk = [j, j2, j2]
188             ind = [6, 5, 4, 3, 2, 1, 0, 7]
189             i_ = ip[ind]
190             j_ = jp[ind]
191
192         elif reloj == 3:
193             ik = [i, i1, i1]
194             jk = [j2, j2, j]
195             ind = [4, 3, 2, 1, 0, 7, 6, 5]
196             i_ = ip[ind]
197             j_ = jp[ind]
198
199         elif reloj >= 4:
200             reloj = int(reloj - np.floor(reloj / 4) * 4)
201             ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i,
    j, dim)
202         elif reloj < 0:
203             reloj = int(reloj + 4)
204             ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i,
    j, dim)
205         return ik, jk, reloj, i_, j_, borde
206
207 def casosbinarios(ik, jk, x):
208     v = []
209     for i in range(len(ik)):
```

```
210             v.append(x[jk[i], ik[i]])
211     caso = 0
212     for j in range(len(v)):
213         caso += v[j] * 2 ** (2 - j)
214     return caso
215 
216 def trazar(i, j, x_, paso, horario=False):
217     fin = False
218     dim = (len(x_[0]), len(x_))
219     x = []
220     y = []
221     reloj = 2
222 
223     ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i, j, dim)
224     bc = casosbinarios(ik, jk, x_)
225 
226     inicio = (i, j, reloj)
227 
228     paso.append((i, j))
229 
230     abierto = False
231     if borde != 0:
232         abierto = True
233         if borde == 0 or borde == 9 or borde == 8:
234             x.append(i_[4])
235             y.append(j_[4])
236         elif borde == 1 or borde == 3:
237             x.append(i_[6])
238             y.append(j_[6])
239             reloj = 3
240             ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i,
    j, dim)
241             bc = casosbinarios(ik, jk, x_)
242         elif borde == 2:
243             x.append(i_[0])
244             y.append(j_[0])
245             reloj = 0
246             ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i,
    j, dim)
247             bc = casosbinarios(ik, jk, x_)
248     bca = pila(["p", "p"], 2)
249     control = pila([(i, j, reloj), "p", "p"], 3)
250     while not fin:
251         paso.append((i, j))
252         control.agregar((i, j, reloj))
253         if bc == 0:
254             reloj += 1
255         elif bc == 1:
256             reloj += 1
```

```
257             i = ik[1]
258             j = jk[1]
259             x.append(i_[3])
260             y.append(j_[3])
261         elif bc == 2:
262             reloj += 1
263             i = ik[2]
264             j = jk[2]
265             x.append(i_[4])
266             y.append(j_[4])
267             print("Caso 2 en", i, j)
268         elif bc == 3:
269             i = ik[0]
270             j = jk[0]
271             x.append(i_[3])
272             y.append(j_[3])
273         elif bc == 4:
274             i = ik[2]
275             j = jk[2]
276             x.append(i_[3])
277             y.append(j_[3])
278             print("Caso 4 en", i, j)
279         elif bc == 5:
280             reloj -= 1
281             x.append(i_[3])
282             y.append(j_[3])
283             if bca.lista[0] != 1:
284                 x.append((i_[3] + i_[2] + i_[1]) / 3)
285                 y.append((j_[3] + j_[2] + j_[1]) / 3)
286         elif bc == 6:
287             reloj += 1
288             i = ik[2]
289             j = jk[2]
290             x.append(i_[4])
291             y.append(j_[4])
292         elif bc == 7:
293             reloj -= 1
294             if bca.lista[0] == 7 and (bca.lista[1] != 7 or bca.
    lista[1] != "p"):
295                 x.append((i_[3] + i_[4] + i_[5]) / 3)
296                 y.append((j_[3] + j_[4] + j_[5]) / 3)
297             x.append(i_[3])
298             y.append(j_[3])
299             if bca.lista[0] != 1:
300                 x.append((i_[3] + i_[2] + i_[1]) / 3)
301                 y.append((j_[3] + j_[2] + j_[1]) / 3)
302         bca.agregar(bc)
303         ik, jk, reloj, i_, j_, borde = clock(horario, reloj, i, j,
    dim)
```

```
304         if borde != 0:
305             abierto = True
306         bc = casosbinarios(ik, jk, x_)
307         if not abierto and (i, j, reloj) == inicio:
308             fin = True
309             x.append(x[0])
310             y.append(y[0])
311         elif abierto and len(x) > 1 and borde != 0:
312             if control.lista[0] == control.lista[1] == control.
    lista[2]:
313                 fin = True
314                 x.append(i_[2])
315                 y.append(j_[2])
316 
317     return x, y, paso
318 
319 def vect(x_, suavizar=False, factor=4, reducir=False):
320     nelx = len(x_[0])
321     nely = len(x_)
322     p, sentido = cambio(x_, nelx, nely)
323     paso = []
324     contornos = []
325 
326     for k in range(len(p)):
327         x, y, paso = trazar(p[k][0], p[k][1], x_, paso, horario=
    sentido[k])
328         if x[0] != x[-1] or y[0] != y[-1]:
329 
330             contornos.append(Contorno(x.copy(), y.copy(), True))
331         else:
332             contornos.append(Contorno(x.copy(), y.copy(), False))
333 
334     ii = [[k for k in range(0, nelx)], [nelx - 1] * nely, [k for k
    in range(nelx - 1, -1, -1)], [0] * nely]
335 
336     jj = [[0] * nelx, [k for k in range(0, nely)], [nely - 1] *
    nelx, [k for k in range(nely - 1, -1, -1)]]
337     inicio = False
338     x = []
339     y = []
340 
341     for k in range(len(ii)):
342         if k == 0:
343             re = 2
344         elif k == 1:
345             re = 1
346         elif k == 2:
347             re = 0
348         elif k == 3:
```

```
            re = 3
        inicio = False
        for l in range(len(ii[k])):
            i = ii[k][l]
            j = jj[k][l]
            i_, j_ = node_coordinates(re, i, j, (nelx, nely))
            if x_[j, i] == 1 and not inicio:
                x.append(i_[4])
                y.append(j_[4])
                inicio = True
            elif inicio and x_[j, i] == 0:
                x.append(i_[4])
                y.append(j_[4])
                inicio = False
                contornos.append(Contorno(x.copy(), y.copy(), True,
    limite=True))
                x = []
                y = []
            if inicio and l == len(ii[k]) - 1:
                x.append(i_[2])
                y.append(j_[2])
                inicio = False
                contornos.append(Contorno(x.copy(), y.copy(), True,
    limite=True))
                x = []
                y = []

    return contornos

def filter(x_sol):
    filtered = x_sol.copy()
    for i in range(len(x_sol)):
        for j in range(len(x_sol[0])):
            if x_sol[i, j] < 0.5:
                filtered[i, j] = 0
            elif x_sol[i, j] >= 0.5:
                filtered[i, j] = 1
    return filtered

class Contorno:
    def __init__(self, x, y, externo: bool, limite=False):

        self.x = np.array(x)
        self.y = np.array(y)
        self.externo = externo
        self.limite = limite

    def suavizado(self, dist):
        x = []
```

```
396         y = []
397         if len(self.x) <= 20:
398             pass
399         elif self.externo == False:
400             for i in range(len(self.x) + 1):
401                 x_, y_ = promedio_c(self.x, self.y, i - dist, i +
    dist)
402                 x.append(x_)
403                 y.append(y_)
404             self.x = np.array(x)
405             self.y = np.array(y)
406         else:
407             x.append(self.x[0])
408             y.append(self.y[0])
409             for i in range(1, len(self.x) - 1):
410                 x_, y_ = promedio_e(self.x, self.y, dist, i)
411                 x.append(x_)
412                 y.append(y_)
413             x.append(self.x[-1])
414             y.append(self.y[-1])
415             self.x = np.array(x)
416             self.y = np.array(y)
417 
418     def simplificar(self,ep, lim ):
419 
420         if len(self.x) <= lim:
421             if not self.externo and not self.limite:
422                 n2 = int(len(self.x) / 2)
423                 x, y = douglaspeucker(self.x[0:n2 + 1], self.y[0:n2
     + 1], 0.01)
424                 x2, y2 = douglaspeucker(self.x[n2:2 * n2 + 1], self
    .y[n2:2 * n2 + 1], 0.01)
425                 x += x2[1:]
426                 y += y2[1:]
427                 self.x = np.array(x)
428                 self.y = np.array(y)
429             else:
430                 x, y = douglaspeucker(self.x, self.y, 0.01)
431                 self.x = np.array(x)
432                 self.y = np.array(y)
433         elif not self.externo and not self.limite:
434             n2 = int(len(self.x) / 2)
435             x, y = douglaspeucker(self.x[0:n2 + 1], self.y[0:n2 +
    1], ep)
436             x2, y2 = douglaspeucker(self.x[n2:2 * n2 + 1], self.y[
    n2:2 * n2 + 1], ep)
437             x += x2[1:]
438             y += y2[1:]
439             self.x = np.array(x)
```

```
440             self.y = np.array(y)
441         else:
442             x, y = douglaspeucker(self.x, self.y, ep)
443             self.x = np.array(x)
444             self.y = np.array(y)
445 
446 def promedio_c(x, y, inf, sup):
447     l = len(x)
448     N = sup - inf
449     sumx = 0
450     sumy = 0
451     t = 0
452     for i in range(inf, sup + 1):
453         t += 1
454         if 0 <= i < l:
455             sumx += x[i]
456             sumy += y[i]
457         elif i >= l:
458             sumx += x[-l + i]
459             sumy += y[-l + i]
460         else:
461             sumx += x[l + i]
462             sumy += y[l + i]
463     return sumx / t, sumy / t
464 
465 
466 def promedio_e(x, y, N, i):
467     l = len(x)
468     inf = i - N
469     sup = i + N
470     sumx = 0
471     sumy = 0
472     t = 0
473     for i in range(inf, sup + 1):
474         t += 1
475         if 0 <= i < l:
476             sumx += x[i]
477             sumy += y[i]
478         elif i >= l:
479             sumx += x[-1]
480             sumy += y[-1]
481         else:
482             sumx += x[0]
483             sumy += y[0]
484     return sumx / t, sumy / t
485 
486 
487 def douglaspeucker(x: list, y: list, epsilon):
488     dmax = 0
```

```
489     index = 0
490     resx = []
491     resy = []
492     for i in range(1, len(x)):
493         d = distancia_pl(x[i], y[i], x[0], x[-1], y[0], y[-1])
494         if d > dmax:
495             index = i
496             dmax = d
497     if dmax > epsilon:
498         res1x, res1y = douglaspeucker(x[0:index + 1], y[0:index +
    1], epsilon)
499         res2x, res2y = douglaspeucker(x[index:len(x)], y[index:len(
    x)], epsilon)
500 
501         resx = resx + res1x.copy() + res2x[1:].copy()
502         resy = resy + res1y.copy() + res2y[1:].copy()
503     else:
504         resx.append(x[0])
505         resx.append(x[-1])
506 
507         resy.append(y[0])
508         resy.append(y[-1])
509 
510     return resx.copy(), resy.copy()
511 
512 
513 def distancia_pl(px, py, xi, xf, yi, yf):
514 
515     a1 = xf - xi
516     a2 = yf - yi
517     return abs(a2 * px - a1 * py - xi * a2 + yi * a1) / np.sqrt(a2
    ** 2 + a1 ** 2)
518 
519 
520 class trazador:
521     def __init__(self, x_sol):
522         self.x_f = filter(x_sol)
523         self.nelx = len(x_sol[0])
524         self.contornos = vect(self.x_f)
525 
526     def suavizar(self, N=2):
527         for i in range(len(self.contornos)):
528             self.contornos[i].suavizado(N)
529 
530     def reducir(self,ep=0.25,lim = 20):
531         for i in range(len(self.contornos)):
532             self.contornos[i].simplificar(ep, lim)
533 
534     def exportar(self, name: string, medida_x=150.0):
```

```
535         ruta_puntos = os.getcwd() + "\\" + name
536         os.mkdir(ruta_puntos)
537         external = dict()
538         internal = dict()
539         r = 0
540         escala = medida_x / self.nelx
541         for i in range(len(self.contornos)):
542             if self.contornos[i].externo == True:
543                 external[i] = [self.contornos[i].x * escala, self.
    contornos[i].y * escala]
544 
545 
546             else:
547                 internal[i] = [self.contornos[i].x * escala, self.
    contornos[i].y * escala]
548 
549         file1 = open(ruta_puntos + "\external.pkl", "wb")
550         file2 = open(ruta_puntos + "\internal.pkl", "wb")
551         pickle.dump(external, file1)
552         pickle.dump(internal, file2)
553         file1.close()
554         file2.close()
```

A.2. FreeCAD

```
1  import PartDesign
2  import Part
3  import Sketcher
4  import pickle
5  import numpy as np
6  import math as m
7  import FreeCAD
8  import Draft
9  
10 file1 = open("...\\external.pkl", "rb")
11 file2 = open("...\\internal.pkl", "rb")
12 external = pickle.load(file1)
13 internal = pickle.load(file2)
14 file1.close()
15 file2.close()
16 def line(x0, y0, xf, yf):
17     return Part.LineSegment(FreeCAD.Vector(x0,y0,0),FreeCAD.Vector(
    xf,yf,0))
18 
19 def pad (doc,L,name, croquis):
20     doc.getObject('Body').newObject('PartDesign::Pad',name)
```

```
21      doc.getObject(name).Profile = doc.getObject(croquis)
22      doc.getObject(croquis).Visibility = False
23      doc.getObject(name).Length = L
24      doc.getObject(name).UseCustomVector = 0
25      doc.getObject(name).Direction = (1, 1, 1)
26      doc.getObject(name).Type = 0
27      doc.getObject(name).UpToFace = None
28      doc.getObject(name).Reversed = 0
29      doc.getObject(name).Midplane = 0
30      doc.getObject(name).Offset = 0
31
32  def pocket (doc,L,name, croquis):
33      doc.getObject('Body').newObject('PartDesign::Pocket',name)
34      doc.getObject(name).Profile = doc.getObject(croquis)
35      doc.getObject(croquis).Visibility = False
36      doc.getObject(name).Length = L
37      doc.getObject(name).UpToFace = None
38      doc.getObject(name).Type = 1
39      doc.getObject(name).UpToFace = None
40      doc.getObject(name).Reversed = 0
41      doc.getObject(name).Midplane = 1
42      doc.getObject(name).Offset = 0
43
44
45  doc = FreeCAD.newDocument("Pieza_TFM")
46  doc.addObject('PartDesign::Body','Body')
47  doc.getObject('Body').newObject('Sketcher::SketchObject','external'
        )
48  d = FreeCAD.getDocument('Pieza_TFM').getObject('external')
49  print(external.keys())
50  for i in external.keys():
51      for j in range(len(external[i][0])-1):
52          d.addGeometry(line(external[i][0][j],external[i][1][j],
        external[i][0][j+1],external[i][1][j+1]), False)
53  doc.addObject('Part::Extrusion','Extrude')
54  f = doc.getObject('Extrude')
55  f.Base = doc.getObject('external')
56  f.DirMode = "Normal"
57  f.DirLink = None
58  f.LengthFwd = 10.000000000000000
59  f.LengthRev = 0.000000000000000
60  f.Solid = True
61  f.Reversed = False
62  f.Symmetric = False
63  f.TaperAngle = 0.000000000000000
64  f.TaperAngleRev = 0.000000000000000
65  App.getDocument('Pieza_TFM').recompute()
66
67  link=[]
```

```
for i in internal.keys():
    doc.getObject('Body').newObject('Sketcher::SketchObject',f'internal{i}')
    d = FreeCAD.getDocument('Pieza_TFM').getObject(f'internal{i}')
    for j in range(len(internal[i][0])-1):

        d.addGeometry(line(internal[i][0][j],internal[i][1][j],internal[i][0][j+1],internal[i][1][j+1]), False)
    doc.addObject('Part::Extrusion',f'Extrude{i}')
    f = doc.getObject(f'Extrude{i}')
    f.Base = doc.getObject(f'internal{i}')
    f.DirMode = "Normal"
    f.DirLink = None
    f.LengthFwd = 10.000000000000000
    f.LengthRev = 0.000000000000000
    f.Solid = True
    f.Reversed = False
    f.Symmetric = False
    f.TaperAngle = 0.000000000000000
    f.TaperAngleRev = 0.000000000000000
    link.append(doc.getObject(f'Extrude{i}'))

doc.addObject("Part::Compound","Compound")
doc.Compound.Links =link
doc.addObject("Part::Cut","Cut")
doc.Cut.Base = App.activeDocument().Extrude
doc.Cut.Tool = App.activeDocument().Compound

doc.recompute()
```

Bibliografía

[1] T. Pavlidis, *Algorithms for graphics and image processing*. Rockville, MD: Computer Science Pr, 1984, ISBN: 091489465X.

[2] K. Svanberg, «The method of moving asymptotes-a new method for structural optimization,» *International journal for numerical methods in engineering*, vol. 24, n.º 2, págs. 359-373, feb. de 1987. DOI: 10.1002/nme.1620240207. dirección: https://api.istex.fr/ark:/67375/WNG-7DDM2SFJ-Q/fulltext.pdf.

[3] M. P. Bendsøe y N. Kikuchi, «Generating optimal topologies in structural design using a homogenization method,» *Computer Methods in Applied Mechanics and Engineering*, vol. 71, n.º 2, págs. 197-224, nov. de 1988. DOI: 10.1016/0045-7825(88)90086-2. dirección: https://doi.org/10.1016/0045-7825(88)90086-2.

[4] M. P. Bendsøe, «Optimal shape design as a material distribution problem,» *Structural Optimization*, vol. 1, n.º 4, págs. 193-202, dic. de 1989. DOI: 10.1007/bf01650949. dirección: https://doi.org/10.1007/bf01650949.

[5] P. Y. Papalambros y M. Chirehdast, «An Integrated Environment for Structural Configuration Design,» *Journal of engineering design*, vol. 1, n.º 1, págs. 73-96, ene. de 1990. DOI: 10.1080/09544829008901645. dirección: https://www.tandfonline.com/doi/abs/10.1080/09544829008901645.

[6] M. P. Bendsøe, *Optimization of structural topology, shape, and material*. Berlin [u.a.]: Springer, 1995, ISBN: 9783540590576.

[7] P. Hajela y E. Lee, «Genetic algorithms in truss topological optimization,» *International Journal of Solids and Structures*, vol. 32, n.º 22, págs. 3341-3357, 1995, ISSN: 0020-7683. DOI: https://doi.org/10.1016/0020-7683(94)00306-H. dirección: https://www.sciencedirect.com/science/article/pii/002076839400306H.

[8] K. Maute y E. Ramm, «Adaptive topology optimization,» *Structural optimization*, vol. 10, n.º 2, págs. 100-112, oct. de 1995, ISSN: 1615-1488. DOI: 10.1007/BF01743537. dirección: https://doi.org/10.1007/BF01743537.

[9] O. Sigmund, «On the Design of Compliant Mechanisms Using Topology Optimization,» *Mechanics of Structures and Machines*, vol. 25, n.º 4, págs. 493-524, 1997. DOI: 10.1080/08905459708945415. eprint: https://doi.org/10.1080/08905459708945415. dirección: https://doi.org/10.1080/08905459708945415.

[10] M. Beckers, «Topology optimization using a dual method with discrete variables,» *Structural Optimization*, vol. 17, n.º 1, págs. 14-24, feb. de 1999. DOI: 10.1007/bf01197709. dirección: https://doi.org/10.1007/bf01197709.

[11] M. P. Bendsøe y O. Sigmund, «Material interpolation schemes in topology optimization,» *Archive of Applied Mechanics (Ingenieur Archiv)*, vol. 69, n.º 9-10, págs. 635-654, nov. de 1999. DOI: 10.1007/s004190050248. dirección: https://doi.org/10.1007/s004190050248.

[12] T. E. Bruns y D. A. Tortorelli, «Topology optimization of non-linear elastic structures and compliant mechanisms,» *Computer methods in applied mechanics and engineering*, vol. 190, n.º 26, págs. 3443-3459, 2001. DOI: 10.1016/S0045-7825(00)00278-4. dirección: https://dx.doi.org/10.1016/S0045-7825(00)00278-4.

[13] O. Sigmund, «A 99 line topology optimization code written in Matlab,» *Structural and multidisciplinary optimization*, vol. 21, n.º 2, págs. 120-127, abr. de 2001. DOI: 10.1007/s001580050176. dirección: https://search.proquest.com/docview/2262623662.

[14] P.-S. Tang y K.-H. Chang, «Integration of topology and shape optimization for design of structural components,» *Structural and multidisciplinary optimization*, vol. 22, n.º 1, págs. 65-82, ago. de 2001. DOI: 10.1007/PL00013282. dirección: https://search.proquest.com/docview/2262628062.

[15] M. P. Bendsøe y O. Sigmund, *Topology Optimization*. Springer Berlin Heidelberg, 2004. DOI: 10.1007/978-3-662-05086-6. dirección: https://doi.org/10.1007/978-3-662-05086-6.

[16] J. K. Guest, J. H. Prévost y T. Belytschko, «Achieving minimum length scale in topology optimization using nodal design variables and projection functions,» *International Journal for Numerical Methods in Engineering*, vol. 61, n.º 2, págs. 238-254, ago. de 2004. DOI: 10.1002/nme.1064. dirección: https://doi.org/10.1002/nme.1064.

[17] O. Sigmund, «Morphology-based black and white filters for topology optimization,» *Structural and Multidisciplinary Optimization*, vol. 33, n.º 4-5, págs. 401-424, ene. de 2007. DOI: 10.1007/s00158-006-0087-x. dirección: https://doi.org/10.1007/s00158-006-0087-x.

[18] F. Wang, B. S. Lazarov y O. Sigmund, «On projection methods, convergence and robust formulations in topology optimization,» *Structural and Multidisciplinary Optimization*, vol. 43, n.º 6, págs. 767-784, dic. de 2010. DOI: 10.1007/s00158-010-0602-y. dirección: https://doi.org/10.1007/s00158-010-0602-y.

[19] J. Daaboul, C. D. Cunha, A. Bernard y F. Laroche, «Design for mass customization: Product variety vs. process variety,» *CIRP annals*, vol. 60, n.º 1, págs. 169-174, 2011. DOI: 10.1016/j.cirp.2011.03.093. dirección: https://dx.doi.org/10.1016/j.cirp.2011.03.093.

[20] B. S. Lazarov y O. Sigmund, «Filters in topology optimization based on Helmholtz-type differential equations,» *International journal for numerical methods in engineering*, vol. 86, n.º 6, págs. 765-781, mayo de 2011. DOI: 10.1002/nme.3072. dirección: https://api.istex.fr/ark:/67375/WNG-C4D5Q9BL-W/fulltext.pdf.

[21] P. D. Dunning y H. A. Kim, «Introducing the sequential linear programming level-set method for topology optimization,» *Structural and multidisciplinary optimization*, vol. 51, n.º 3, págs. 631-643, sep. de 2014. DOI: 10.1007/s00158-014-1174-z. dirección: https://link.springer.com/article/10.1007/s00158-014-1174-z.

[22] P. Jiang, J. Leng, K. Ding, P. Gu e Y. Koren, «Social manufacturing as a sustainable paradigm for mass individualization,» *Proceedings of the Institution of Mechanical Engineers. Part B, Journal of engineering manufacture*, vol. 230, n.º 10, págs. 1961-1968, oct. de 2016. DOI: 10.1177/0954405416666903. dirección: https://journals.sagepub.com/doi/full/10.1177/0954405416666903.

[23] J. Wu, N. Aage, R. Westermann y O. Sigmund, «Infill optimization for additive manufacturing—approaching bone-like porous structures,» *IEEE transactions on visualization and computer graphics*, vol. 24, n.º 2, págs. 1127-1140, 2017.

[24] J. Wu, A. Clausen y O. Sigmund, «Minimum compliance topology optimization of shell–infill composites for additive manufacturing,» *Computer methods in applied mechanics and engineering*, vol. 326, págs. 358-375, nov. de 2017. DOI: 10.1016/j.cma.2017.08.018. dirección: https://dx.doi.org/10.1016/j.cma.2017.08.018.

[25] P. Bianchi, *4.0: La nueva revolución industrial*, 10 septiembre 2020. Alianza Editorial, 2018, ISBN: 9788491819561.

[26] J. Wu, N. Aage, R. Westermann y O. Sigmund, «Infill Optimization for Additive Manufacturing—Approaching Bone-Like Porous Structures,» *IEEE transactions on visualization and computer graphics*, vol. 24, n.º 2, págs. 1127-1140, feb. de 2018. DOI: 10.1109/TVCG.2017.2655523. dirección: https://ieeexplore.ieee.org/document/7829422.

[27] J. M. Kinser, *Image operators*, 1.ª ed. Boca Raton, FL ; London: CRC Press an imprint of the Taylor & Francis Group, 2019, ISBN: 9781498796187. DOI: 10.1201/9780429451188.

[28] Y. Xia, M. Langelaar y M. A. N. Hendriks, «Automated optimization-based generation and quantitative evaluation of Strut-and-Tie models,» *Computers & structures*, vol. 238, pág. 106 297, oct. de 2020. DOI: 10.1016/j.compstruc.2020.106297. dirección: https://dx.doi.org/10.1016/j.compstruc.2020.106297.

[29] *OpenSCAD documentation*, 2022. dirección: https://openscad.org/index.html.

[30] *Additive Manufacturing Technologies poster*. dirección: https://www.hubs.com/get/am-technologies/.

[31] *CadQuery 2 documentation*. dirección: https://cadquery.readthedocs.io/en/latest/index.html.

[32] K. K. Choi y N. H. Kim, *Structural Sensitivity Analysis and Optimization 1 : Linear Systems*. New York, NY: Springer New York, ISBN: 038723232X. DOI: 10.1007/b138709. dirección: https://library.biblioboard.com/viewer/badc943f-bfb0-11ea-87ad-0a28bb48d135.

[33] *Display tray for infill pattern and infill density*. dirección: https://pinshape.com/items/25524-3d-printed-display-tray-for-infill-pattern-and-infill-density.