Generador de mallas 3D a partir de dibujos de los perfiles de un objeto

3D mesh generator from drawings of object profiles

Trabajo de Fin de Grado
Curso 2024–2025

Autor
Alejandro Segarra Chacón

Director
Rubén Rafael Rubio Cuéllar

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Generador de mallas 3D a partir de dibujos de los perfiles de un objeto

3D mesh generator from drawings of object profiles

Trabajo de Fin de Grado en Desarrollo de Videojuegos

Autor
Alejandro Segarra Chacón

Director
Rubén Rafael Rubio Cuéllar

Convocatoria: *Septiembre 2025*
Calificación: *6,6*

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

22 de septiembre de 2025

DEDICATORIA

A mi madre, por estar siempre en cada paso y hacer posible lo que parecía tan lejano.

AGRADECIMIENTOS

A Rubén, por su comprensión y dedicación guiándome en todo momento, y por preguntar lo justo para que encontrara las respuestas.

A todos los profesores y compañeros de la facultad, con los que he compartido todos estos años.

Y a Cristina, por su apoyo cuando más lo he necesitado.

RESUMEN

Generador de mallas 3D a partir de dibujos en papel o dispositivo electrónico de los perfiles de un objeto

La reconstrucción tridimensional a partir de tres vistas ortogonales sigue siendo un proceso costoso y propenso a errores. Este trabajo presenta una herramienta que reconstruye una malla 3D cerrada y exportable a partir de esas vistas, automatizando este procedimiento que de otro modo sería manual y complejo.

El programa toma como entrada tres imágenes que representan las vistas principales en el sistema diédrico de un objeto: alzado, planta y perfil. A partir de ellas, extrae la geometría 2D, establece correspondencias entre vistas para reconstruir la estructura 3D y verifica la consistencia geométrica y topológica antes de generar la malla.

El resultado es una malla 3D completa que puede ser visualizada y exportada en un formato estándar, lista para tareas posteriores como texturización, animación o edición detallada, así como para su uso en una amplia gama de campos como ingeniería y diseño industrial, arquitectura y construcción, y desarrollo de videojuegos y realidad aumentada.

Palabras clave

Reconstrucción 3D, Modelado 3D, Proyecciones ortogonales, Procesamiento de imágenes, Geometría computacional, Teoría de grafos.

ABSTRACT

3D mesh generator from graph paper or electronic device drawings of object profiles

Three-dimensional reconstruction from three orthogonal views remains a costly and error-prone process. This paper presents a tool that reconstructs a closed and exportable 3D mesh from these views, automating a procedure that would otherwise be manual and complex.

The program takes as input three images that represent the main views of an object in the dihedral system: front view, top view, and side view. From them, it extracts the 2D geometry, establishes correspondences between views to reconstruct the 3D structure, and verifies geometric and topological consistency before generating the mesh.

The result is a complete 3D mesh that can be visualized and exported in a standard format, ready for subsequent tasks such as texturing, animation or detailed editing, as well as for use in a wide range of fields such as Engineering and Industrial Design, Architecture and Construction, and Video Game Development and Augmented Reality.

Keywords

3D Reconstruction, 3D Modeling, Orthographic Projections, Image Processing, Computational Geometry, Graph Theory.

ÍNDICE

1. Introducción **1**
1.1. Motivación . . . 2
1.2. Objetivos y alcance . . . 3
1.2.1. Objetivos específicos . . . 4
1.2.2. Alcance y supuestos . . . 4
1.3. Plan de trabajo . . . 6
1.4. Estructura de la memoria . . . 6
1.5. Recursos externos . . . 7

2. Antecedentes y Fundamentos **9**
2.1. Algoritmos y técnicas fundamentales . . . 9
2.2. Digitalización de dibujos técnicos . . . 10
2.3. Herramientas y dependencias . . . 12

3. Metodología e Implementación **15**
3.1. Descripción del método . . . 15
3.2. Detalles de implementación . . . 16
3.2.1. Preprocesado y extracción 2D . . . 17
3.2.2. Corrección y normalización de vistas . . . 19
3.2.3. Reconstrucción de puntos 3D . . . 20
3.2.4. Reconstrucción de aristas 3D . . . 22
3.2.5. Detección de caras 3D . . . 24
3.2.6. Verificaciones de consistencia . . . 25
3.2.7. Visualización y exportación . . . 25
3.3. Arquitectura del proyecto . . . 26

4. Interfaz de Usuario **29**
4.1. Flujo de uso . . . 29
4.1.1. Entrada de datos . . . 30
4.1.2. Ajuste de parámetros . . . 30
4.1.3. Reconstrucción y visualización . . . 31
4.2. Diseño y arquitectura . . . 32

4.2.1. Principios de diseño aplicados 32
4.2.2. Manejo de errores 33

5. Evaluación y Resultados **35**
5.1. Casos de estudio 35
5.1.1. Caso A: Poliedros simples y convexos 36
5.1.2. Caso B: Concavidades 36
5.1.3. Caso C: Aristas ocultas con trazo discontinuo 36
5.1.4. Caso D: Escaneos imperfectos 38
5.1.5. Caso E: Simetrías y repeticiones 38
5.1.6. Caso F: Vista inconsistente 39
5.2. Limitaciones 40
5.3. Rendimiento y estadísticas 41
5.4. Casos de prueba adicionales 42

6. Conclusiones y Trabajo Futuro **43**
6.1. Discusión de resultados 43
6.2. Líneas de mejora y extensiones 44

Introduction **45**

Conclusions and Future Work **53**

Bibliografía **55**

ÍNDICE DE FIGURAS

1.1. Vistas ortogonales de un objeto: (a) alzado, (b) planta, y (c) perfil. . 1
1.2. DAC-1, primer sistema CAD comercial. 2
1.3. Resumen del *pipeline* de reconstrucción. 5

3.1. Ejemplo de camino colineal y arista auxiliar. 23
3.2. Ejemplo de ciclo con y sin cuerdas. 24
3.3. Visualización 2D de alzado, planta y perfil con los puntos y aristas detectados. 26
3.4. Visualización 3D de la malla reconstruida. 26

4.1. Interfaz de la aplicación con la pestaña *Vistas 2D* seleccionada. . . . 29

5.1. Caso A: vistas normalizadas con puntos y aristas detectados (izq.) y malla 3D reconstruida (der.). 36
5.2. Caso B: vistas normalizadas con puntos y aristas detectados (izq.) y malla 3D reconstruida (der.). 37
5.3. Caso C: vistas normalizadas con aristas ocultas en trazo continuo (izq.) y malla 3D reconstruida (der.). 37
5.4. Caso D: vistas normalizadas de dibujos a mano (izq.) y malla 3D reconstruida (der.). 38
5.5. Caso E: vistas normalizadas con puntos y aristas detectados (izq.) y malla 3D reconstruida (der.). 39
5.6. Caso F: vistas normalizadas tras eliminar un segmento en el alzado, marcado en rojo. 40

6.1. Orthographic views of an object: (a) elevation, (b) plan, and (c) section. 45
6.2. DAC-1, the first commercial CAD system. 46
6.3. Summary of the reconstruction *pipeline*. 49

ÍNDICE DE TABLAS

4.1. Palabras clave para asignación automática de vistas. 30

5.1. Configuración de los parámetros por defecto. 35
5.2. Métricas de rendimiento de cada etapa del *pipeline*. 41

CAPÍTULO

I

INTRODUCCIÓN

La reconstrucción tridimensional a partir de proyecciones ortogonales es un problema clásico de la geometría descriptiva y de la visión por computador (Monge, 1799; Sugihara, 1986).

En *Géométrie Descriptive*, Gaspard Monge formaliza la idea de que la forma de un sólido puede expresarse mediante sus proyecciones ortogonales, donde cada vista retiene dos coordenadas:

- Alzado $\to (x, y)$, correspondiente a anchura y altura.
- Planta $\to (x, z)$, correspondiente a anchura y profundidad.
- Perfil $\to (y, z)$, correspondiente a altura y profundidad.

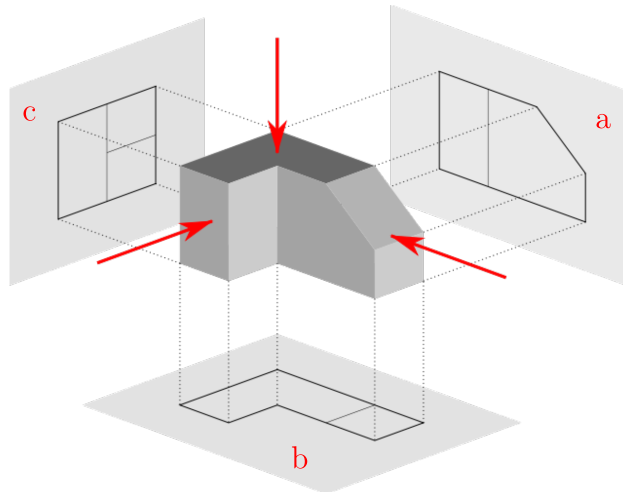


Figura 1.1: Vistas ortogonales de un objeto:
(a) alzado, (b) planta, y (c) perfil.
(IDIS, 2025)

La idea clave es que las tres proyecciones son compatibles entre sí (véase figura 1.1): la intersección coherente de las parejas de coordenadas permite recuperar puntos (x, y, z) del sólido. La dificultad principal no está en calcular las proyecciones,

sino en recuperar la figura 3D resolviendo la *correspondencia* entre vistas: decidir qué punto del *alzado* se empareja con cuál en la *planta* y con cuál en el *perfil*. En visión por computador, el término "correspondencia" se refiere a emparejar características entre imágenes reales; en nuestro caso, se utiliza para el emparejamiento geométrico de vértices y aristas entre las tres vistas.

Por ejemplo, si en el alzado aparece un punto con coordenadas (x=10, y=15), buscamos en planta los puntos cuya x sea $\approx$ 10 y en perfil los puntos cuya y sea $\approx$ 15. Si para alguno de esos candidatos la coordenada z coincide en planta y perfil, recuperamos el punto 3D $(10, 15, z)$. El problema es complejo aun disponiendo de datos exactos: pueden aparecer ambigüedades incluso sin errores de aproximación, por lo que introducimos una leve *tolerancia* en la comparación de coordenadas; esta tolerancia no resuelve la ambigüedad del problema, pero permite absorber ruido o ligeras desalineaciones entre vistas. Esta regla guía el emparejamiento inicial, que más adelante se valida mediante criterios de coherencia (véanse las secciones 3.2.3 y 3.2.6).

En la práctica, muchas ideas de artistas y diseñadores siguen naciendo como bocetos en papel o en digital, y traducir esas vistas a un modelo 3D utilizable continúa siendo un proceso manual lento e ineficiente.

Este proyecto aborda esta necesidad mediante una herramienta que traduce dibujos de tres vistas a una malla 3D cerrada que se puede visualizar y exportar. El objetivo es agilizar la fase inicial del flujo de diseño y modelado 3D: pasar del boceto estructural a una geometría base que sirva como punto de partida para texturizado, animación o edición detallada.

1.1. Motivación

Un hito temprano fue *Sketchpad* (Sutherland, 1963), que introdujo la edición gráfica interactiva haciendo uso de un lápiz óptico que transmitía información sobre su posición en la pantalla. En paralelo, General Motors e IBM presentaron *DAC-1* (véase figura 1.2), uno de los primeros sistemas de CAD interactivos (Computer History Museum, 2011; Krull, 1994), capaz de digitalizar planos 2D y aplicar transformaciones geométricas (rotación, traslación, etc.) sobre ellos. Estos avances marcaron el inicio del desarrollo de herramientas orientadas a transformar la representación gráfica en modelos geométricos manipulables.

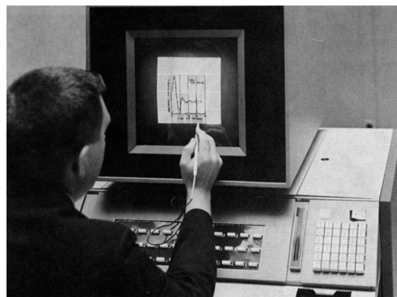


Figura 1.2: DAC-1, primer sistema CAD comercial.
(Computer History Museum, 2011)

Seis décadas después, el hardware y el software son accesibles para cualquiera, pero la necesidad persiste: los sistemas actuales siguen presentando limitaciones a la hora de convertir automáticamente tres vistas ortogonales en un sólido base utilizable, minimizando la intervención del usuario. En concreto: (i) los sistemas CAD comerciales exigen reconstruir el modelo 3D mediante operaciones (extrusión, revolución, etc.) y decisiones de correspondencia que el usuario debe especificar y (ii) los enfoques basados en visión por computador se apoyan en datos densos como nubes de puntos o escáneres, no en dibujos técnicos con información limitada.

A pesar de los avances tecnológicos en estos campos, la mayoría de soluciones automáticas presuponen entradas y objetivos distintos a los de este proyecto. Por ello, la reconstrucción automática no es trivial; automatizar la reconstrucción desde tres vistas plantea dificultades ampliamente estudiadas y originadas por diversos factores (Sugihara, 1986):

- **Ambigüedad de correspondencia.** Un mismo punto puede tener varios candidatos con coordenadas compatibles en las demás vistas, especialmente en figuras con simetría o repeticiones.

- **Proyecciones degeneradas.** Cuando aristas o caras son paralelas a la dirección o al plano de proyección, se producen colapsos a puntos o segmentos. Esto genera solapamientos y coincidencias que no siempre representan relaciones reales.

- **Consistencia topológica.** Las superficies no cerradas, caras que se autointersectan o vértices con conexiones insuficientes pueden generar sólidos no válidos.

- **Visibilidad limitada.** Aristas ocultas, discontinuas o fragmentadas dificultan su correcta detección.

- **Ruido y distorsiones.** Factores como el grosor del trazo, ligeras rotaciones del escaneo o variaciones de escala alteran la correcta detección de vértices y aristas continuas.

- **Unicidad de la solución.** Distintos sólidos pueden compartir las mismas proyecciones ortogonales. Sin hipótesis adicionales o criterios de selección adecuados, el problema admite múltiples soluciones plausibles.

Este proyecto se enmarca en dicha problemática y plantea un enfoque distinto al de los sistemas CAD tradicionales: automatizar la transición de tres vistas coherentes a una malla inicial. De este modo, se busca minimizar las tareas mecánicas asociadas a la construcción manual del modelo, acelerando procesos de diseño o prototipado y priorizando la parte creativa del diseño.

1.2. Objetivos y alcance

El objetivo general es desarrollar una herramienta que, a partir de tres vistas ortogonales en 2D de un objeto, dibujadas sobre papel o un dispositivo electrónico,

genere automáticamente una malla 3D cerrada (no existen huecos en la superficie y cada arista pertenece exactamente a dos caras, con las normales orientadas coherentemente hacia el exterior) y exportable a un formato estándar.

1.2.1. Objetivos específicos

A continuación se enumeran los objetivos específicos del proyecto y, para cada uno, se indican las condiciones que permitirán verificar su cumplimiento (*criterios de aceptación*):

(a) Definir un **modelo de datos** y un *pipeline* robusto para la reconstrucción desde tres proyecciones. Los criterios de aceptación son: (i) una especificación clara de las estructuras del modelo (p. ej., aristas como pares ordenados de vértices, caras como listas ordenada de vértices, etc.), (ii) un diagrama del flujo completo (desde la entrada hasta la exportación), y (iii) una descripción de la complejidad algorítmica de cada etapa.

(b) Implementar un **prototipo funcional** en Python con interfaz gráfica para cargar vistas, ajustar tolerancias y exportar el resultado en formato OBJ. Se considerará cumplido cuando se permita: (i) cargar las tres vistas ortogonales, (ii) ajustar parámetros relevantes para la reconstrucción, (iii) visualizar el resultado en 2D / 3D y (iv) exportar un archivo OBJ compatible con herramientas de modelado como Blender o MeshLab.

(c) Elaborar un conjunto de **casos de prueba** representativo (primitivas, concavidades, simetrías). Se considerará cumplido cuando exista un banco de al menos 10–15 casos con sus resultados esperados y una documentación de cada caso.

(d) **Evaluar** precisión y robustez y **medir** tiempos de ejecución por etapa. El criterio de aceptación es disponer de la tasa total de reconstrucciones válidas y de los tiempos de ejecución por fase para cada ejemplo de prueba, indicando las etapas más costosas y su causa.

1.2.2. Alcance y supuestos

Para acotar el problema y garantizar resultados reproducibles, se asumen:

(I) Vistas ortogonales coherentes, sin deformación por perspectiva y con ejes alineados entre vistas (o rotaciones ligeras corregibles).

(II) Resolución suficiente para detectar contornos y vértices de forma fiable.

(III) Geometría poligonal con caras planas, siendo los contornos de entrada segmentos rectos. No se soportan curvas y, de aparecer, se aproximarán linealmente y no se garantiza su reconstrucción geométrica exacta.

Estos supuestos permiten centrar el esfuerzo en el problema de correspondencia y en la construcción de una malla cerrada que respete aristas rectas y caras planas.

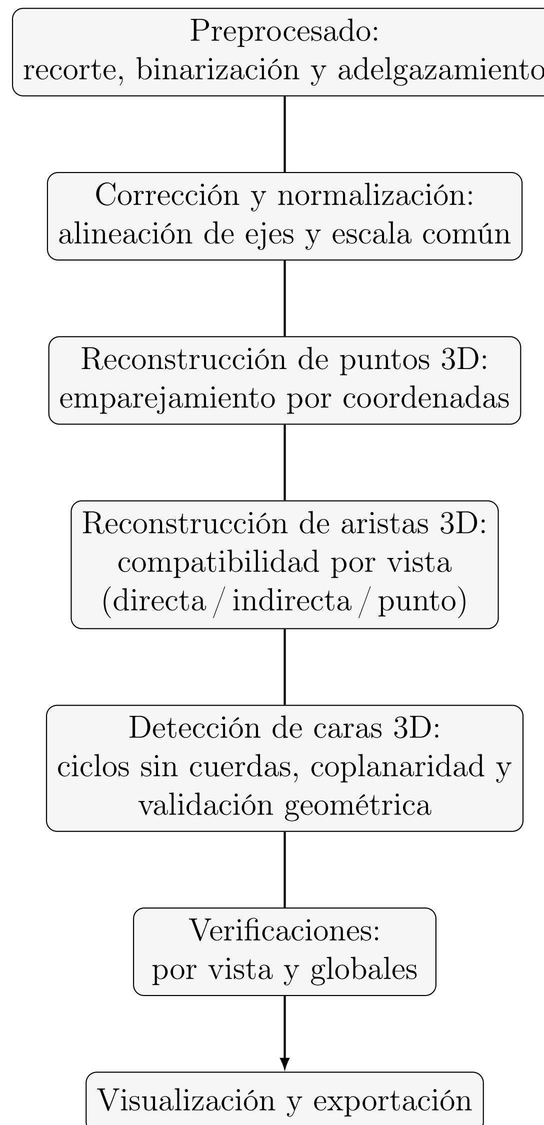


Figura 1.3: Resumen del *pipeline* de reconstrucción.

1.3. Plan de trabajo

El proyecto se organiza en una secuencia de tareas:

1. **Análisis y diseño.** Definición del problema, requisitos y supuestos de entrada.

2. **Conjunto de pruebas.** Preparación de un banco de ejemplos (primitivas, concavidades y combinaciones) que permita validar de forma progresiva cada fase del pipeline e identificar fallos en los supuestos.

3. **Preprocesado y extracción 2D.** Conversión a escala de grises y binarización para separar trazo y fondo; adelgazamiento (*thinning*) del trazo para obtener un esqueleto de 1 píxel; detección de contornos y aproximación por polilíneas (cadena de segmentos rectos conectados). Los vértices y segmentos resultantes constituyen el grafo 2D (nodos y aristas) sobre el que se realizará la reconstrucción.

4. **Normalización y alineación.** Corrección de ortogonalidad de las aristas y unificación de origen y escala.

5. **Reconstrucción 3D.** Emparejamiento de coordenadas entre vistas y verificación de conectividad y consistencia global del grafo 3D generado.

6. **Detección de caras.** Búsqueda de ciclos mínimos en las proyecciones y comprobación de conectividad, coplanaridad y orientación de caras.

7. **Visualización y exportación.** Previsualización y exportación del modelo 3D final.

8. **Evaluación y documentación.** Medición de tiempos, análisis de casos límite y redacción final de la memoria.

La figura 1.3 muestra la secuencia de operaciones que se aplica a las vistas para transformarlas en una malla 3D. Esta secuencia se detalla en el capítulo 3.

1.4. Estructura de la memoria

El documento se organiza de la siguiente forma:

1. **Introducción.** Contexto, objetivos del proyecto y plan de trabajo.

2. **Antecedentes y Fundamentos.** Problema de correspondencia en reconstrucción 3D a partir de proyecciones y bibliotecas y algoritmos utilizados. Se incluyen técnicas exploradas que finalmente no han sido adoptadas y sus utilidades y limitaciones frente al enfoque de este proyecto para justificar decisiones.

3. **Metodología e Implementación.** Descripción detallada del flujo de trabajo: preprocesado, normalización, reconstrucción de puntos y aristas, generación de caras, verificación de consistencia, visualización y exportación. Se incluyen fragmentos de código relevantes.
4. **Interfaz de Usuario.** Diseño de la interfaz y arquitectura.
5. **Evaluación y Resultados.** Casos de prueba, métricas de ejecución por fase, análisis de fallos y discusión crítica.
6. **Conclusiones y Trabajo Futuro.** Lecciones aprendidas, limitaciones y líneas de mejora y extensión.

1.5. Recursos externos

El código del proyecto se encuentra disponible en el repositorio de GitHub: `https://github.com/asegar01/visione`. La estructura del repositorio es la siguiente:

```
.
├── docs/ (memoria)
├── examples/ (ejemplos de prueba)
├── src/ (código)
├── README.md (instrucciones de uso)
└── requirements.txt
```

El conjunto de **datos y ejemplos** utilizados durante el desarrollo y en la evaluación (véase capítulo 5) se incluye en el directorio `examples` del repositorio, mientras que la **guía de uso y dependencias** necesarias se detallan en los ficheros `README.md` y `requirements.txt`, respectivamente.

CAPÍTULO

II

ANTECEDENTES Y FUNDAMENTOS

Este capítulo sitúa el proyecto en su contexto histórico y técnico, desde la geometría descriptiva de finales del siglo XVIII hasta técnicas y herramientas modernas de visión por computador. El objetivo es documentar los trabajos previos y las técnicas que se utilizan de forma habitual en esta línea, sin entrar en la implementación concreta de este proyecto (detallada en el capítulo 3).

2.1. Algoritmos y técnicas fundamentales

La idea de describir un sólido mediante tres proyecciones ortogonales tiene su origen en la *Géométrie Descriptive* de Gaspard Monge, que formaliza cómo el alzado, la planta y el perfil describen, por pares de coordenadas, la forma en el espacio (Monge, 1799).

La reconstrucción 3D abarca una variedad de técnicas que presuponen diversas entradas. Para contextualizar, conviene situar el problema de las tres vistas frente a otras líneas conocidas:

- **Nubes de puntos densas y superficies continuas.** Métodos que requieren datos densos de la superficie como *Poisson Surface Reconstruction* (Kazhdan et al., 2006), *Ball-pivoting* (Bernardini et al., 1999) o los *alpha-shapes* (Edelsbrunner et al., 1983), basados en la triangulación de Delaunay (Delaunay, 1934); producen mallas cerradas y suaves, pero asumen grandes muestras de datos de la superficie. Estas nubes de puntos se pueden obtener mediante escáneres 3D o fotogrametría que triangula puntos a partir de múltiples imágenes calibradas del objeto.
- **Reconstrucción a partir de múltiples imágenes.** La entrada son imágenes de un mismo objeto, capturadas desde distintos puntos de vista y con textura o variación de intensidad suficiente para poder establecer correspondencias (detectar y emparejar puntos) entre ellas. A partir de estas correspondencias, el método estima la posición y la orientación desde donde se capturó cada imagen y genera una nube de puntos densa, que puede convertirse en una malla tridimensional con los métodos mencionados anteriormente. Dentro de este

marco existe la variante *shape-from-silhouette*, que estima la forma mediante la intersección de las proyecciones de siluetas tomadas desde diferentes puntos de vista, generando una *visual hull* que sobreaproxima el objeto e ignora concavidades (Laurentini, 1994).

- **Modelado a partir de un boceto (*sketch-based modeling*).** Sistemas como *Teddy* (Igarashi et al., 1999) interpretan contornos dibujados a mano para inflar superficies o generar volúmenes libres en tiempo real. Resultan eficaces para modelado rápido o conceptual, pero no abordan el caso de tres vistas ortogonales típicas del dibujo técnico.

- **Aprendizaje profundo.** En la última década se han logrado avances significativos en reconstrucción 3D mediante modelos de redes neuronales profundas. Modelos como Pix2Vox (Xie et al., 2019) o 3D-R2N2 (Choy et al., 2016) generan representaciones volumétricas o mallas 3D aproximadas a partir de una o varias imágenes de entrada.

Todos estos casos comparten un supuesto que no se cumple en los dibujos técnicos: la entrada consiste en un muestreo denso de la superficie del objeto. Estos métodos están diseñados para datos densos, provenientes de escáner o fotogrametría, y suelen suavizar los bordes y esquinas definidos de la superficie. Dado que el problema de las tres vistas ortogonales parte de un conjunto reducido de vértices y aristas definidos a partir de proyecciones, y que es prioritario conservar la geometría exacta indicada en las vistas, estos métodos de reconstrucción no resultan adecuados.

Una referencia influyente es el trabajo de Sugihara sobre interpretación geométrica de dibujos de líneas y reconstrucción de poliedros a partir de proyecciones (Sugihara, 1986). Sugihara establece condiciones para que un dibujo de líneas se pueda interpretar como la proyección ortogonal de un poliedro y propone métodos de reconstrucción compatibles con vistas dadas, destacando que la solución puede no ser única y que es necesario verificar la consistencia topológica del resultado. En esta línea, diversos autores abordan la *reconstrucción a partir de tres vistas* como un problema de correspondencia bajo restricciones geométricas y topológicas.

2.2. Digitalización de dibujos técnicos

Para que el problema esté bien planteado, el primer paso en casi todos los sistemas consiste en transformar un dibujo rasterizado (escaneado o fotografiado) en una representación geométrica con vértices y segmentos 2D que formen un grafo plano. Esto se resuelve clásicamente con una cadena de métodos bien estudiados en visión por computador:

Binarización con Otsu. Para separar trazo y fondo en una imagen sin ajustar un umbral a mano, se emplea el método de Otsu: analiza el histograma de la imagen en escala de grises y elige el umbral que maximiza la varianza entre dos clases (fondo claro y trazo oscuro). En esencia, prueba todos los umbrales posibles y elige el que mejor separa trazo y fondo: que los píxeles de cada

grupo queden lo más agrupados posible y lo más alejados de los del otro. En dibujos con fondo oscuro puede invertirse el resultado haciendo uso de un *umbral invertido*, dejando el trazo en blanco y el fondo en negro (Otsu, 1979).

Detector de bordes de Canny. Sirve para marcar con precisión los bordes de las líneas; suaviza la imagen para reducir ruido, calcula el gradiente (magnitud y orientación de los cambios de intensidad), suprime no máximos para adelgazar los contornos y, por último, aplica un doble umbral: uno alto para identificar bordes fuertes y otro bajo que conserva los bordes débiles cuando están conectados a los fuertes (Canny, 1986).

Seguimiento de contornos de Suzuki–Abe. Es una técnica que recorre la imagen binaria detectando transiciones entre fondo y figura, sigue cada contorno hasta cerrarlo y devuelve, para cada componente conexa, una lista ordenada de puntos (el contorno) y su *jerarquía* (Suzuki y Abe, 1985). Se entiende por jerarquía la relación padre-hijo entre los contornos (p. ej., un contorno exterior y sus huecos internos).

Adelgazamiento (*thinning*) del esqueleto. El objetivo es eliminar el grosor del trazo para trabajar con líneas de 1 píxel, preservando su conectividad. Algunos algoritmos iterativos van eliminando píxeles del contorno siempre y cuando no rompan el esqueleto y se detienen cuando ya no se puede adelgazar más (Guo y Hall, 1989; Zhang y Suen, 1984).

Aproximación poligonal con Douglas–Peucker. Se utiliza para simplificar líneas, eliminando puntos; conecta el principio y el final de la línea con una recta y descarta todos los puntos intermedios situados a una distancia menor a un umbral, repitiendo el proceso en cada tramo hasta tener los vértices esenciales que definen la forma (Douglas y Peucker, 1973).

Todas estas técnicas son *métodos clásicos* generalizados que constituyen la "caja negra" de numerosos sistemas de digitalización de dibujos técnicos y reconocimiento de primitivas geométricas.

En dibujos sobre *papel cuadriculado* o con patrones regulares impresos, es posible estimar una orientación y escala detectando líneas rectas paralelas, por ejemplo *líneas de Hough*, una práctica habitual en calibración (Duda y Hart, 1972). La ventaja es contar con un sistema de referencia visible, aunque el inconveniente es que condiciona el soporte al exigir la presencia de una cuadrícula visible y es sensible a sombras, pliegues y deformaciones por perspectiva durante el escaneo. Generalmente, se prefiere optar por métodos basados en la normalización de la propia geometría del dibujo, aplicables tanto en dibujos sobre papel liso como en escaneos imperfectos o trazos generados de manera digital.

Los principales sistemas CAD comerciales ofrecen vías para convertir dibujos 2D en modelos 3D, pero no lo hacen de forma automática a partir de un boceto: el usuario debe decidir las correspondencias, restricciones y operaciones. Por ejemplo, *AutoCAD* dispone de herramientas para convertir un dibujo escaneado en líneas digitales (Autodesk, 2025) y *SolidWorks* permite generar un volumen distribuyendo el

boceto en planos ortogonales (Dassault Systèmes, 2023). Este proyecto aborda precisamente esa automatización a partir de proyecciones digitalizadas y normalizadas (véase capítulo 3).

En conjunto, los antecedentes muestran tres aspectos relevantes para el problema:

(i) existe una base teórica sólida que relaciona vistas (alzado, planta y perfil) con puntos y aristas del sólido (Monge, 1799; Sugihara, 1986),

(ii) la interpretación de dibujos de líneas ha sido estudiada en profundidad y conecta con grafos, ciclos y comprobaciones de consistencia (Huffman, 1971; Clowes, 1971; Waltz, 1975; Diestel, 2017), y

(iii) la digitalización de bocetos y la detección 2D de primitivas se resuelven de manera robusta con algoritmos clásicos de visión por computador que pueden adoptarse como módulos de preprocesado (Otsu, 1979; Canny, 1986; Douglas y Peucker, 1973; Guo y Hall, 1989).

2.3. Herramientas y dependencias

El desarrollo se realiza en Python (3.12), que proporciona un entorno completo con bibliotecas bien integradas y documentadas. Aunque no es el lenguaje más eficiente, su rendimiento resulta suficiente para los tamaños de entrada objetivo del proyecto (véase sección 5.3). Sobre esta base se emplean las siguientes bibliotecas:

NumPy. Es la base para programación de arrays y álgebra en Python; proporciona tipos de datos vectoriales y operaciones eficientes que sustentan el cálculo numérico (Harris et al., 2020; NumPy Developers, 2025).

OpenCV. Es un conjunto de herramientas de visión por computador que incluye operaciones de filtrado, detección de bordes y contornos, y transformaciones geométricas (Bradski, 2000; OpenCV Team, 2025).

Scikit-image. Colección de algoritmos de procesado de imagen (p. ej., adelgazamiento, filtrado, segmentación, etc.) implementados en Python (van der Walt et al., 2014; scikit-image contributors, 2025).

NetworkX. Permite modelar y analizar grafos y redes: estructuras de datos, algoritmos de conectividad y ciclos (Hagberg et al., 2008; NetworkX Developers, 2025).

Matplotlib. Biblioteca de visualización 2D/3D en Python para gráficos y primitivas geométricas como polígonos 3D y colecciones (Hunter, 2007; Matplotlib Development Team, 2025).

Tkinter. *Toolkit* de interfaces gráficas incluido en la biblioteca estándar de Python que proporciona ventanas, eventos y elementos gráficos para aplicaciones de escritorio (Mark Roseman, 2025; Python Software Foundation, 2025).

Estas bibliotecas fundamentan las técnicas descritas y sirven de base para la implementación. El capítulo 3 describe el método concreto adoptado, detallando los módulos de preprocesado, normalización, emparejamiento entre vistas, generación de caras y verificación.

CAPÍTULO

III

METODOLOGÍA E IMPLEMENTACIÓN

"La simplicidad es la máxima sofisticación"
— Leonardo da Vinci

Este capítulo describe el flujo completo de la herramienta desde las imágenes de entrada hasta la malla 3D exportable. A diferencia del capítulo 2, aquí se detalla *cómo* se implementa este proyecto: estructuras de datos, parámetros y decisiones de diseño. Se incluyen fragmentos de código representativos; el código completo se encuentra en el repositorio (véase sección 1.5).

3.1. Descripción del método

El método propuesto en este trabajo se fundamenta en tres ideas principales:

(a) **Correspondencias desde proyecciones ortogonales.** En las vistas se identifican puntos y aristas entre ellos que se pueden emparejar para reconstruir puntos y aristas en 3D (véanse las secciones 3.2.3 y 3.2.4).

(b) **Modelado por grafos.** Cada vista 2D se representa como un *grafo no dirigido* $G = (V, E)$ donde V son vértices del dibujo y E sus segmentos. Sobre este grafo trabajamos con *ciclos simples*; en particular, nos interesan los *ciclos sin cuerdas* (ciclos que no contienen diagonales internas) para proponer caras mínimas que luego se elevan y validan en 3D (véase sección 3.2.5).

    Los ciclos se buscan en cada vista ya que (i) antes de reconstruir el grafo 3D no existe un esqueleto fiable sobre el que detectarlos, (ii) buscarlos en 3D exigiría conocer previamente caras o aristas válidas, lo que introduciría una dependencia circular, y (iii) reduce el espacio de búsqueda al requerir que una cara candidata aparezca como un recorrido cerrado en al menos una proyección 2D.

(c) **Tolerancias geométricas adaptativas.** Para compensar pequeñas imprecisiones de trazo o digitalización, todas las comparaciones de coordenadas y validaciones usan tolerancias proporcionales a la escala del dibujo. Sea

$$D = \text{máx}\{\text{diag}_{\text{alzado}},\ \text{diag}_{\text{planta}},\ \text{diag}_{\text{perfil}}\}$$

la diagonal de la *caja delimitadora* más grande entre vistas. Definimos:

$$\texttt{matching_tolerance} = \alpha D, \qquad \texttt{geometry_tolerance} = \beta D,$$

con los valores de α, β configurables por el usuario en la interfaz (véase sección 4.1.2). La primera rige el emparejamiento de coordenadas entre vistas y la compatibilidad de aristas en proyección (véanse las secciones 3.2.3 y 3.2.4); la segunda verifica validaciones como coplanaridad y geometría de caras (véase sección 3.2.5).

El resultado es un sistema que prioriza la fidelidad geométrica y la claridad estructural frente a enfoques que dependen de datos densos o suponen convexidad global, alineándose con los objetivos definidos. Aplicando estos principios, el flujo de trabajo se compone de los siguientes pasos:

1. Preprocesar cada imagen de vista para extraer vértices y aristas 2D.

2. Corregir ortogonalidad y unificar sistema de referencia (origen, ejes y escala).

3. Reconstruir puntos 3D por correspondencia de coordenadas entre vistas.

4. Reconstruir aristas 3D imponiendo compatibilidad de proyecciones.

5. Detectar caras 3D elevando *ciclos sin cuerdas* 2D de las proyecciones y validando su geometría.

6. Verificar consistencia local y global del sólido.

7. Visualizar la malla y exportar en formato OBJ.

3.2. Detalles de implementación

La clase `Projection` encapsula la geometría 2D de cada vista y almacena sus puntos y aristas 2D, la normal del plano y el nombre de la vista:

```python
class Projection:
    def __init__(self, points, edges, normal, name):
        self.points = points    # np.ndarray [N, 2]
        self.edges = edges      # np.ndarray [M, 2], índices en 'points'
        self.normal = normal    # np.ndarray [1, 3]
        self.name = name        # 'plan' | 'elevation' | 'section'
```

Más adelante (sección 3.2.4) se introduce `auxiliary_edges`, un conjunto de aristas auxiliares que extiende `edges` para facilitar la detección de ciclos y la reconstrucción.

3.2.1. Preprocesado y extracción 2D

Partimos de tres imágenes ortogonales del objeto (alzado, planta y perfil), que pueden proceder de un escaneo del boceto, una fotografía del papel o un dibujo realizado en medio electrónico. La extracción de vértices y aristas se implementa en la función `get_points_and_edges_from_contours` (fichero `main.py`) y, para cada vista, devuelve:

(a) un conjunto de puntos (*vértices*) y

(b) un conjunto de segmentos (*aristas*) que conectan esos puntos.

```
plan_points, plan_edges = get_points_and_edges_from_contours(plan_img)
elev_points, elev_edges = get_points_and_edges_from_contours(elev_img)
sect_points, sect_edges = get_points_and_edges_from_contours(sect_img)
```

Esto se consigue realizando los siguientes pasos para cada vista:

1. **Recorte y binarización.** Primero se detecta el contorno exterior de mayor área para recortar la imagen a su *caja delimitadora* y eliminar márgenes, centrando el trabajo en el área útil. La implementación concreta puede verse en la función `crop_image` (fichero `main.py`), que devuelve también el desplazamiento (`offset_x`, `offset_y`) respecto al origen de la imagen original (esquina superior izquierda). Este desplazamiento se suma a las coordenadas detectadas en la imagen recortada para reubicarlas en el sistema original.

   Binarizar consiste en convertir la imagen en dos clases, fondo y trazo (blanco/negro), a partir de un umbral de intensidad para quedarnos con el trazo. Se emplea el método de *Otsu*, que determina automáticamente el umbral que mejor separa ambas clases analizando el histograma de niveles de gris (Otsu, 1979). Dado que el trazo en nuestras imágenes es más oscuro que el fondo, se aplica la *binarización invertida* para que el trazo quede como primer plano (blanco) y el fondo como negro. Esto evita calibraciones manuales y facilita la detección posterior de contornos y el adelgazamiento.

2. **Adelgazamiento (*thinning*).** Sobre la imagen recortada y binarizada se aplica adelgazamiento con el método `thin` de la biblioteca scikit-image para convertir el trazo en un *esqueleto* de 1 píxel de ancho (Guo y Hall, 1989), preservando la conectividad y facilitando la aproximación poligonal posterior al eliminar duplicidades de vértices debidas al grosor del trazo.

3. **Detección de contornos y aproximación poligonal.** Se detectan contornos sobre el esqueleto binario usando el método `findContours` de OpenCV y, cada contorno, se simplifica mediante el método `approxPolyDP` de OpenCV, que implementa el algoritmo de Douglas-Peucker (Douglas y Peucker, 1973). El parámetro ε controla la máxima distancia permitida entre el contorno original y la polilínea resultante; su valor lo fijamos al 1 % del perímetro por defecto, que ha funcionado de manera robusta en nuestros ejemplos, aunque conviene ajustarlo en algunos escenarios (véase sección 4.1.2). Además, la función

descarta contornos con área inferior a `noise_threshold`[1], ajustable también desde la interfaz.

4. **Fusión de vértices y construcción de aristas.** Para cada contorno aproximado se añaden segmentos entre vértices consecutivos. A continuación, la función `get_unique_point` fusiona vértices cuya distancia euclídea sea menor que `vertex_distance`[2], valor ajustable desde la interfaz (véase sección 4.1.2). Esta fusión es una *heurística*: no garantiza la unicidad perfecta de los vértices; un valor alto de este parámetro reduce duplicados pero puede fusionar vértices distintos, mientras que un valor bajo preserva detalles pero puede dejar vértices duplicados. Finalmente, se corrigen índices y se devuelven los puntos y aristas (pares de enteros) detectados.

En dibujo técnico, las aristas *ocultas* se dibujan a menudo con trazo discontinuo. Para poder tratarlas como segmentos continuos en la extracción, es necesario completar esas líneas discontinuas. Esto se consigue aplicando cierres morfológicos *después* de la binarización y *antes* del adelgazamiento. Para ello, la función `get_points_and_edges_from_contours` ofrece el parámetro `kernel_shape`, que llamaremos k. Cuando $k = 0$ (valor por defecto), no se aplica ningún cierre; si su valor es $k > 0$, se aplican cierres morfológicos horizontales y verticales sobre la imagen binaria.

Un *cierre morfológico* conecta pequeñas discontinuidades aplicando una *dilatación* para cerrar huecos y, a continuación, una *erosión* para recuperar el grosor original, ambos con el mismo elemento estructurante. Se aplican dos cierres: uno horizontal con elemento estructurante rectangular de tamaño $(k, 1)$ que "rellena" huecos alineados horizontales y otro vertical de tamaño $(1, k)$ que realiza lo mismo en huecos alineados verticalmente. Este enfoque prioriza la eficiencia y cubre los casos más comunes en dibujo técnico, con un coste significativamente inferior al de un enfoque generalizado para cualquier inclinación, ya que exigiría aplicar múltiples elementos estructurantes rotados.

```
kernel_h = cv2.getStructuringElement(cv2.MORPH_RECT, (k, 1))
kernel_v = cv2.getStructuringElement(cv2.MORPH_RECT, (1, k))
image_bin = cv2.morphologyEx(image_bin, cv2.MORPH_CLOSE, kernel_h)
image_bin = cv2.morphologyEx(image_bin, cv2.MORPH_CLOSE, kernel_v)
```

El tamaño del cierre se controla mediante el valor de k (ajustable desde la interfaz, véase sección 4.1.2), que se fuerza a *impar* (si se fija un valor par, se incrementa en 1) para que el elemento tenga centro. El uso de valores pequeños corrige discontinuidades leves, mientras que valores altos reconectan huecos mayores pero pueden unir trazos que no deberían conectarse, por lo que se recomienda ajustar de forma gradual.

[1] Umbral expresado en px^2 (área en píxeles) para filtrar contornos pequeños (ruido).
[2] Radio de fusión en px (píxeles) para considerar detecciones cercanas como el mismo vértice.

3.2.2. Corrección y normalización de vistas

La extracción puede introducir pequeñas desviaciones respecto a la ortogonalidad ideal. Para que el cruce de información entre *alzado* (x, y), *planta* (x, z) y *perfil* (y, z) sea fiable, las tres vistas deben compartir:

(a) un **origen** común (comparar coordenadas de forma directa tiene sentido si "cero" significa lo mismo en todas),

(b) una **escala** compatible (una unidad en x vale lo mismo en alzado y planta),

(c) y **ejes** coherentes (misma convención sobre qué coordenadas representa cada vista).

El propósito es minimizar las ambigüedades de correspondencia antes de la reconstrucción y garantizar la comparación directa de coordenadas entre vistas. Si esta coherencia inicial falla, las ambigüedades se propagan a las etapas posteriores y dificultan la reconstrucción correcta del modelo. Estos aspectos se abordan mediante:

(a) **Corrección de ortogonalidad global.** La función `evaluate_rotation` estima (i) el ángulo de rotación principal de la vista y (ii) un grado de *alineación* $A \in [0, 1]$ que mide la concentración direccional (media circular de las direcciones de las aristas, ponderada por su longitud). Si $A >$ `rotation_threshold`, la función `align_view_auto` rota la vista en sentido contrario a ese ángulo y, a continuación, la función `align_view` fuerza la ortogonalidad: clasifica como horizontales o verticales los segmentos cuyo ángulo difiere menos de `angle_tolerance` y aplica un *snap* geométrico: promedia la coordenada común de los vértices que caen en la misma línea horizontal o vertical dentro de `line_tolerance`.

Esto evita ligeras desviaciones o desajustes producidos por imprecisiones en la extracción o en el trazo original. Conviene corregirlas incluso si se encuentran por debajo del umbral de tolerancia ya que (i) reducen ambigüedades en el emparejamiento de puntos entre vistas, (ii) facilitan que una misma arista sea compatible simultáneamente en las tres proyecciones, y (iii) previenen errores acumulativos que pueden afectar la coplanaridad de las caras.

(b) **Alineación de ejes.** En el sistema de coordenadas de imagen usado en esta implementación (OpenCV/NumPy), el origen $(0, 0)$ está en la esquina superior izquierda y el eje vertical crece hacia abajo, por lo que se invierte la y para trabajar en un sistema cartesiano convencional (con y creciente hacia arriba).

En el perfil, por convención el eje horizontal de la imagen representa la profundidad z y el vertical la altura y, es decir, el par de esa vista se interpreta como (z, y). Se permutan entonces las coordenadas para que represente (y, z) y mantener la consistencia con las demás vistas.

(c) **Traslación y reescalado conjuntos.** La función `normalize_and_scale_views` traslada cada vista para que su mínimo esté en $(0, 0)$ y reescala para

que las dimensiones comunes (anchura, altura, profundidad) coincidan entre vistas. Así, un mismo valor de x significa lo mismo en alzado y planta; y lo mismo para y, z en las demás combinaciones.

Este paso minimiza la necesidad de exigir una calibración explícita desde el principio. En la mayoría de casos, las pequeñas discrepancias de escala u origen se corrigen automáticamente y las vistas quedan en un sistema común, haciendo el emparejamiento más robusto.

La unificación es robusta cuando se dispone de tres vistas coherentes porque cada coordenada (x, y, z) aparece en dos proyecciones, lo que permite resolver de forma cruzada la escala, la traslación y la orientación de cada eje. Con solo dos vistas (p. ej., alzado y perfil) quedan grados de libertad: los ejes no compartidos entre ambas (p. ej., x y z) no pueden verificarse cruzadamente, por lo que no se puede determinar un sistema de referencia común sin supuestos adicionales.

En este proceso los siguientes parámetros son relevantes y se pueden ajustar:

- `angle_tolerance` (*grados*): umbral angular para clasificar una arista 2D como horizontal o vertical en la función `align_view`; se usa para agrupar y promediar coordenadas durante la corrección de ortogonalidad.
- `line_tolerance` (*px*): distancia ortogonal máxima respecto a la horizontal o vertical más cercana, usada en la función `align_view` para agrupar puntos en la misma recta (tras la rotación).
- `rotation_threshold` ($[0, 1]$): umbral sobre el grado de alineación A devuelto por la función `evaluate_rotation`, que decide si se aplica la rotación automática.

Con las vistas ya corregidas y normalizadas, se construyen las proyecciones 2D. Aquí, el tercer argumento corresponde a la normal del plano de proyección $(0, 0, 1)$ para el alzado, $(0, 1, 0)$ para la planta y $(1, 0, 0)$ para el perfil:

```
plan = Projection(plan_points, plan_edges, np.array([0, 1, 0]), 'plan')
elev = Projection(elev_points, elev_edges, np.array([0, 0, 1]), 'elev')
sect = Projection(sect_points, sect_edges, np.array([1, 0, 0]), 'sect')
```

3.2.3. Reconstrucción de puntos 3D

El principio clásico de la geometría descriptiva establece que cada punto $P = (x, y, z)$ del sólido debe proyectarse como:

$$\pi_{xy}(P) = (x, y) \text{ en alzado}, \quad \pi_{xz}(P) = (x, z) \text{ en planta}, \quad \pi_{yz}(P) = (y, z) \text{ en perfil}$$

La reconstrucción invierte esta relación: dado un punto (x_e, y_e) en el alzado, se buscan candidatos en la planta que cumplan $x \approx x_e$[3] o que pertenezcan a la recta

[3] Todas las comparaciones entre coordenadas se evalúan con una tolerancia común (`matching_tolerance`).

$x = x_e$ cuando, por degeneración de la proyección (p. ej., una arista paralela a la dirección de proyección), el punto del alzado queda indistinguible a lo largo de esa recta. Análogamente, en el perfil se toman candidatos con $y \approx y_e$ o que pertenezcan a la recta $y = y_e$. Si, para algún par de candidatos, las coordenadas z deducidas desde la planta y el perfil coinciden, se acepta el punto tridimensional (x_e, y_e, z). La función `reconstruct_points_3d` implementa la regla de emparejamiento propuesta:

```
for (x_e, y_e) in elevation.points:
    Cx = [j for j, (x_p, z_p) in enumerate(plan.points)
          if |x_p - x_e| < tol]
    Cy = [k for k, (y_s, z_s) in enumerate(section.points)
          if |y_s - y_e| < tol]
    for j in Cx:
        x_p, z_p = plan.points[j]
        for k in Cy:
            y_s, z_s = section.points[k]
            if |z_plan(j) - z_section(k)| < tol:
                points_3d.add((x_e, y_e, (z_p + z_s)/2))
```

Cuando distintas combinaciones corresponden al mismo punto 3D dentro de un umbral, se unifican los duplicados por *cuantización*: sea τ la tolerancia de emparejamiento (`matching_tolerance`), a cada coordenada (x, y, z) se le asigna una celda

$$key = (\text{round}(x/\tau), \text{round}(y/\tau), \text{round}(z/\tau))$$

y, si dos reconstrucciones "caen" en la misma celda key, se conserva un único representante. Esto garantiza que no queden duplicados a distancia menor que τ.

En figuras simétricas (p. ej., un cubo), un punto del alzado puede corresponder a varios puntos en planta y perfil. La estrategia es aceptar todas las coincidencias posibles y posponer la decisión a etapas posteriores con más información (validación de aristas y caras), donde la conectividad filtra combinaciones incompatibles.

En este proceso los siguientes parámetros son relevantes y se pueden ajustar:

- `matching_tolerance` (*px normalizados*[4]): umbral para considerar como equivalentes las coordenadas entre vistas durante el emparejamiento de puntos y la reconstrucción de aristas.
- `geometry_tolerance` (*px normalizados*): umbral para validaciones geométricas posteriores (coplanaridad de caras, suma de giros, etc.). Actúa como distancia máxima al plano de la cara y margen numérico en comprobaciones angulares.

Las tolerancias dependen del tamaño del dibujo. Se calcula una escala por vista como la diagonal de su *bounding box* mediante la función `get_view_scale`, y se toma la mayor:

[4]Coordenadas tras `normalize_and_scale_views`, comunes a las tres vistas y proporcionales a los píxeles originales.

```
scale = max(scale_elevation, scale_section, scale_plan)
matching_tolerance = scale * (tolerance_1 / 100.0)
geometry_tolerance = scale * (tolerance_2 / 100.0)
```

3.2.4. Reconstrucción de aristas 3D

No toda conexión entre dos puntos reconstruidos corresponde a una arista real. Para decidir qué aristas existen en 3D, se exige *compatibilidad de proyecciones*. Una arista (u, v) es válida si, al proyectarla sobre cada vista, coincide con:

(a) una arista 2D directa existente en esa vista,

(b) una arista 2D indirecta por colinealidad, es decir, conecta los extremos de un *camino de segmentos colineales*, o

(c) un único punto, cuando la arista es paralela al eje de proyección.

Por ejemplo, la arista superior frontal de un prisma rectangular:

- en *alzado* se ve como un segmento horizontal,
- en *planta* también es un segmento (misma x y variación en z),
- en *perfil* puede reducirse a un *punto* si la arista es paralela a la dirección de proyección de esa vista.

Decimos que dos segmentos son *colineales* si, denotando por $\theta \in [0°, 180°]$ el ángulo de sus direcciones, se cumple

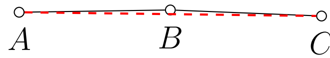
$$\text{mín}(\theta,\ 180° - \theta) \ \leq\ \theta_{\text{col}},$$

donde $\theta_{\text{col}} > 0$ es un umbral angular (en la implementación, `collinear_tolerance`[5]). Idealmente estarían sobre la misma recta, $\theta = 0°$ (mismo sentido) o $\theta = 180°$ (sentido opuesto), pero admitimos un pequeño ángulo para compensar imprecisiones de trazo o digitalización.

Si unimos varios tramos seguidos que están casi en la misma recta, obtenemos un *camino colineal* (véase figura 3.1) y tratamos sus extremos como si estuvieran unidos por un segmento auxiliar. Esta idea permite reconocer líneas largas que el dibujo ha troceado en varios segmentos casi alineados. Estas aristas auxiliares (*aristas indirectas*) se utilizan únicamente para comprobar la conectividad durante la reconstrucción en 3D. Sin embargo, si a partir de ellas se valida una arista 3D real, esta sí se incluye y se exporta en la malla final.

Esto cubre trazos que pueden aparecer fragmentados en el dibujo (p. ej., por discontinuidades del trazo, digitalización o por el propio procesado de la imagen).

[5] Margen angular máximo en grados para considerar que dos segmentos pertenecen al mismo *camino colineal* al construir las *aristas auxiliares* en `calculate_collinear_edges`.



Camino colineal ⇒ añadimos arista indirecta A–C

Figura 3.1: Ejemplo de camino colineal y arista auxiliar.

Lo importante no es la continuidad perfecta del trazo, sino la coherencia geométrica de lo que se representa.

La función `calculate_collinear_edges` recorre vecinos en el grafo de proyección y, manteniendo una dirección base, realiza un recorrido en profundidad (*depth-first search*) que solo avanza por segmentos cuyo ángulo con dicha dirección permanezca dentro de un margen `collinear_tolerance`. Para evitar la acumulación de errores por el encadenamiento de pequeños giros (p. ej., dar la vuelta a una circunferencia trazada con segmentos muy cortos), cada nuevo tramo se compara simultáneamente con (i) la dirección base $\overrightarrow{uv}$ fijada en el primer tramo y (ii) el vector $\overrightarrow{uw}$ desde el origen del camino hasta el vértice actual. Para cada vértice inicial u y cada vértice w alcanzable desde u por un camino colineal, se añade la *arista auxiliar* entre u y w al conjunto `auxiliary_edges`. Este proceso se aplica a cada proyección antes de la reconstrucción de aristas, de modo que las aristas auxiliares estén disponibles durante la comprobación de compatibilidad.

Esta lógica se implementa en la función `reconstruct_edges_3d`, que (i) construye aristas visibles e indirectas por vista, (ii) mapea cada punto 3D a sus índices 2D, (iii) recorre todas las parejas (u, v) y aplica la regla de compatibilidad, y (iv) devuelve el conjunto de aristas 3D:

```
# Conjuntos de aristas visibles e indirectas por vista
plan_v = {tuple(sorted(e)) for e in plan.edges}
plan_i = {tuple(sorted(e)) for e in plan.auxiliary_edges}
# ... análogo para alzado y perfil
# probar todas las parejas (u, v) de puntos 3D
if (plan_edge in plan_v or plan_edge in plan_i or plan_is_point) \
    and (elev_edge in elev_v or elev_edge in elev_i or elev_is_point) \
    and (sect_edge in sect_v or sect_edge in sect_i or sect_is_point):
        edges_3d.add((u, v))
```

Este filtro impide añadir conexiones inexistentes (cuerdas o diagonales internas) que no son compatibles en alguna de las vistas. Por ejemplo, se descarta la existencia de una arista entre vértices no adyacentes (diagonal interna), porque, en alguna vista, su proyección no coincide con ninguna arista 2D (directa o indirecta por colinealidad) ni se proyecta como un punto; solo sería aceptada si, en cada vista, su proyección resulta compatible: o bien un segmento válido (visible o auxiliar por colinealidad) o bien un punto.

3.2.5. Detección de caras 3D

Una cara visible en una vista se percibe como un *recorrido cerrado* de aristas que delimita una región. De forma conceptual, cada vista puede pensarse como un **grafo no dirigido**: los *nodos* o vértices del grafo corresponden a puntos del dibujo y las *aristas* a los segmentos dibujados que los conectan. Este modelo permite identificar recorridos cerrados mínimos, es decir, *ciclos sin cuerdas* (Diestel, 2017).

Un **ciclo sin cuerdas** (en inglés, *chordless cycle*) es un ciclo tal que no hay dos vértices del ciclo conectados por una arista que no pertenezca al ciclo, es decir, no hay diagonales internas (véase figura 3.2).

Ciclo con cuerda $(A, C) \Rightarrow$ *no* chordless

Ciclo sin cuerdas $\Rightarrow$ *chordless*

Figura 3.2: Ejemplo de ciclo con y sin cuerdas.

Si un ciclo 2D contiene una cuerda, su área se descompone en ciclos más pequeños, lo que evita considerar como una única cara una región que, en realidad, está subdividida por diagonales visibles en la vista (p. ej., un cuadrado con una diagonal).

Las caras se derivan de *ciclos sin cuerdas* detectados en cada proyección 2D; la función `find_cycles` obtiene estos ciclos por vista construyendo un grafo a través de la biblioteca `networkx`, con las aristas visibles más las aristas auxiliares por colinealidad. Cada ciclo 2D candidato se eleva a 3D mediante la función `cycles_to_faces` siguiendo un método de búsqueda en profundidad con *backtracking* y poda que evita enumerar el producto cartesiano completo de todas las combinaciones posibles. El procedimiento es:

- **Correspondencia de vértices:** para cada vértice 2D del ciclo, se seleccionan como candidatos, entre los puntos 3D ya reconstruidos, aquellos cuyo par de coordenadas proyectadas en esa vista coincide dentro de `matching_tolerance`.
- **Poda por conectividad:** se recorre el ciclo en orden y se van eligiendo candidatos 3D sin repetir vértices. Cada vez que se añade un nuevo vértice, se comprueba de forma progresiva que la arista 3D con el vértice anterior existe en el conjunto `edges_3d`. Si no existe, se poda esa rama de búsqueda. Al cerrar el ciclo se comprueba también la arista de cierre.
- **Coplanaridad:** la función `is_face_coplanar` exige que todos los puntos de la cara se encuentren en un mismo plano, con distancia al plano inferior a `geometry_tolerance`.

- **Validación geométrica:** la función `is_face_geometry_valid` recorre todos los vértices de la cara en orden. La *suma de los giros* alrededor del contorno debe corresponder a una vuelta completa ($\approx 360°$). Si la suma es cercana a 0° u otro valor, el recorrido se cruza a sí mismo o está mal ordenado, por lo que se descarta.

Las caras duplicadas (mismo conjunto de índices) se descartan y, una vez aceptada la cara, la función `fix_face_orientation` calcula la normal con el orden actual de los vértices y la compara con el vector que une el centroide[6] global del modelo con el centroide de la cara; si el producto escalar es negativo (direcciones opuestas), invierte el orden de los vértices para que la normal apunte hacia el exterior del sólido.

En términos de eficiencia, la implementación actual explora las combinaciones haciendo uso de *backtracking* y aplica podas tempranas por conectividad, lo que reduce notablemente el espacio de búsqueda. Las validaciones de coplanaridad y geometría se aplican únicamente cuando el ciclo está completo, evitando así cómputos innecesarios en ramas descartadas.

3.2.6. Verificaciones de consistencia

Validar puntos, aristas y caras por separado no garantiza por sí solo un objeto cerrado. Un indicador útil es el *grado* de cada vértice, que es el número de aristas que inciden en él. En un cubo, por ejemplo, cada vértice conecta exactamente con tres aristas (grado 3).

Antes de aceptar el modelo final se ejecutan validaciones *locales* y *globales* que evitan reconstrucciones que, aun siendo válidas localmente, no forman un sólido coherente:

- **Conectividad por proyección:** Para calcular el *grado* de cada vértice 2D, se utiliza la función `check_projection_connectivity`. Los vértices de grado 0 o 1 revelan trazos incompletos o errores de extracción. En ese caso, se detiene la reconstrucción y se informa de la vista y el punto conflictivo para su depuración.

- **Consistencia global del modelo:** La función `check_model_consistency` verifica que, para modelos consistentes con más de 4 vértices, cada vértice sea de grado mayor o igual que 3. Esto ayuda a detectar aristas o caras ausentes.

3.2.7. Visualización y exportación

Previsualización 2D. La función `show_views` representa, para cada vista, los puntos y aristas detectados usando la biblioteca Matplotlib (véase figura 3.3). En caso de existir aristas ocultas (indirectas) representadas con trazo discontinuo y haberse aplicado el cierre morfológico, se tratan como segmentos continuos y se dibujan igual que las demás aristas. Es una herramienta de depuración rápida para ajustar umbrales de extracción.

[6] Media aritmética de las coordenadas de los vértices.

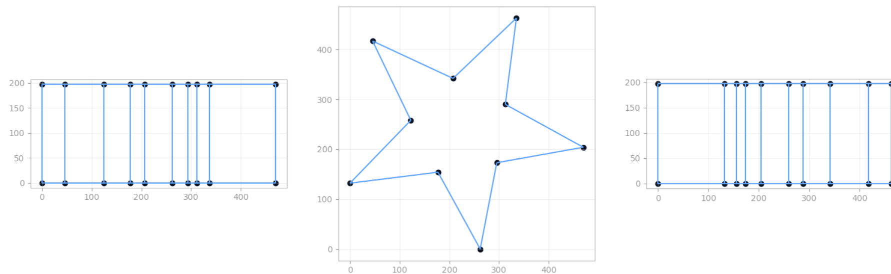


Figura 3.3: Visualización 2D de alzado, planta y perfil con los puntos y aristas detectados.

Previsualización 3D. Para visualizar el modelo 3D reconstruido, se hace uso de la clase `Poly3DCollection` de la biblioteca Matplotlib, una colección de polígonos 3D que permite renderizar caras poligonales sobre un eje tridimensional (véase figura 3.4). Concretamente, la función `plot_3d_mesh` construye la colección a partir de las caras aceptadas y ajusta automáticamente el encuadre, fijando una relación de aspecto $1:1:1$ y una vista isométrica del modelo.

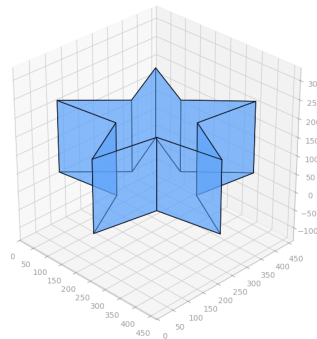


Figura 3.4: Visualización 3D de la malla reconstruida.

Exportación. La función `export_obj` escribe el modelo en formato OBJ, aplicando un reescalado por el máximo valor absoluto de coordenadas para normalizar el rango y generando las listas de vértices y caras correctamente indexadas, preservando la orientación de las caras para asegurar normales salientes. El archivo resultante es compatible con Blender, MeshLab y otras herramientas, y sirve como punto de partida para texturizado, animación o edición posterior del modelo.

3.3. Arquitectura del proyecto

A nivel de código, el proyecto se organiza en los siguientes módulos:

- `main.py`: *pipeline* principal; carga de imágenes, normalización, reconstrucción, visualización y exportación.
- `vistas.py`: operaciones geométricas; alineación, emparejamiento, colinealidad, detección de ciclos, caras y validaciones.

- `graph.py`: implementación de un grafo no dirigido con listas de adyacencia; se utiliza para (i) calcular grados por vértice en las comprobaciones de conectividad y (ii) realizar recorridos entre vecinos para construir el cierre colineal (aristas auxiliares).
- `gui.py`: interfaz para cargar vistas, ajustar parámetros y exportar resultados (descrita en el capítulo 4).

Esta disposición separa el código fuente en el directorio `src` y los datos de ejemplo con sus salidas en el directorio `examples`; la documentación se encuentra en el directorio `docs`, mientras que `README.md` y `requirements.txt` recogen, respectivamente, las instrucciones de uso y las dependencias.

CAPÍTULO
IV

INTERFAZ DE USUARIO

Este capítulo describe la interfaz gráfica que acompaña al sistema y que permite ejecutar el *pipeline* completo (figura 1.3) sin interactuar con código. El diseño prioriza un flujo claro y lineal, retroalimentación inmediata y parámetros que se corresponden con los usados en la implementación (capítulo 3).

4.1. Flujo de uso

La interfaz se compone de dos zonas diferenciadas: un encabezado con botones de carga de ficheros (1), panel de parámetros (2) y barra de acciones (3); y un cuerpo (4), compuesto por pestañas para la visualización 2D, la malla 3D y una consola de mensajes (figura 4.1).

Figura 4.1: Interfaz de la aplicación con la pestaña *Vistas 2D* seleccionada.

Vista	Palabras clave aceptadas
Alzado	`front`, `elevation`, `alzado`
Planta	`top`, `plan`, `planta`
Perfil	`right`, `section`, `perfil`

Tabla 4.1: Palabras clave para asignación automática de vistas.

4.1.1. Entrada de datos

La entrada consiste en tres imágenes (en formato `PNG` o `JPEG`), una por vista ortogonal: alzado, planta y perfil. En la zona izquierda del encabezado hay tres botones: *Cargar alzado*, *Cargar planta* y *Cargar perfil*. Cada botón abre un selector de archivos y muestra la ruta elegida en el campo contiguo. Alternativamente, es posible arrastrar y soltar (*drag* & *drop*) un archivo sobre el campo de cada vista (asigna sólo ese campo) o múltiples archivos en cualquier zona de la ventana.

Para acelerar la carga de imágenes, la interfaz asigna las vistas automáticamente a partir de *palabras clave* presentes en el nombre del archivo, sin distinguir entre mayúsculas y minúsculas (tabla 4.1).

Si se sueltan varios archivos, se sobreescriben las asignaciones previas (si las hubiera), siempre que coincidan las palabras clave. Si algún fichero no existe o no es legible, se refleja mediante un cuadro de diálogo y un mensaje en la pestaña *Consola* informando del error.

La interfaz guarda el último directorio utilizado en un fichero de preferencias (`.visione_prefs.json`) para abrir el selector de archivos en esa ruta la próxima vez.

4.1.2. Ajuste de parámetros

El panel *Parámetros de reconstrucción* agrupa deslizadores que controlan tanto el preprocesado como la validación geométrica:

- *Área mínima de contorno* (px^2). Controla el parámetro `noise_threshold` descrito en la sección 3.2.1 y descarta contornos cuya área sea menor que el valor fijado. Valores bajos preservan detalle pero pueden introducir falsos contornos; valores altos eliminan trazos residuales, pero pueden borrar ligeros detalles. El rango máximo se fija en el 0,2 % del área de la imagen más grande cargada.

- *Distancia para fusionar vértices (px)*. Controla el parámetro `vertex_distance` (sección 3.2.1) y unifica en un único vértice las detecciones cuya distancia sea menor que este valor. Un valor bajo conserva vértices separados pero puede generar vértices duplicados; un valor alto reduce vértices duplicados pero puede fusionar vértices distintos. El rango máximo se fija en el 10 % de la *diagonal* de la vista de mayor tamaño.

- *Tolerancia de emparejamiento (%)*. Ajusta el valor de `matching_tolerance` (sección 3.2.3) y establece el margen para considerar coordenadas como "equi-

valentes” al comparar puntos entre vistas. Si es bajo, exige entradas más limpias y alineadas; si es alto, facilita emparejamientos en dibujos con rotación o escala imperfectas pero aumenta el riesgo de emparejamientos erróneos. *Valores típicos*: 1–5 %.

- *Tolerancia geométrica (%)*. Controla el parámetro `geometry_tolerance` (sección 3.2.5) y fija el umbral usado en validaciones geométricas. Valores altos toleran pequeñas desviaciones numéricas pero pueden aceptar caras degeneradas; valores bajos son más estrictos y pueden rechazar caras válidas si el trazo contiene un ruido notable. Suele ser menor o igual que la de emparejamiento. *Valores típicos*: 1–3 %.

- *Aproximación de contorno (%)*. Controla el parámetro `approx_ratio` (umbral ε de `cv2.approxPolyDP`) descrito en la sección 3.2.1, y define el nivel de simplificación del trazo. Valores bajos preservan la forma original pero pueden introducir vértices redundantes; valores altos simplifican el contorno pero pueden perder detalles. *Valores típicos*: 0.5–2 %. Conviene reducir su valor si se redondean esquinas o desaparecen detalles.

- *Cierre de trazos discontinuos (px)*. Ajusta el valor de `kernel_shape` (sección 3.2.1) y establece el tamaño del cierre que une huecos en trazos discontinuos. Si es 0, desactiva el cierre; valores pequeños reconectan trazos discontinuos; valores grandes pueden unir segmentos que no deberían conectarse.

Los deslizadores permanecen deshabilitados hasta que las *tres* rutas sean válidas. Al cargarlas todas, el sistema calibra automáticamente sus *rangos máximos*. Estas cotas dinámicas evitan parámetros desproporcionados respecto al tamaño del dibujo y mejoran la estabilidad del preprocesado.

4.1.3. Reconstrucción y visualización

Con las tres rutas válidas, el botón *Reconstruir modelo* se habilita. Al pulsarlo:

1. Se lanza la reconstrucción en un hilo de trabajo (*thread*), que se ejecuta en segundo plano dentro del mismo proceso. Esto evita bloquear la interfaz y mantiene la aplicación receptiva.

2. Aparece una barra de progreso indeterminada; los botones de *Reconstruir modelo* y *Exportar* quedan deshabilitados y el botón de *Cancelar* se habilita.

3. Registra en la pestaña *Consola* mensajes de estado y error con formato diferenciado.

Completadas las fases de extracción, alineación y normalización, la pestaña *Vistas 2D* dibuja los puntos y aristas detectados en cada vista (figura 3.3). Tras la reconstrucción de puntos, aristas y caras 3D, la pestaña *Malla 3D* muestra el resultado en una vista isométrica, ajustando encuadre y orientación para facilitar su inspección (figura 3.4). Si la reconstrucción tiene éxito, el botón *Exportar* se habilita

y permite guardar el resultado en formato OBJ. En caso contrario, se informa del motivo.

En cualquier momento del proceso el usuario puede pulsar *Cancelar*. Esto señala una condición de cancelación que la rutina de reconstrucción comprueba en varios puntos de control, tras cada fase relevante. Al cancelar, se detiene el hilo de trabajo, se restablece el estado de la UI y se informa de ello en *Consola*.

4.2. Diseño y arquitectura

El diseño de la interfaz combina principios de usabilidad con una arquitectura ordenada basada en la separación de responsabilidades.

4.2.1. Principios de diseño aplicados

La psicología de la Gestalt surge a inicios del siglo XX y estudia cómo nuestro sistema perceptivo tiende a organizar los estímulos en estructuras significativas de forma rápida y, a menudo, automática (Köhler, 1929; Koffka, 1935). En esencia, se perciben patrones que el cerebro agrupa siguiendo ciertas reglas. Aplicar estos principios en interfaces aporta claridad estructural y reduce la carga cognitiva del usuario. A continuación se detallan los principios utilizados en esta herramienta:

- **Ley de proximidad:** los elementos cercanos entre sí se perciben como pertenecientes al mismo grupo. En la interfaz, los tres campos de carga aparecen físicamente contiguos y alineados en la zona izquierda del encabezado, mientras que los deslizadores de parámetros se agrupan en un bloque independiente en el centro, y la barra de acciones se sitúa agrupada a la derecha. Esta distribución deja claro que se trata de tres conjuntos funcionales diferentes: entrada, ajuste y ejecución. El cuaderno de pestañas (2D, 3D, Consola) refuerza esta proximidad, ya que cada vista se mantiene dentro de su propia pestaña, evitando mezclas visuales innecesarias.

- **Ley de semejanza:** agrupamos elementos que se parecen en forma, color, tamaño o estilo. La interfaz utiliza el mismo estilo visual para todos los *botones de acción* (cargar, reconstruir, exportar y cancelar) y otro consistente para *controles de ajuste*, de modo que el usuario identifica rápidamente el comportamiento de cada uno.

- **Ley de continuidad:** el encabezado está organizado de izquierda a derecha siguiendo el flujo de trabajo: primero *cargar vistas*, luego *ajustar parámetros* y, finalmente, *reconstruir y exportar*. Esta continuidad guía la acción del usuario sin introducir indicaciones explícitas. Esta continuidad se refuerza con la activación progresiva de controles: los deslizadores y el botón de reconstrucción se habilitan solo cuando se han cargado las tres imágenes; así como el botón de exportación tras una reconstrucción correcta.

- **Ley de simetría:** las configuraciones simétricas se perciben como ordenadas y fácilmente interpretables. La zona superior mantiene un equilibrio visual: el

bloque de carga de imágenes a la izquierda y, en espejo, la barra de acciones a la derecha, con el panel de parámetros centrado. Esta simetría reduce fatiga visual y es especialmente útil cuando el usuario alterna entre ajustes y ejecuciones repetidas.

4.2.2. Manejo de errores

La interfaz gestiona los fallos de manera específica, ofreciendo retroalimentación clara para que el usuario pueda corregir la entrada o ajustar parámetros.

(a) **Consola interna.** Lleva el registro de todos los mensajes de la sesión. Distingue entre mensajes *informativos* (no requieren acción del usuario) y *errores*, resaltados en rojo. Los mensajes se añaden en tiempo real y sirven como traza de depuración.

(b) **Cuadros de diálogo.** Cuando es necesaria la intervención del usuario se emplean las funciones del módulo `tkinter.messagebox` (Mark Roseman, 2025):

- *Advertencia* (`messagebox.showwarning`): falta alguna imagen de entrada, conectividad insuficiente en una vista, o ausencia de suficientes puntos o aristas reconstruidos.
- *Error* (`messagebox.showerror`): fallo de lectura de imágenes, problemas inesperados durante la reconstrucción o error al exportar.
- *Información* (`messagebox.showinfo`): no existe ninguna malla disponible para exportar o confirmación de exportación completada correctamente.

CAPÍTULO

V

EVALUACIÓN Y RESULTADOS

Este capítulo presenta una evaluación del prototipo implementado mediante diversos ejemplos. El objetivo no es solo validar su correcto funcionamiento, sino analizar la robustez, precisión y eficiencia bajo distintos escenarios. Para ello, se ha diseñado un banco de pruebas con casos de estudio de complejidad creciente y se han medido los tiempos de ejecución en cada etapa del proceso. Finalmente, se discuten las limitaciones observadas y sus causas.

5.1. Casos de estudio

Se ha seleccionado un conjunto de casos de prueba para estudiar el comportamiento del *pipeline* en diferentes escenarios. Cada caso consta de las tres vistas ortogonales de un objeto: alzado, planta y perfil. Las imágenes de entrada son archivos en blanco y negro en formato `PNG`, con resolución suficiente y trazos rectos. Salvo indicación contraria, los casos se ejecutan con la misma configuración de parámetros, que aparecen en la tabla 5.1.

El banco de ejemplos completo utilizado durante la evaluación está disponible en la carpeta `examples` del repositorio de código.

Parámetro	**Descripción**	**Valor por defecto**
`noise_threshold`	Área mínima de contorno	$100\,px^2$
`vertex_distance`	Distancia para fusionar vértices	15 px
`matching_tolerance`	Tolerancia de emparejamiento	2.0 %
`geometry_tolerance`	Tolerancia geométrica	1.5 %
`approx_ratio`	Aproximación de contorno	1.00 %
`kernel_shape`	Cierre de trazos discontinuos	0 px (desactivado)

Tabla 5.1: Configuración de los parámetros por defecto.

5.1.1. Caso A: Poliedros simples y convexos

El objetivo es validar la funcionalidad básica del *pipeline*: extracción 2D, normalización, reconstrucción de puntos y aristas y detección de caras en escenarios sin ambigüedades complejas. Se utiliza un prisma triangular convexo con una leve rotación espacial que permite visualizar todas las aristas desde las tres vistas. Las vistas 2D normalizadas se disponen de forma habitual en dibujo técnico: el alzado arriba centrado, la planta justo debajo y el perfil derecho a la izquierda del alzado; para cada vista se representan los vértices detectados y las aristas que los unen (véase figura 5.1).

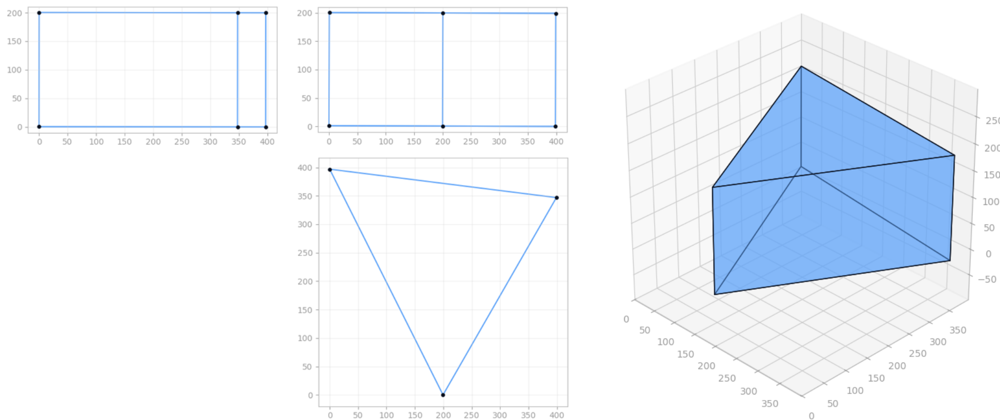


Figura 5.1: Caso A: vistas normalizadas con puntos y aristas detectados (izq.) y malla 3D reconstruida (der.).

El resultado es exitoso y confirma que la lógica central del sistema funciona como se espera en un escenario ideal.

5.1.2. Caso B: Concavidades

Se desea comprobar la capacidad de manejar figuras cóncavas que, a diferencia de las convexas, no se puede deducir la geometría a partir de sus contornos exteriores: requiere identificar correctamente todos los ciclos internos que conforman sus caras.

El resultado cumple con lo esperado y reconstruye una malla coherente y cerrada (véase figura 5.2), lo que sugiere que el enfoque basado en grafos y ciclos mínimos resulta adecuado para modelar las conexiones internas de la figura en este escenario.

5.1.3. Caso C: Aristas ocultas con trazo discontinuo

Para poder interpretar y reconstruir geometrías que incluyen aristas no visibles, trazadas con línea discontinua, es necesario activar el cierre morfológico. Para ello, se establece el cierre de trazos discontinuos en 9 píxeles (tamaño del elemento estructurante del cierre morfológico). Este paso puede introducir pequeñas imprecisiones en las vistas, que se resuelven con las tolerancias de emparejamiento y geometría.

El resultado revela cómo el cierre morfológico “rellena” el trazo discontinuo y permite que la extracción de contornos identifique las aristas ocultas como segmentos

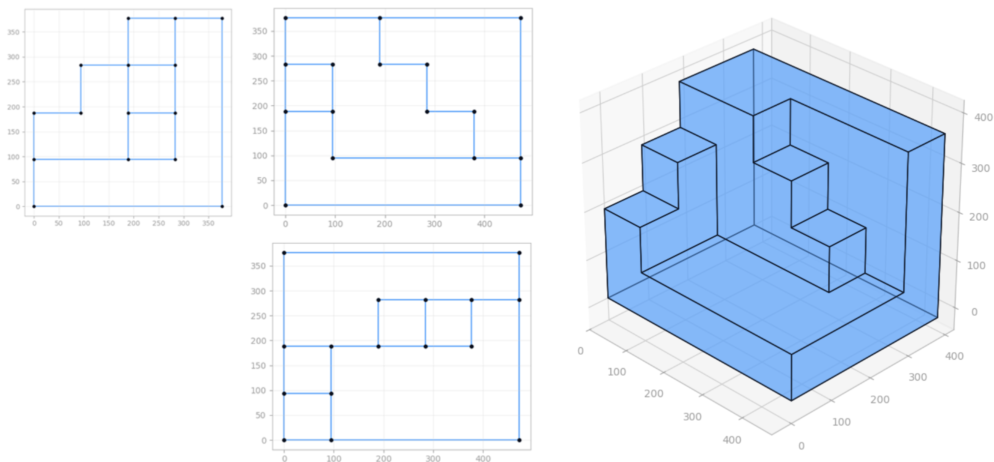


Figura 5.2: Caso B: vistas normalizadas con puntos y aristas detectados (izq.) y malla 3D reconstruida (der.).

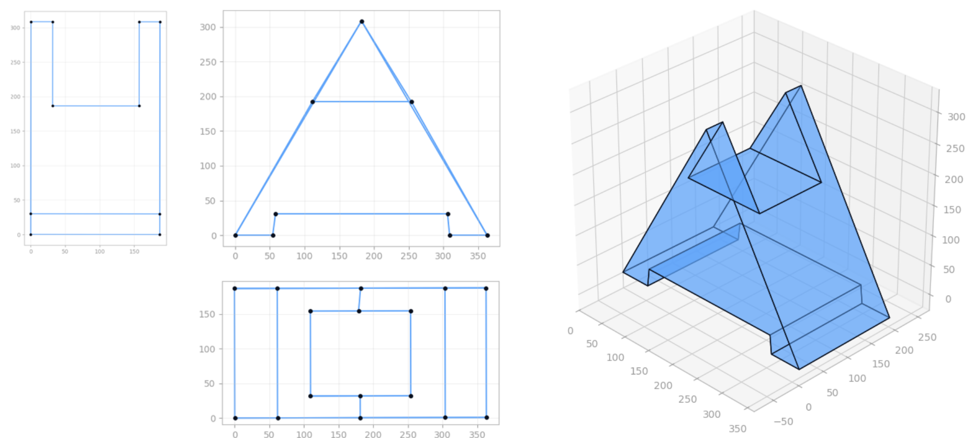


Figura 5.3: Caso C: vistas normalizadas con aristas ocultas en trazo continuo (izq.) y malla 3D reconstruida (der.).

continuos y conectados (véase figura 5.3).

5.1.4. Caso D: Escaneos imperfectos

El objetivo es evaluar la robustez y la capacidad de aplicación frente a entradas reales, donde el trazado a mano alzada introduce ruido y discrepancias de escala entre vistas. Para ello, se utilizan las vistas de un prisma hexagonal dibujadas a mano sobre papel liso y escaneadas, simulando el flujo de trabajo real de un artista.

Las imágenes de entrada presentan desafíos que no presentan los casos rasterizados previos, ya que (i) el grosor del trazo no es uniforme, (ii) las líneas no son perfectamente rectas ni perpendiculares y (iii) el escaneo introduce ruido en las vistas.

Con la configuración propuesta aparecen vértices duplicados y la aproximación poligonal elimina algunas esquinas; se aumenta la distancia de fusión de vértices a 50 píxeles y se reduce la aproximación de contornos al 0.8 % del perímetro. Aun así, persisten algunos vértices falsos, por lo que la alineación automática no logra una ortogonalidad perfecta.

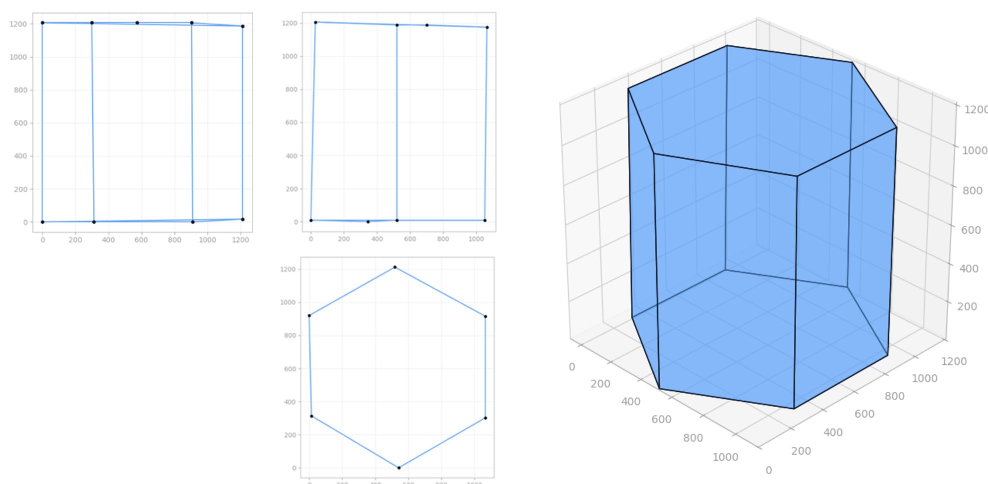


Figura 5.4: Caso D: vistas normalizadas de dibujos a mano (izq.) y malla 3D reconstruida (der.).

A pesar de estas imprecisiones, la reconstrucción se realiza correctamente: los parámetros de tolerancia permiten emparejar puntos de manera estable y validar aristas en las proyecciones, siendo el modelo final coherente con lo esperado, como se observa en la figura 5.4.

En conjunto, este caso muestra que la herramienta no requiere entradas perfectamente calibradas: es posible reconstruir modelos coherentes ajustando umbrales y tolerancias de manera adecuada.

5.1.5. Caso E: Simetrías y repeticiones

El siguiente ejemplo pretende evaluar la resolución de ambigüedades que ocurre cuando múltiples vértices comparten coordenadas en distintas proyecciones, lo que

puede llegar a generar geometría inexistente. Para ello se utiliza una figura compuesta por una base en forma de ortoedro y una extrusión en forma de cruz en la parte superior, lo que provoca que los vértices de la base compartan coordenadas con cada uno de los “brazos” de la cruz.

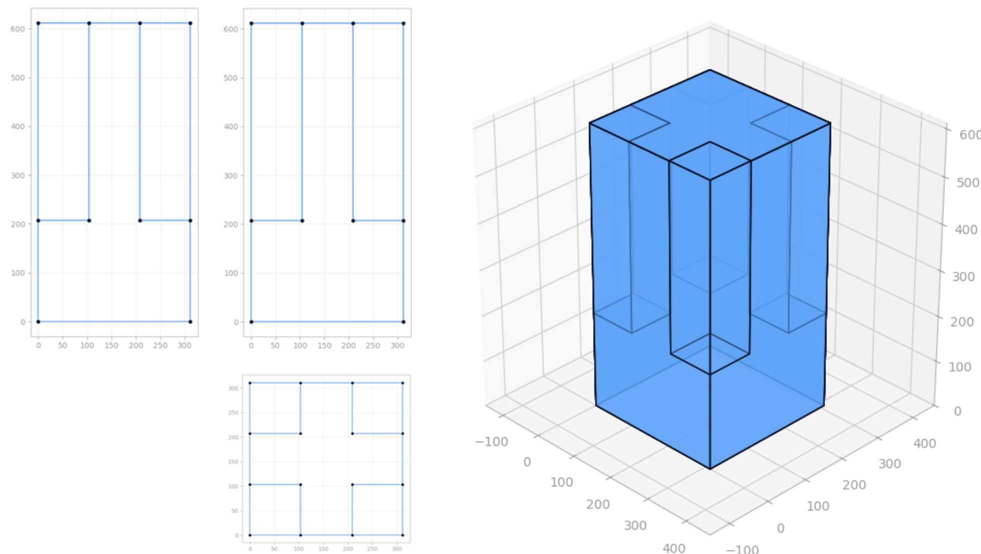


Figura 5.5: Caso E: vistas normalizadas con puntos y aristas detectados (izq.) y malla 3D reconstruida (der.).

Se produce un fallo parcial en la reconstrucción; aunque la forma de la figura es reconocible, se generan cuatro vértices falsos que no existen en el modelo real. Estos vértices aparecen en las esquinas exteriores del volumen que ocuparía la cruz, a la altura de los vértices de la base, como se muestra en la figura 5.5.
Este error muestra una limitación fundamental del método de reconstrucción. Concretamente, ocurre lo siguiente:

1. El programa toma, desde el alzado, las coordenadas (x, y) de un vértice exterior de la cruz.

2. Encuentra una coordenada z compatible en la planta, que realmente pertenece a una esquina de la base cúbica ya que comparten la misma x.

3. Valida esta combinación con la vista de perfil, donde ocurre algo similar.

El resultado es la creación de un punto tridimensional, por ejemplo (0, 600, 0), que, siendo matemáticamente plausible con las proyecciones, es geométricamente incorrecto. Las fases de reconstrucción de aristas y detección de caras no logran descartar estos vértices porque pueden llegar a formar aristas y caras válidas con otros puntos (véase sección 5.2).

5.1.6. Caso F: Vista inconsistente

Con el objetivo de verificar que el sistema rechaza correctamente entradas que incumplen los supuestos, partimos de un caso que funciona adecuadamente (p. ej.,

el caso A) y eliminamos un segmento en la vista de alzado, marcado en color rojo (véase figura 5.6). De esta manera, queda una arista colgante (grado 1) y un contorno abierto en esa proyección.

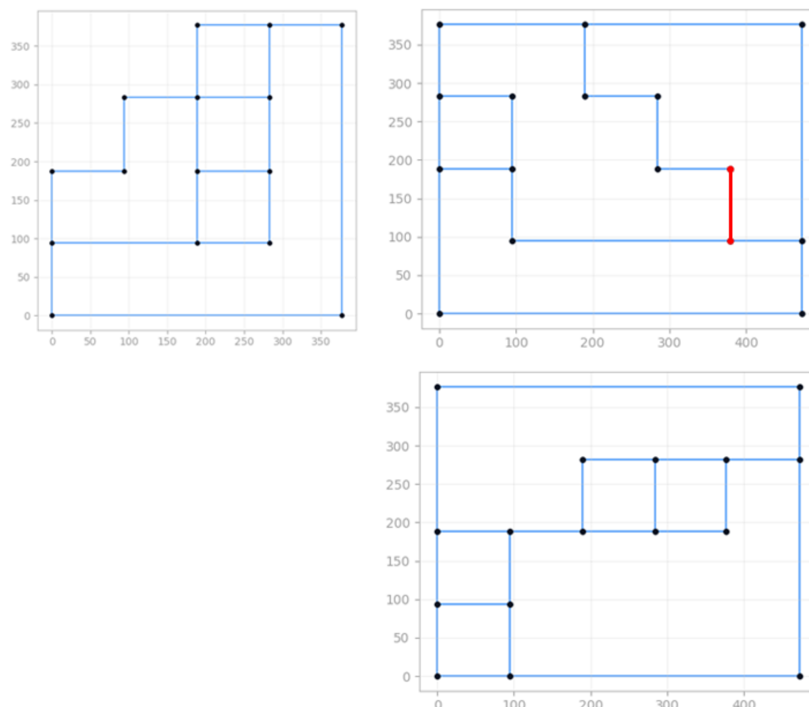


Figura 5.6: Caso F: vistas normalizadas tras eliminar un segmento en el alzado, marcado en rojo.

Como resultado, la reconstrucción se detiene antes de generar la malla. La comprobación de conectividad exige que no existan vértices de grado 1; por tanto, el sistema rechaza la entrada y muestra el siguiente mensaje de error:

```
Conectividad inconsistente. Los puntos deben tener al menos dos conexiones
```

Este comportamiento es correcto, ya que la entrada incumple los supuestos de coherencia y conectividad. En consecuencia, no se genera ninguna malla ni se habilita la exportación.

5.2. Limitaciones

La evaluación ha demostrado la viabilidad del enfoque propuesto, pero también ha revelado ciertas limitaciones inherentes al método y la implementación actual.

En el *caso E* (sección 5.1.5) se observa cómo la validación basada únicamente en la compatibilidad de coordenadas y la conectividad local es insuficiente para resolver todas las ambigüedades que plantea la reconstrucción. Es necesario implementar heurísticas más avanzadas o un análisis topológico global que permita identificar y descartar correctamente estas degeneraciones. Las principales limitaciones identificadas son las siguientes:

- **Ambigüedad geométrica.** La limitación más significativa, como muestra el *caso E* (figura 5.5), es la generación de geometría inexistente (*vértices falsos*). El sistema actual es susceptible a falsos positivos cuando la simetría o la repetición de patrones hacen coincidir características no relacionadas entre distintas vistas; no se dispone de un mecanismo de validación global capaz de identificar vértices plausibles localmente pero incompatibles con la topología del sólido.

- **Entrada y ajuste de parámetros.** Aunque el *caso D* (figura 5.4) sugiere que el método es robusto frente a escaneos imperfectos, la reconstrucción depende de un ajuste preciso de los parámetros. Una imagen con baja resolución o una desacertada elección de los parámetros puede provocar fallos o discrepancias a lo largo del proceso. La herramienta, por tanto, no es completamente automática y requiere intervención explícita del usuario para optimizar los resultados.

- **Conexión y geometría.** El modelo y el flujo de trabajo parten del supuesto de que las vistas representan una única figura conexa. Esto impide la reconstrucción de escenas con múltiples objetos independientes. Además, se asume que el sólido es cerrado, con caras planas y segmentos rectos; no se contemplan curvas ni superficies no planas.

Estas limitaciones definen el alcance del prototipo actual y señalan las áreas de mejora futuras, discutidas en el capítulo 6.

5.3. Rendimiento y estadísticas

En general, el rendimiento del prototipo es adecuado para su uso en flujos de trabajo con bocetos digitales, con tiempos totales de reconstrucción inferiores a *0,1 segundos*, favoreciendo una experiencia fluida.

Etapa del proceso	**A**	**B**	**C**	**D**	**E**
1. Extracción 2D	29 ms	53 ms	45 ms	1488 ms	49 ms
2. Alineación y normalización	1 ms	1 ms	1 ms	1 ms	1 ms
3. Reconstrucción de puntos 3D	0 ms	1 ms	0 ms	0 ms	1 ms
4. Reconstrucción de aristas 3D	3 ms	21 ms	11 ms	6 ms	18 ms
5. Detección de caras 3D	5 ms	12 ms	8 ms	6 ms	19 ms
Tiempo total de ejecución	38 ms	88 ms	65 ms	1501 ms	88 ms

Tabla 5.2: Métricas de rendimiento de cada etapa del *pipeline*.

Como se detalla en la tabla 5.2, el tiempo total de reconstrucción para un modelo de complejidad moderada como son los casos *A*, *B*, *C* y *E* es mínimo; en el caso *D* (sección 5.1.4), sin embargo, se observa un incremento alto, con un tiempo de ejecución superior a *1,5 segundos*. El análisis de rendimiento muestra que las fases más exigentes computacionalmente son:

1. **Extracción 2D de características.** Esta es la etapa más costosa, llegando a consumir en el caso D el 99 % del tiempo total de ejecución. La causa principal es la cantidad de píxeles que contiene la imagen escaneada: las operaciones de visión por computador (binarización, adelgazamiento, etc.) deben analizar cada píxel de la imagen, por lo que su coste computacional escala de forma proporcional con su tamaño.

   Esto revela que la principal causa de ineficiencia no es la complejidad del modelo 3D en sí, sino la resolución de las imágenes de entrada en el preprocesado de imágenes.

2. **Reconstrucción de aristas y detección de caras 3D.** Tras la extracción, estas son las dos etapas más relevantes. Su complejidad depende del número de vértices y aristas del modelo de entrada, ya que la reconstrucción de aristas explora combinaciones de pares de vértices y la detección de caras se basa en algoritmos de recorrido de grafos para hallar ciclos.

3. **Normalización y reconstrucción de puntos 3D.** Estas etapas son las más rápidas en los casos evaluados, con tiempos de ejecución mínimos ($< 1\,ms$), debido al número moderado de vértices en los ejemplos.

En conclusión, podemos afirmar que el rendimiento es eficiente para su uso con bocetos digitales y se ajusta a los objetivos del proyecto. Para imágenes escaneadas, los tiempos se mantienen en un rango aceptable pese al aumento notable de su tiempo de procesamiento.

5.4. Casos de prueba adicionales

Además de los casos expuestos en este capítulo, la evaluación completa se ha realizado sobre un total de 17 ejemplos del directorio `examples`, que incluye figuras destinadas a complementar los distintos tipos de casos mencionados. Una reconstrucción se considera válida si produce una malla cerrada, supera la validación de consistencia (función `check_model_consistency`), y se puede exportar y visualizar correctamente en formato OBJ.

Usando la configuración por defecto (véase tabla 5.1) y aplicando ajustes cuando fue necesario (distancia de fusión de vértices, aproximación poligonal, etc.), la tasa de éxito fue del 82,35 % (14 de los 17 casos evaluados).

Los tiempos de ejecución observados en los demás ejemplos de prueba son consistentes con los mostrados en la tabla 5.2. En particular, las entradas de alta resolución o con muchos contornos incrementan el coste de la fase de extracción, mientras que la complejidad del modelo (número de vértices y aristas) afecta principalmente a las etapas de reconstrucción de aristas y detección de caras.

En los tres casos fallidos, las causas principales fueron (i) ambigüedades por simetrías y repeticiones que introducen vértices falsos (véase sección 5.1.5), (ii) detección incorrecta del cierre morfológico cuando se aplica sobre segmentos cercanos, y (iii) proyecciones degeneradas con ejes no alineados.

CAPÍTULO

VI

CONCLUSIONES Y TRABAJO FUTURO

Este capítulo recoge las principales conclusiones del proyecto, contrasta los resultados obtenidos con los objetivos iniciales propuestos y se analizan las limitaciones observadas para proponer líneas de mejora futura de la herramienta.

6.1. Discusión de resultados

Se ha desarrollado una herramienta capaz de automatizar el paso de bocetos ortogonales 2D en papel liso (o con ligera curvatura, siempre que no introduzca deformaciones apreciables) o digital a una malla 3D funcional, agilizando el flujo de trabajo de artistas y diseñadores. Después de la evaluación de los resultados obtenidos, podemos concluir que el objetivo general propuesto se ha cumplido satisfactoriamente, si bien persisten limitaciones significativas (resolución de ambigüedades por simetrías o repeticiones y automatización del ajuste de parámetros) que señalan margen de mejora y orientan el trabajo futuro.

Respecto a los objetivos específicos definidos en la introducción (sección 1.2), podemos afirmar que se han cubierto correctamente:

- **Modelo de datos y *pipeline*.** Se ha propuesto un flujo completo desde la etapa de preprocesado de imágenes (binarización, adelgazamiento, aproximación poligonal, etc.), pasando por la extracción de características 2D (vértices y aristas), la reconstrucción de la geometría 3D (puntos, aristas y caras) y las verificaciones de consistencia (por vista y global) hasta la visualización y exportación del resultado.

- **Prototipo funcional.** La herramienta integra el *pipeline* en una interfaz gráfica que permite cargar vistas, ajustar parámetros, previsualizar el resultado y exportarlo en formato OBJ.

- **Evaluación.** Los casos de prueba muestran un comportamiento robusto en poliedros convexos y cóncavos, con y sin aristas ocultas. Se han medido los tiempos de ejecución por etapa, concluyendo que la etapa de preprocesado es la más costosa computacionalmente, especialmente en escaneos de alta resolución

(una solución habitual es reducir la resolución de entrada), mientras que la normalización y la reconstrucción de puntos son instantáneas; la reconstrucción de aristas y caras crece con el número de vértices, pero se mantiene manejable para los tamaños objetivo.

La evaluación mediante casos de estudio (capítulo 5) muestra que el enfoque basado en la reconstrucción de puntos por correspondencia cruzada (sección 3.2.3) y de aristas por compatibilidad de proyecciones con aristas auxiliares colineales (sección 3.2.4) produce mallas cerradas y coherentes. Al mismo tiempo, los casos con simetrías y repeticiones (sección 5.1.5) evidencian que el sistema puede generar vértices falsos cuando distintas proyecciones comparten coordenadas, reforzando la necesidad de criterios adicionales de selección y validación.

6.2. Líneas de mejora y extensiones

A partir de las limitaciones observadas (véase sección 5.2), se proponen líneas de trabajo listadas por orden de prioridad:

- **Resolución de ambigüedades geométricas.** La aparición de vértices falsos ocurre porque el sistema actual acepta las conexiones al ser válidas localmente pero no comprueba la coherencia global de la figura. Una posible solución sería definir la selección final como un problema de *optimización*: elegir la figura más simple (mínimo número de vértices, aristas y caras) de entre todas las reconstrucciones posibles, respetando la compatibilidad con las tres proyecciones. En esta línea, también se podría añadir un paso de verificación donde el usuario pueda descartar vértices o aristas de forma manual en la reconstrucción 3D.
- **Geometría curva.** Una extensión significativa sería añadir soporte para contornos circulares, lo que implicaría (i) implementar métodos de detección de círculos, elipses y arcos y (ii) generar las primitivas a partir de las curvas 2D detectadas. Esto permitiría reconstruir un mayor abanico de figuras y generalizar su uso.
- **Reconstrucción de múltiples objetos.** El modelo actual procesa todos los contornos como una única figura. Una solución sería detectar componentes separadas en las proyecciones y tratarlas como objetos independientes en la reconstrucción, exportando tanto las figuras individuales como el modelo formado por la unión de todas las componentes.
- **Integración en herramientas profesionales.** Para maximizar su utilidad, podría integrarse como una extensión en herramientas de modelado 3D (como Blender) que permita realizar reconstrucciones desde el mismo entorno.

Estas líneas de mejora convertirían el prototipo en una herramienta de modelado más versátil y útil, manteniendo la filosofía original: generar una malla 3D a partir de los perfiles de un objeto.

INTRODUCTION

Three-dimensional reconstruction from orthographic projections is a classic problem in descriptive geometry and computer vision (Monge, 1799; Sugihara, 1986).

In *Géométrie Descriptive*, Gaspard Monge formalized the idea that the shape of a solid can be expressed through its orthographic projections, where each view retains two coordinates:

- Elevation $\rightarrow (x, y)$, corresponding to width and height.
- Plan $\rightarrow (x, z)$, corresponding to width and depth.
- Section $\rightarrow (y, z)$, corresponding to height and depth.

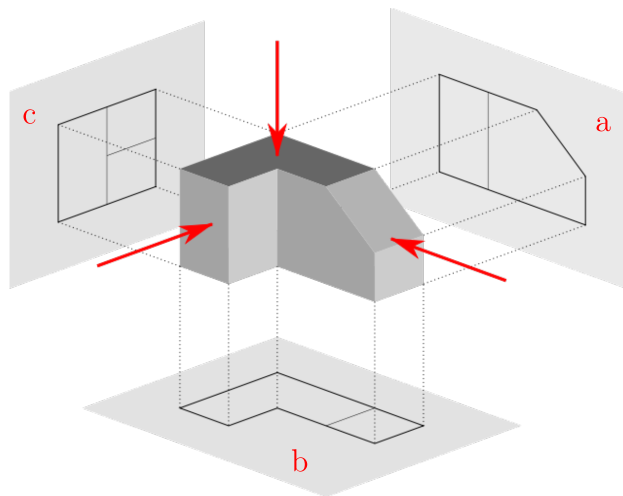


Figure 6.1: Orthographic views of an object:
(a) elevation, (b) plan, and (c) section.
(IDIS, 2025)

The key idea is that the three projections are mutually coherent (see figure 6.1): the coherent intersection of the coordinate pairs allows us to recover (x, y, z) points of the solid. The main challenge is not computing the projections but recovering the 3D shape by solving the *correspondence* across views: deciding which point in the *elevation* matches which point in the *plan* and which in the *section*. In computer vision, the term "correspondence" refers to matching features between real images;

here, we use it for the geometric matching of vertices and edges across the three views.

For example, if the elevation shows a point with coordinates (x=10, y=15), we look in the plan for points whose x is $\approx$ 10 and in the section for points whose y is $\approx$ 15. If, for any of those candidates, the z coordinate matches in both plan and section, we recover the 3D point $(10, 15, z)$. The problem is challenging even with exact data: ambiguities can arise even without approximation errors, so we introduce a small *tolerance* when comparing coordinates; this tolerance does not resolve the problem's inherent ambiguity but absorbs noise and slight misalignments across views. This rule guides the initial matching, which is later validated with consistency criteria (see sections 3.2.3 and 3.2.6).

In practice, many ideas from artists and designers still begin as sketches on paper or digitally, and translating those views into a usable 3D model remains a slow and inefficient manual process.

This project addresses that need by providing a tool that translates three-view drawings into a closed 3D mesh that can be visualized and exported. The goal is to streamline the initial stage of the 3D design and modeling workflow: to move from a structural sketch to a base geometry that serves as a starting point for texturing, animation, or detailed editing.

Motivation

An early milestone was *Sketchpad* (Sutherland, 1963), which introduced interactive graphical editing using a light pen that reported its position on the screen. In parallel, General Motors and IBM presented *DAC-1* (see figure 6.2), one of the first interactive CAD systems (Computer History Museum, 2011; Krull, 1994), capable of digitizing 2D drawings and applying geometric transformations (rotation, translation, etc.) to them. These advances marked the beginning of tools intended to transform graphical representations into manipulable geometric models.

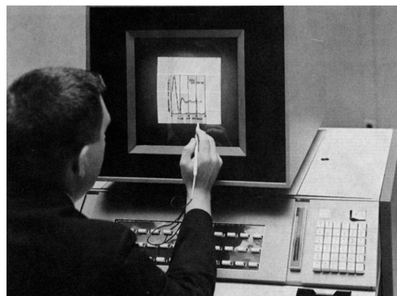


Figure 6.2: DAC-1, the first commercial CAD system.
(Computer History Museum, 2011)

Six decades later, hardware and software are accessible to everyone, yet the need persists: current systems still have limitations when automatically converting three orthographic views into a usable base solid while minimizing user intervention. Specifically: (i) commercial CAD systems require reconstructing the 3D model via

operations (extrusion, revolution, etc.) and correspondence decisions that the user must specify; and (ii) computer vision approaches rely on dense data such as point clouds or scans rather than technical drawings with limited information.

Despite technological advances in these fields, most automatic solutions assume inputs and objectives different from those of this work. Therefore, automatic reconstruction is not trivial; automating reconstruction from three views poses long-studied difficulties stemming from various factors (Sugihara, 1986):

- **Correspondence ambiguity.** A single point may have several candidates with compatible coordinates in the other views, specifically in shapes with symmetry or repetitions.
- **Degenerate projections.** When edges or faces are parallel to the projection direction or plane, they collapse to points or segments. This produces overlaps and coincidences that do not always reflect actual relationships.
- **Topological consistency.** Open surfaces, self-intersecting faces, or vertices with insufficient connections can yield invalid solids.
- **Limited visibility.** Hidden, dashed, or fragmented edges hinder correct detection.
- **Noise and distortions.** Factors such as stroke thickness, slight scan rotations, or scale variations alter reliable vertex detection and continuous edges.
- **Uniqueness of the solution.** Different solids can share the same orthographic projections. Without additional hypotheses or selection criteria, the problem admits multiple plausible solutions.

This work fits into that problem space and proposes an approach different from traditional CAD systems: automating the transition from three coherent views to an initial mesh. The aim is to minimize the mechanical tasks associated with manual model construction, accelerating design or prototyping processes and prioritizing the creative side of design.

Objectives and scope

The overall objective is to develop a tool that, given three 2D orthographic views of an object, drawn on paper or an electronic device, automatically generates a closed 3D mesh (no holes in the surface and each edge belongs to exactly two faces, with normals coherently oriented outward) that can be exported to a standard format.

Specific objectives

Below are the specific objectives of the work and, for each one, the conditions that will verify their fulfillment (*acceptance criteria*):

(a) Define a **data model** and a robust *pipeline* for reconstruction from three projections. Acceptance criteria: (i) a clear specification of model structures (e. g., edges as ordered vertex pairs, faces as an ordered list of vertices, etc.), (ii) a diagram of the complete flow (from input to export), and (iii) a description of the algorithmic complexity of each stage.

(b) Implement a **functional prototype** in Python with a graphical interface to load views, adjust tolerances, and export the result in OBJ format. It will be considered complete when it allows: (i) loading the three orthographic views, (ii) adjusting relevant reconstruction parameters, (iii) visualizing the result in 2D / 3D, and (iv) exporting an OBJ file compatible with modeling tools such as Blender or MeshLab.

(c) Prepare a representative **test set** (primitives, concavities, symmetries). It will be considered complete when there is a bank of at least 10-15 cases with their expected results and documentation for each case.

(d) **Evaluate** accuracy and robustness and **measure** execution times per stage. The acceptance criterion is to report the overall rate of valid reconstructions and the execution times per phase for each test example, indicating the most expensive stages and why.

Scope and assumptions

To bound the problem and ensure reproducible results, we assume:

(i) Coherent orthographic views, without perspective distortion and with axes aligned across views (or slight rotations that can be corrected).

(ii) Sufficient resolution to reliably detect contours and vertices.

(iii) Polygonal geometry with planar faces, where the input contours are straight segments. Curves are not supported; if they appear, they will be approximated linearly and exact geometric reconstruction is not guaranteed.

These assumptions focus the effort on the correspondence problem and on constructing a closed mesh that preserves straight edges and planar faces.

Work plan

The work is organized as a sequence of tasks:

1. **Analysis and design.** Problem definition, requirements, and input assumptions.
2. **Test suite.** Preparation of a bank of examples (primitives, concavities, and combinations) to progressively validate each stage of the pipeline and identify violations of the assumptions.

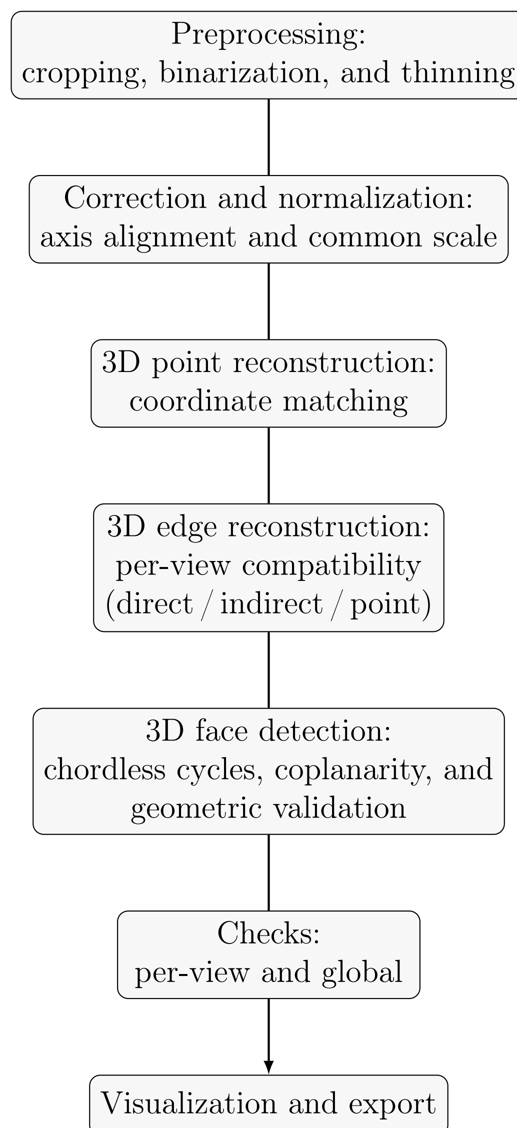


Figure 6.3: Summary of the reconstruction *pipeline*.

3. **Preprocessing and 2D extraction.** Conversion to grayscale and binarization to separate stroke and background; stroke thinning to obtain a 1-pixel skeleton; contour detection and approximation by polylines (a chain of connected straight segments). The resulting vertices and segments constitute the 2D graph (nodes and edges) on which reconstruction will be performed.

4. **Normalization and alignment.** Orthogonality correction of edges and unification of origin and scale.

5. **3D reconstruction.** Coordinate matching across views and verification of connectivity and global consistency of the generated 3D graph.

6. **Face detection.** Search for minimal cycles in the projections and checks of connectivity, coplanarity, and face orientation.

7. **Visualization and export.** Preview and export of the final 3D model.

8. **Evaluation and documentation.** Timing measurements, analysis of edge cases, and final write-up.

Figure 6.3 shows the sequence of operations applied to the views to turn them into a 3D mesh. This sequence is detailed in chapter 3.

Document structure

The document is organized as follows:

1. **Introducción.** Project context, objectives and work plan.

2. **Antecedentes y Fundamentos.** The correspondence problem in 3D reconstruction from projections and the libraries and algorithms used. Techniques explored but ultimately not adopted are included, along with their utility and limitations compared to this work's approach to justify design decisions.

3. **Metodología e Implementación.** Detailed description of the workflow: preprocessing, normalization, reconstruction of points and edges, face generation, consistency checks, visualization, and export. Relevant code snippets are included.

4. **Interfaz de Usuario.** Interface design and architecture.

5. **Evaluación y Resultados.** Test cases, per-phase execution metrics, failure analysis, and critical discussion.

6. **Conclusiones y Trabajo Futuro.** Lessons learned, limitations, and lines for improvement and extension.

External resources

The project code is available in the GitHub repository: `https://github.com/asegar01/visione`. The repository structure is as follows:

```
.
├── docs/ (report)
├── examples/ (sample view images)
├── src/ (source code)
├── README.md (usage instructions)
└── requirements.txt
```

The set of **data and examples** used during development and in the evaluation (see chapter 5) is included in the `examples` directory of the repository, while the **usage guide and required dependencies** are detailed in `README.md` and `requirements.txt`, respectively.

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the main conclusions of the project, contrasts the results with initial objectives, and analyzes the observed limitations to propose lines for future improvements to the tool.

Discussion of results

A tool has been developed that automates the transition from 2D orthographic sketches on plain paper (or with slight curvature, provided it does not introduce appreciable distortions) or in digital form to a functional 3D mesh, streamlining the workflow of artists and designers. After evaluating the results, we conclude that the overall objective has been satisfactorily met, although significant limitations remain (resolving ambiguities due to symmetries or repetitions and automating parameter tuning) that indicate room for improvement and guide future work.

With respect to the specific objectives defined in the introduction (section 1.2), we can state that they have been correctly fulfilled:

- **Data model and *pipeline*.** A complete workflow has been proposed from the image preprocessing stage (binarization, thinning, polygonal approximation, etc.), through the extraction of 2D features (vertices and edges), 3D geometry reconstruction (points, edges, and faces), and consistency checks (per view and global), all the way to visualization and export of the result.

- **Functional prototype.** The tool integrates the *pipeline* into a graphical interface that allows loading views, adjusting parameters, previewing the result, and exporting it in OBJ format.

- **Evaluation.** The test cases show robust behavior on complex and concave polyhedra, with and without hidden edges. Execution times per stage have been measured, concluding that the preprocessing stage is the most computationally expensive, especially for high-resolution scans (a common solution is to reduce input resolution), whereas normalization and point reconstruction are essentially instantaneous; edge and face reconstruction grows with the number of vertices but remains manageable for the target sizes.

The evaluation via case studies (chapter 5) shows that the approach based on reconstructing points through cross-view correspondence (section 3.2.3) and edges

through projection compatibility with collinear auxiliary edges (section 3.2.4) produces closed and coherent meshes. At the same time, cases with symmetries and repetitions (section 5.1.5) make it evident that the system can generate spurious vertices when different projections share coordinates, reinforcing the need for additional selection and validation criteria.

Lines for improvement and extensions

Based on the observed limitations (see section 5.2), we propose the following lines of work, listed in order of priority:

- **Resolving geometric ambiguities.** Spurious vertices appear because the current system accepts connections that are locally valid but does not check the figure's global coherence. One possible solution is to pose the final selection as an *optimization* problem: choose the simplest figure (minimum number of vertices, edges, and faces) among all possible reconstructions while respecting compatibility with the three projections. Along these lines, a verification step could be added where the user can manually discard vertices or edges in the 3D reconstruction.

- **Curved geometry.** A significant extension would be to add support for circular contours, which would involve (i) implementing methods to detect circles, ellipses, and arcs; and (ii) generating primitives from the detected 2D curves. This would enable reconstruction of a broader range of shapes and generalize the tool's use.

- **Reconstruction of multiple objects.** The current model processes all contours as a single figure. A solution would be to detect separate components in the projections and treat them as independent objects in reconstruction, exporting both individual figures and the model formed by their union.

- **Integration into professional tools.** To maximize utility, it could be integrated as an add-on in 3D modeling tools (such as Blender) to perform reconstructions within the same environment.

These improvement lines would turn the prototype into a more versatile and useful modeling tool, while maintaining the original philosophy: generate a 3D mesh from an object's profiles.

BIBLIOGRAFÍA

La verdadera sabiduría está en reconocer
la propia ignorancia.

Atribuida a Sócrates (Platón, *Apología*, 21d)

AUTODESK. Raster design toolset in AutoCAD. 2025. Accedido el 26-08-2025.

BERNARDINI, F., MITTLEMAN, J., RUSHMEIER, H., SILVA, C. y TAUBIN, G. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, vol. 5(4), páginas 349–359, 1999.

BRADSKI, G. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.

CANNY, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8(6), páginas 679–698, 1986.

CHOY, C. B., XU, D., GWAK, J., CHEN, K. y SAVARESE, S. 3D-R2N2: A unified approach for single and multi-view 3d object reconstruction. 2016.

CLOWES, M. B. On seeing things. *Artif. Intell.*, vol. 2(1), página 79–116, 1971. ISSN 0004-3702.

COMPUTER HISTORY MUSEUM. Computerizing car design: The dac-1. 2011. Accedido el 26-08-2025.

DASSAULT SYSTÈMES. 2d to 3d conversion. 2023. Accedido el 26-08-2025.

DELAUNAY, B. Sur la sphère vide. *Bulletin de lÁcademie des Sciences de lÚnion des Républiques Soviétiques Socialistes. Classe des sciences mathématiques et naturelles*, 1934.

DIESTEL, R. *Graph Theory*. Springer Publishing Company, Incorporated, 5th edición, 2017. ISBN 3662536218.

DOUGLAS, D. H. y PEUCKER, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, páginas 112–122, 1973.

Duda, R. O. y Hart, P. E. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM*, vol. 15(1), página 11–15, 1972. ISSN 0001-0782.

Edelsbrunner, H., Kirkpatrick, D. y Seidel, R. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, vol. 29(4), páginas 551–559, 1983.

Guo, Z. y Hall, R. W. Parallel thinning with two-subiteration algorithms. *Commun. ACM*, vol. 32(3), página 359–373, 1989. ISSN 0001-0782.

Hagberg, A., Swart, P. y Chult, D. Exploring network structure, dynamics, and function using NetworkX. En *Proceedings of the 7th Python in Science Conference*. 2008.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. y Oliphant, T. E. Array programming with NumPy. *Nature*, vol. 585(7825), páginas 357–362, 2020.

Huffman, D. A. Impossible objects as nonsense sentences. En *Machine Intelligence 6*. Edinburgh University Press, 1971.

Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, vol. 9(3), páginas 90–95, 2007.

IDIS. Monge y el sistema diédrico. 2025. Accedido el 3-09-2025.

Igarashi, T., Matsuoka, S. y Tanaka, H. *Teddy: A Sketching Interface for 3D Freeform Design*. Association for Computing Machinery, New York, NY, USA, 1 edición, 1999. ISBN 9798400708978.

Kazhdan, M., Bolitho, M. y Hoppe, H. Poisson surface reconstruction. En *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, página 61–70. Eurographics Association, Goslar, DEU, 2006. ISBN 3905673363.

Koffka, K. *Principles of Gestalt Psychology*. Harcourt Brace And Company, 1935.

Köhler, W. *Gestalt Psychology*. Liveright, 1929.

Krull, F. The origin of computer graphics within General Motors. *IEEE Annals of the History of Computing*, vol. 16(3), páginas 40–, 1994.

Laurentini, A. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16(2), páginas 150–162, 1994.

Mark Roseman. Tkdocs: Modern Tk best practices. `https://tkdocs.com/`, 2025. Accedido el 26-08-2025.

Matplotlib Development Team. Matplotlib: Visualization with Python. `https://matplotlib.org/`, 2025. Accedido el 26-08-2025.

Monge, G. *Géométrie Descriptive*. Baudouin, 1799.

NetworkX Developers. NetworkX: Software for complex networks. `https://networkx.org/`, 2025. Accedido el 26-08-2025.

NumPy Developers. NumPy. `https://numpy.org/`, 2025. Accedido el 26-08-2025.

OpenCV Team. OpenCV (open source computer vision library). `https://opencv.org/`, 2025. Accedido el 26-08-2025.

Otsu, N. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9(1), páginas 62–66, 1979.

Python Software Foundation. Tkinter: Graphical user interfaces with Tk. `https://docs.python.org/3.12/library/tk.html`, 2025. Accedido el 26-08-2025.

scikit-image contributors. scikit-image: Image processing in Python. `https://scikit-image.org/`, 2025. Accedido el 26-08-2025.

Sugihara, K. *Machine interpretation of line drawings*. MIT Press, Cambridge, MA, USA, 1986. ISBN 0262192543.

Sutherland, I. E. Sketchpad: a man-machine graphical communication system. En *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, AFIPS '63 (Spring), página 329–346. Association for Computing Machinery, New York, NY, USA, 1963. ISBN 9781450378802.

Suzuki, S. y Abe, K. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, vol. 30(1), páginas 32–46, 1985. ISSN 0734-189X.

van der Walt, S., Schönberger, J., Nunez-Iglesias, J., Boulogne, F., Warner, J., Yager, N., Gouillart, E. y Yu, T. scikit-image: image processing in Python. *PeerJ*, vol. 2, 2014.

Waltz, D. L. Understanding line drawings of scenes with shadows. En *The Psychology of Computer Vision*. 1975.

Xie, H., Yao, H., Sun, X., Zhou, S. y Zhang, S. Pix2vox: Context-aware 3D reconstruction from single and multi-view images. En *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, página 2690–2698. IEEE, 2019.

Zhang, T. Y. y Suen, C. Y. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, vol. 27(3), página 236–239, 1984. ISSN 0001-0782.