



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

COMPARACIÓN DE ALGORITMOS DE CÁLCULO DEL SKELETON Y SU
APLICACIÓN EN BIOLOGÍA

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN
COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

ALEJANDRO ANDRÉS LAVADO ABARZÚA

PROFESORES GUÍAS:

NANCY HITSCHFELD KAHLER
MAURICIO CERDA VILLABLANCA

MIEMBROS DE LA COMISIÓN:

NELSON BALOIAN TATARYAN
JÉRÉMY BARBAY
DOMINGO MERY QUIROZ

SANTIAGO DE CHILE
2018

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN
Y AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: ALEJANDRO ANDRÉS LAVADO ABARZÚA
FECHA: 26/01/2018
PROF. GUÍAS: MAURICIO CERDA VILLABLANCA, NANCY HITSCHFELD KAHLER

COMPARACIÓN DE ALGORITMOS DE CÁLCULO DEL SKELETON Y SU APLICACIÓN EN BIOLOGÍA

Los algoritmos de cálculo del *skeleton* son una herramienta computacional de amplia utilización en el procesamiento de imágenes y volúmenes médicos y biológicos. Se trata de procedimientos que reducen una figura a un conjunto de líneas que pasan por su centro.

El *skeleton* de una figura puede ser calculado siguiendo estrategias muy diferentes. Debido a esto, cada algoritmo de cálculo del *skeleton* puede producir un resultado muy distinto a los demás algoritmos. Ahora bien, cuando se está trabajando en una aplicación donde se requiere el *skeleton*, ¿cómo elegir el mejor algoritmo para calcularlo?

En esta tesis se proponen métricas originadas en el análisis morfológico de estructuras biológicas para responder cuantitativamente a la pregunta anterior, como el largo total del *skeleton*, su número de nodos y sus ángulos de bifurcación. Estas métricas permiten caracterizar numéricamente un *skeleton* y compararlo con otros. De esta manera, el mejor algoritmo para una aplicación en específico puede ser seleccionado en base a los valores de las métricas relevantes para esa aplicación.

Para demostrar la efectividad de estas métricas, se implementaron tres algoritmos de cálculo del *skeleton* basados en principios teóricos distintos: adelgazamiento topológico, cálculo del *skeleton* basado en la divergencia y cálculo del *skeleton* basado en la distancia. Estos algoritmos, más un cuarto basado en contracción de mallas, fueron utilizados para calcular los *skeletons* de modelos biológicos simulados y reales. Los *skeletons* de modelos simulados permitieron medir la desviación de cada algoritmo con respecto al valor ideal de cada métrica, revelando diferencias significativas en algunos casos. Ejemplo de esto es la métrica del largo total en estructuras tipo neurona: el cálculo del *skeleton* por contracción de mallas produce una estructura significativamente más corta que el *skeleton* calculado mediante un algoritmo basado en la distancia, cuyo largo total es cercano al real. Sin embargo, el algoritmo de contracción de mallas resulta más apropiado para calcular los ángulos de bifurcación. Por último, las métricas para *skeletons* de modelos reales ilustraron marcadas diferencias entre los resultados producidos por cada algoritmo para la misma figura.

Tabla de Contenido

| | |
|--------------------------------------------------------------------------------------|-----------|
| | I |
| 1. Introducción | 1 |
| 1.1. Definiciones del <i>skeleton</i> | 3 |
| 1.2. Aplicaciones del <i>skeleton</i> | 8 |
| 1.3. Representación discreta de volúmenes | 12 |
| 1.4. Componentes conexas | 15 |
| 1.5. Clases de algoritmos de cálculo del <i>skeleton</i> | 21 |
| 1.6. Comparaciones de algoritmos de cálculo del <i>skeleton</i> | 23 |
| 1.7. Hipótesis y objetivos | 26 |
| 1.8. Estructura de esta tesis | 26 |
| 2. Cálculo del <i>Skeleton</i> por Adelgazamiento Topológico (Palágyi y Kuba) | 27 |
| 2.1. Fundamentos teóricos | 28 |
| 2.2. Pseudocódigo del algoritmo | 29 |
| 2.3. Implementación | 30 |
| 2.4. Discusión | 34 |
| 3. Cálculo del <i>Skeleton</i> Basado en la Divergencia (Siddiqi et al.) | 36 |
| 3.1. Fundamentos teóricos | 36 |
| 3.2. Pseudocódigo del algoritmo | 38 |
| 3.3. Implementación | 39 |
| 3.4. Discusión | 41 |
| 4. Cálculo del <i>Skeleton</i> Basado en la Distancia (Arcelli et al.) | 44 |
| 4.1. Fundamentos teóricos | 44 |
| 4.2. Pseudocódigo del algoritmo | 48 |
| 4.3. Implementación | 48 |
| 4.4. Discusión | 57 |

| | |
|----------------------------------------------------------------------------------|-----------|
| 5. Comparación | 59 |
| 5.1. Métricas de comparación | 59 |
| 5.2. Comparación en volúmenes estándar | 62 |
| 5.3. Comparación en volúmenes simulados | 68 |
| 5.4. Comparación en volúmenes de biología | 78 |
| 6. Conclusiones | 83 |
| Bibliografía | 86 |
| Apéndices | 96 |
| A . Comparación en modelos generados a partir de segmentos curvilíneos | 96 |
| B . Obtención de volúmenes estándar | 97 |
| C . Formatos de archivos de vóxeles y programas para su visualización | 99 |
| D . Clasificación de vóxeles para el algoritmo de Arcelli et al. | 100 |

Capítulo 1

Introducción

El *skeleton* fue propuesto por Harry Blum en 1967 como una manera de capturar lo esencial de la forma de una figura biológica [14]. A diferencia de otros estímulos visuales como el color, la intensidad luminosa o el movimiento, Blum observó que el estudio de la forma habría estado sesgado por la geometría de las figuras. Las formas de figuras complejas, comunes en Biología, casi no eran materia de estudio debido a la falta de herramientas para transformarlas en una combinación de componentes más simples. Blum postuló que la unión de los ejes medios, que más tarde sería llamada *skeleton* [39], podría ser la herramienta necesaria. Dada una figura, Blum define su *skeleton* como una estructura delgada y centrada, geométrica y topológicamente expresiva. La Figura 1.1 es un ejemplo del *skeleton* tal como lo definió Blum.

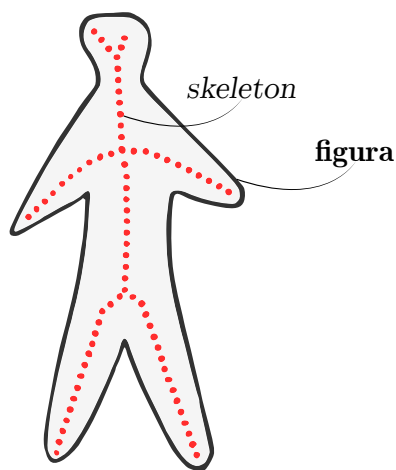


Figura 1.1: Diagrama del *skeleton* de una figura antropomórfica, como aparece en el artículo original de Blum [14].

Cincuenta años más tarde, el *skeleton* de Blum se ha convertido en uno de los descriptores de figuras más usados [89]. La operación de calcular el *skeleton* juega un rol central en muchas aplicaciones que involucran la búsqueda, manipulación, análisis o compresión de figuras en el computador,

en áreas que van mucho más allá de la Biología [76]. La operación de calcular el *skeleton* es también el foco de esta tesis. En principio, interesa responder la siguiente pregunta: ¿cuál es la mejor manera de llevar a cabo esta operación? No obstante, existen dos dificultades importantes que impiden dar una respuesta rápida a la pregunta anterior.

1. Desde la primera definición del *skeleton* se han propuesto decenas de algoritmos para calcularlo. Es por lo tanto inviable comparar todos los métodos conocidos, pero sí es posible categorizarlos y seleccionar algoritmos representativos de cada categoría. Sin embargo, las implementaciones de los algoritmos que calculan el *skeleton* muy rara vez acompañan a sus publicaciones [88].
2. No existe un consenso para la definición del *skeleton*. Es más, existen aplicaciones donde calcular un *skeleton* demasiado ajustado a alguna definición puede ser perjudicial [22]. Esto significa que cualquier comparación entre estos algoritmos debe plantearse de acuerdo a las restricciones y objetivos de alguna aplicación en particular [76].

En consideración con lo anterior, este trabajo de tesis se divide en dos partes. La primera parte consiste en implementar algoritmos representativos de las distintas maneras que se han propuesto para calcular el *skeleton*. A continuación se comparan los *skeletons* calculados por estos algoritmos. Los datos (las figuras cuyo *skeleton* se quiere calcular) y la justificación para la comparación propuesta están ligados a una aplicación muy similar a la originalmente visionada por Blum. Interesa evaluar los *skeletons* según métricas relevantes para la cuantificación de objetos microscópicos capturados utilizando microscopía óptica. En definitiva, la pregunta que busca responder esta tesis queda reformulada como sigue: ¿Qué clase de algoritmos para calcular el *skeleton* es mejor para ciertas aplicaciones de Biología?

El resto de este capítulo introductorio busca precisar lo señalado hasta este punto. Se revisan distintas maneras de definir el *skeleton* de figuras 2D y 3D. Se detallan las propiedades del *skeleton* de acuerdo a la literatura, argumentando por qué satisfacer estas propiedades puede ser contraproducente. Se describen algunas aplicaciones del *skeleton*, poniendo énfasis en la aplicación que motiva este trabajo de tesis. Se presentan los conceptos relevantes de topología discreta para comprender los fundamentos e implementaciones de los algoritmos. Finalmente, se introduce la clasificación usada para seleccionar los algoritmos implementados representantes de cada categoría y se revisan las comparaciones de algoritmos de cálculo del *skeleton* encontradas en la literatura.

1.1. Definiciones del *skeleton*

En la literatura es posible encontrar por lo menos cuatro definiciones formales para el *skeleton* [96]. En esta sección se detallan las dos definiciones más populares [23], que al mismo tiempo son las más relevantes para este trabajo.

1.1.1. El *skeleton* a partir de la metáfora del incendio

La metáfora del incendio de Blum [14] establece que el *skeleton* de un objeto puede obtenerse encendiéndole fuego a toda su frontera simultáneamente. Luego, se asume que los frentes de llamas se propagan a una velocidad constante hacia el interior del objeto. Una vez que el fuego ha alcanzado todo el objeto, el *skeleton* queda formado en los lugares donde dos frentes distintos se encontraron. En otras palabras, el *skeleton* corresponde a los puntos de extinción, donde el fuego no pudo seguir avanzando por haberse encontrado consigo mismo. Estos puntos se conocen hoy en día como *puntos de choque* [29]. La Figura 1.2 ilustra esta definición para una imagen y la Figura 1.3 la muestra en un volumen.

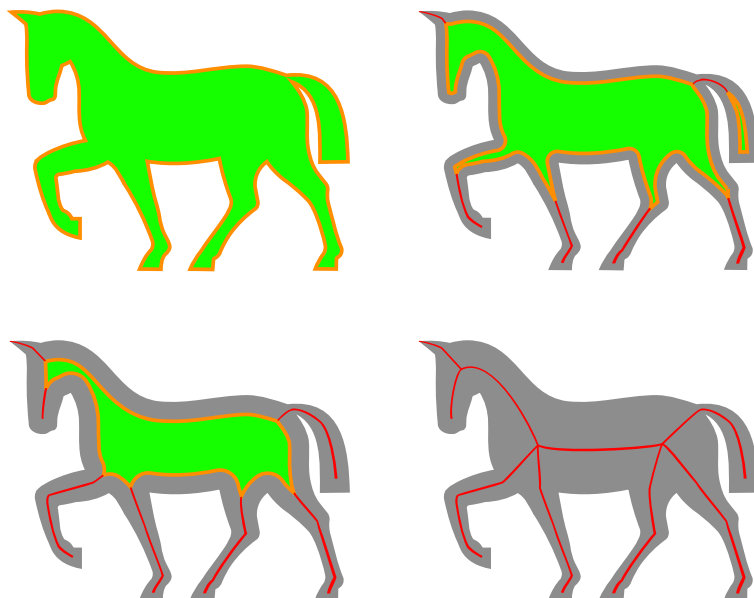


Figura 1.2: Progresión del incendio de Blum en una imagen.

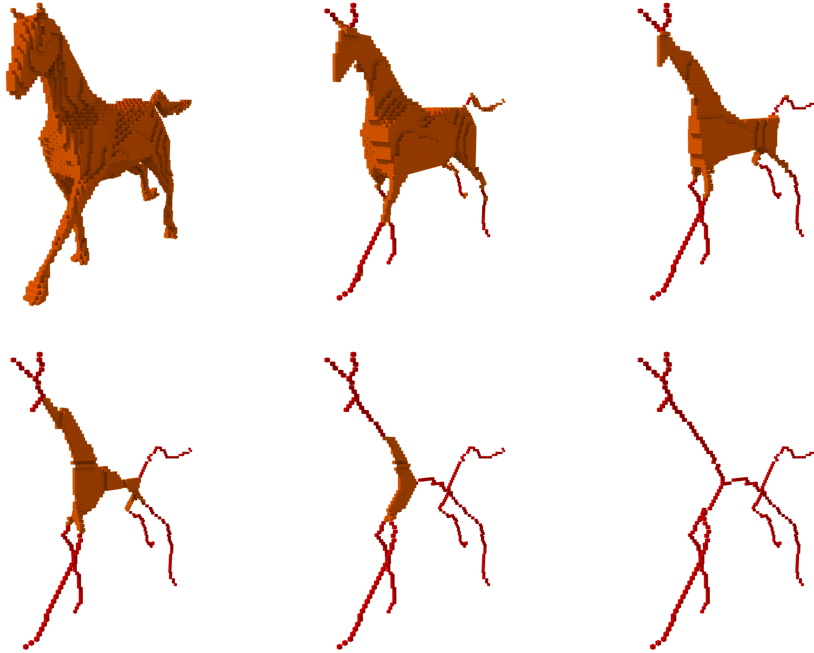


Figura 1.3: Progresión del incendio de Blum en un volumen.

La propagación del fuego desde el borde de la figura es isotrópica, pudiendo ser descrita formalmente mediante la ecuación:

$$\frac{\partial \mathbf{C}(t)}{\partial t} = f \mathbf{n}, \quad (1.1)$$

donde $\mathbf{C}(t)$ representa el borde de la figura a lo largo del tiempo, f la velocidad constante de propagación y \mathbf{n} la normal interior. Encontrar el *skeleton* según esta definición significa detectar los puntos de choque de esta ecuación.

En este trabajo de tesis se implementaron dos algoritmos que se basan en esta definición. Estos son el algoritmo de adelgazamiento del Capítulo 2 y el algoritmo de detección de singularidades del Capítulo 3. En particular, el algoritmo del Capítulo 3 se fundamenta en un desarrollo formal de la ecuación anterior.

1.1.2. El *skeleton* como lugar geométrico

La segunda definición más común para el *skeleton* también aparece esbozada en el artículo de Blum. En él, Blum nota que cada punto de choque p , al ser el lugar de encuentro de frentes de propagación diferentes, siempre es equidistante a por lo menos dos puntos distintos del borde de la figura, que a la vez son los puntos del borde de la figura más cercanos a p . Es decir que, dado un punto p del interior de la figura, p es un punto del *skeleton* si existen dos puntos distintos q y r en el borde \mathbf{C} de la figura tales que se cumplen las dos condiciones:

$$\|q - p\| = \|r - p\| = \mathcal{R}_p,$$

$$\nexists s \in \mathbf{C} \mid \|s - p\| < \mathcal{R}_p,$$

para algún $\mathcal{R}_p > 0$.

Entonces, para todo punto p del *skeleton* es posible definir una esfera (o un disco en el caso 2D) con centro p y radio \mathcal{R}_p completamente inscrita en la figura. Además, el borde de esta esfera comparte por lo menos dos puntos con el borde de la figura. Se dice que una esfera con estas propiedades es una *esfera maximal inscrita* en la figura.

El *skeleton* de una figura queda definido, por lo tanto, como el lugar geométrico de los centros de las esferas maximales inscritas en la figura. El problema de calcular el *skeleton* se traduce en la detección de estos centros. La Figura 1.4 muestra un punto del *skeleton* de un paralelepípedo y su correspondiente esfera maximal.

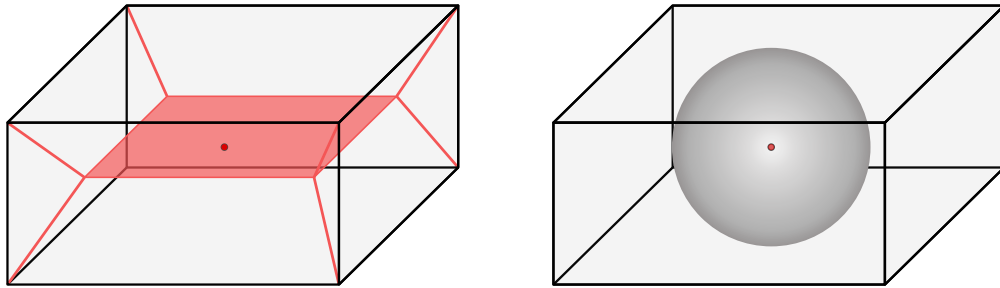


Figura 1.4: A la izquierda, un volumen rectangular y su *skeleton*. A la derecha, un punto del *skeleton* y su correspondiente esfera maximal inscrita.

Detectar los centros de las esferas maximales es excesivamente costoso usando métricas exactas, debido a que, al no existir criterios generales para calcular distancias, cada punto debe ser evaluado independientemente para determinar si corresponde a uno de estos centros [15]. El primer paso del algoritmo del Capítulo 4, basado parcialmente en esta definición, usa una métrica sencilla para detectar estos centros de manera aproximada.

1.1.3. Propiedades del *skeleton*

Se ha demostrado que las distintas definiciones del *skeleton* son equivalentes en teoría; es decir, en conjunto sugieren la existencia de un *skeleton* analítico [96]. Sin embargo, llevar las definiciones a la práctica significa crear algoritmos fundamentalmente distintos entre sí. Aún más, como se comprobará a lo largo de esta tesis, dos algoritmos que adopten la misma definición como punto de

partida pueden producir *skeletons* muy diferentes. Una de las razones para estas diferencias es que las figuras son representadas de manera discreta en el computador, lo cual introduce gran variabilidad en los cálculos [72].

Existe un problema adicional para el caso tridimensional. El *skeleton* de una figura 2D está siempre formado por curvas 1D. En cambio, para figuras 3D es posible definir dos tipos de *skeleton*. En primer lugar está el *skeleton de superficie*, que además de curvas 1D puede contener parches de superficie 2D. El *skeleton* de superficie preserva la geometría general de la figura original. Esto en el sentido de que la figura original podría recuperarse íntegramente a partir de su *skeleton* de superficie [16]. Sin embargo, la estructura del *skeleton* de superficie es difícil de manipular, y la presencia de parches de superficie lo vuelve inadecuado para muchas aplicaciones [42]. Por otra parte, el *skeleton curvilíneo* es una simplificación del *skeleton* de superficie formada exclusivamente por curvas 1D, y es de mayor utilidad práctica [42]. La Figura 1.5 muestra los *skeletons* de superficie y curvilíneo para una volumen 3D.

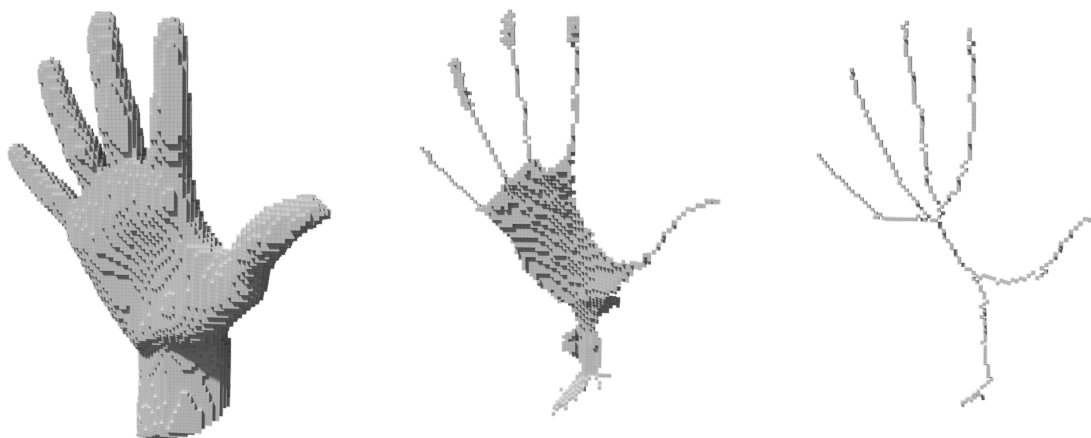


Figura 1.5: A la izquierda, una figura 3D. Al centro, su *skeleton* de superficie, y a la derecha su *skeleton* curvilíneo.

Existen algoritmos que calculan el *skeleton* de superficie, otros que calculan el *skeleton* curvilíneo a partir del *skeleton* de superficie y otros que calculan directamente el *skeleton* curvilíneo. La mayoría de los algoritmos basan su diseño en alguna de las definiciones de la sección anterior, independiente del tipo de *skeleton* calculado. Como se verá en la Sección 1.2, el *skeleton* curvilíneo es el descriptor requerido en las aplicaciones que motivan esta tesis. Por lo tanto, todos los algoritmos implementados en esta tesis calculan el *skeleton* curvilíneo. Los algoritmos de los Capítulos 2 y 3 calculan el *skeleton*

curvilíneo directamente y el algoritmo del Capítulo 4 lo calcula a partir del *skeleton* de superficie. En adelante, salvo cuando se indique algo distinto, la palabra *skeleton* se usará para referirse al *skeleton* curvilíneo en el caso 3D.

De todo lo anterior se desprende que las definiciones formales del *skeleton* no son suficientes para verificar el resultado de un algoritmo de cálculo del *skeleton*. Más bien, solamente sirven como base para el diseño de un algoritmo. Esto ha llevado a definir propiedades fundamentales que debería satisfacer el *skeleton* calculado por cualquier algoritmo. Observar el cumplimiento de estas propiedades sirve como una validación cualitativa de los resultados de un algoritmo. Dado que algunas propiedades son contradictorias entre sí, algoritmos diferentes las satisfacen en distinta medida. Al mismo tiempo, ciertas aplicaciones favorecen algunas propiedades por sobre otras. Las propiedades más comúnmente encontradas [22,95] se detallan a continuación:

- Homotopía

El *skeleton* debe tener la misma topología que la figura a partir de la cual es calculado. Esto significa que el cálculo del *skeleton* debe preservar el número de componentes conexas en imágenes. En el caso de volúmenes, además de las componentes conexas, el *skeleton* debe tener un túnel por cada túnel y cavidad del volumen original [53]. Los conceptos de topología aquí mencionados son retomados con mayor profundidad en la Sección 1.4.1.

- Invariancia

El cálculo del *skeleton* debe ser robusto ante transformaciones isométricas. Por ejemplo, calcular el *skeleton* de una figura y luego rotar el *skeleton* calculado debería dar el mismo resultado que primero rotar la figura y luego calcular su *skeleton*.

- Centralidad

El *skeleton* debe estar centrado en la figura original. Es decir, debe quedar ubicado en los centros de los ejes y planos de simetría de la figura.

- Delgadez

El *skeleton* de una figura de n dimensiones debe ser una figura cuyas partes tienen a lo más $n - 1$ dimensiones. El *skeleton* se puede definir para un número de dimensiones arbitrario [53], pero para esta tesis, puesto que concierne figuras representadas visualmente en el computador, $n \in \{2, 3\}$, y los *skeletons* tanto de figuras como de volúmenes deben estar formados por curvas 1D.

- Suavidad

Algunas aplicaciones requieren que el *skeleton* no tenga saltos bruscos [104]. En teoría, el *skeleton* debería estar formado por curvas de clase \mathcal{C}^2 , es decir, curvas continuas hasta su segunda derivada. Sin embargo, suavizar el *skeleton* en este sentido va siempre en detrimento de la propiedad de centralidad.

- Robustez

El cálculo del *skeleton* debería ser resistente a perturbaciones pequeñas en el borde de las figuras. En otras palabras, el *skeleton* de una figura debería asemejarse al *skeleton* de esa misma figura con algo de ruido. Esta propiedad es deseable, puesto que un *skeleton* estricto es muy sensible al ruido, lo cual reduce su valor expresivo.

- Reconstructibilidad

El cálculo del *skeleton* debería ser reversible. La figura original debería poder ser reconstruída a partir de los puntos del *skeleton* y los radios de sus esferas maximales inscritas correspondientes. Esto es particularmente deseable en aplicaciones donde el *skeleton* se usa para la compresión de datos [43,52]. Sin embargo, la propiedad de robustez, contradictoria con la reconstructibilidad, es frecuentemente favorecida [22].

1.2. Aplicaciones del *skeleton*

En esta sección se describen algunas de las aplicaciones más relevantes para el *skeleton*, principalmente en contextos cercanos a la Biología.

1.2.1. Aplicaciones en Biología

Las aplicaciones del *skeleton* que motivan esta tesis están enmarcadas en el estudio de dos estructuras biológicas: la neurona y el retículo endoplasmático. La Figura 1.6 muestra una neurona, reconstruída como un volumen binario a partir de imágenes capturadas con un microscopio confocal. Este tipo de microscopio permite obtener imágenes de una muestra capa por capa. Luego estas imágenes son segmentadas manualmente, y al ser apiladas forman un volumen como el mostrado en la Figura 1.6. La Figura 1.7 muestra un retículo endoplasmático segmentado manualmente a partir de una imagen del microscopio. El retículo endoplasmático mostrado es prácticamente plano, por lo que se visualiza íntegramente en una imagen, pero otras muestras de retículos podrían requerir una reconstrucción tridimensional.

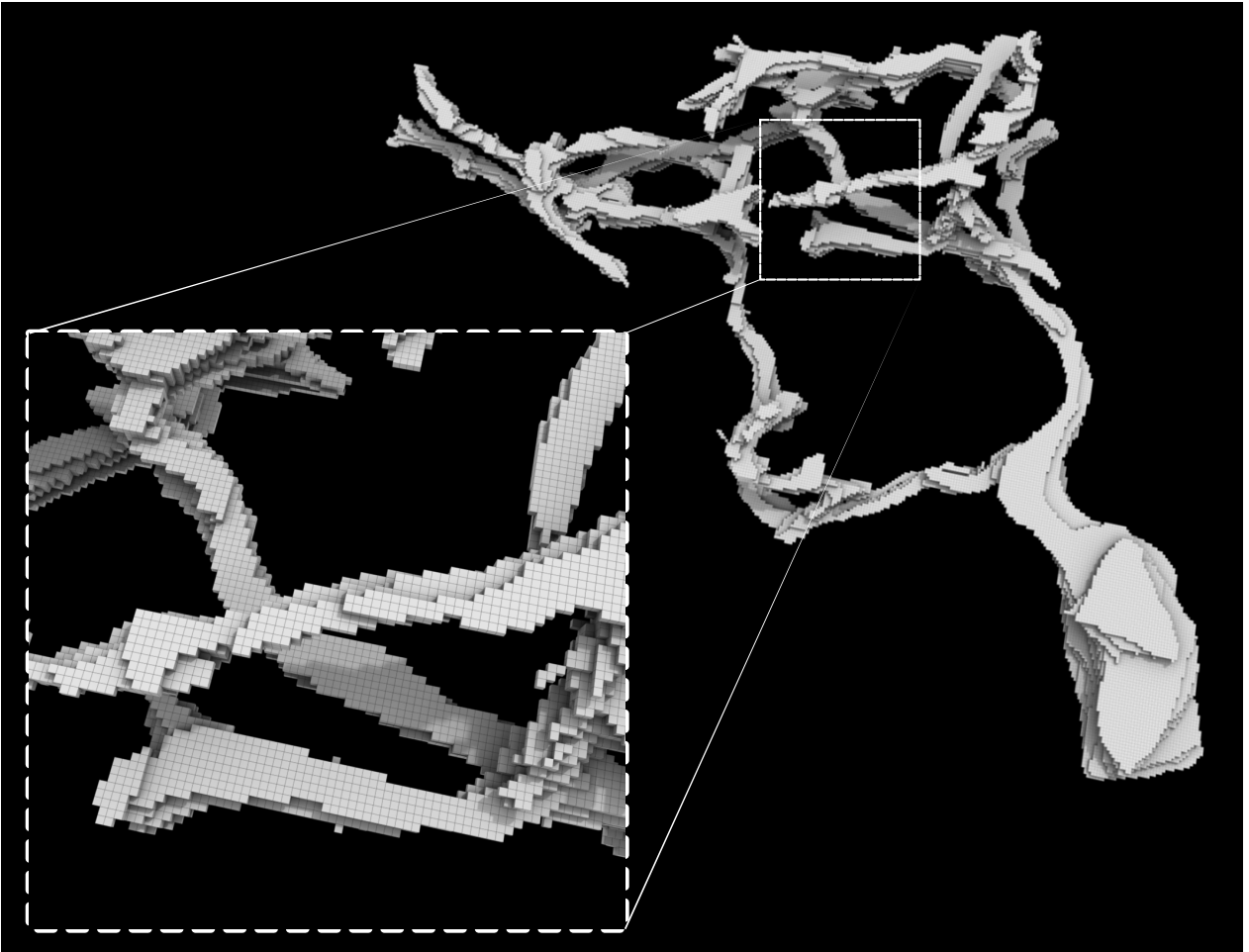


Figura 1.6: Volumen segmentado de una neurona de pez cebra (diencéfalo dorsal, órgano parapineal) marcada con proteína fluorescente mCherry y obtenida mediante microscopía confocal de disco rotatorio.

La complejidad de estas estructuras dificulta el estudio de sus formas sin herramientas computacionales que permitan cuantificarlas [60, 68]. Entre estas herramientas está el *skeleton*, cuya unidimensionalidad y simplicidad lo vuelven ideal para realizar distintos tipos de mediciones a estas estructuras. En general, en Biología se le llama *análisis morfológico cuantitativo* a la caracterización de una estructura biológica mediante números.

En el análisis morfológico cuantitativo de neuronas, es común que los largos de los segmentos del *skeleton* se utilicen como indicador del tamaño de la neurona [40, 80]. Se extraen también otros indicadores a partir del *skeleton*, como por ejemplo el conteo de puntos de término [40] y puntos de ramificación [13, 64] con el objetivo de medir el crecimiento de la neurona. También resulta útil visualizar el *skeleton* de la neurona para analizar interactivamente su forma [56].

El caso de los retículos endoplasmáticos es similar. Estudiar la forma del retículo endoplasmático tiene como principal objetivo entender la relación existente entre su forma y su función [106]. Para

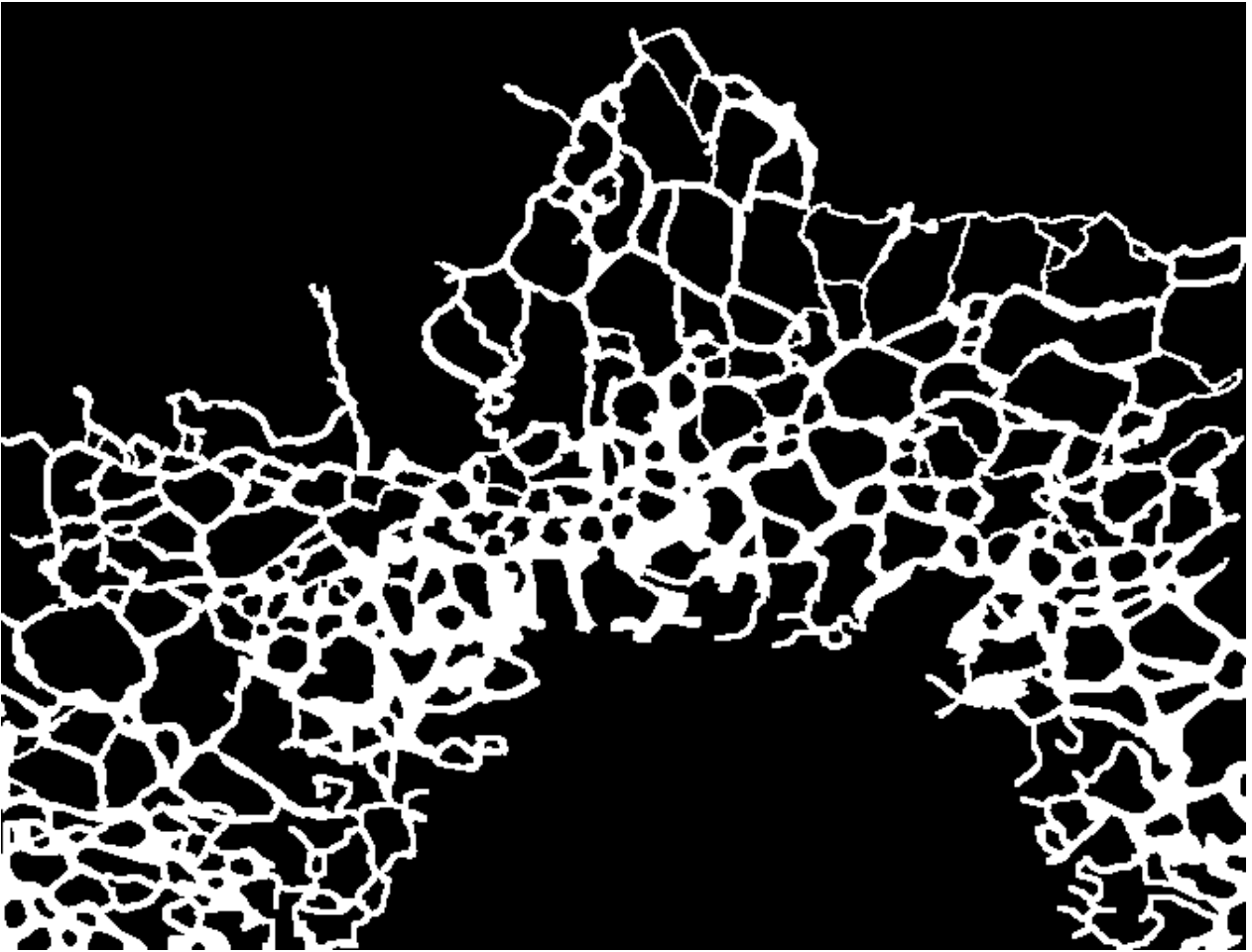


Figura 1.7: Imagen segmentada del retículo endoplasmático de una célula COS-7 (célula epitelial de mono verde) marcada mediante transfección transiente con la proteína KDEL-mNeonGreen obtenida mediante microscopía confocal de disco rotatorio.

esto es de particular interés la distribución de largos de los segmentos del retículo, que se calcula, como en el caso de las neuronas, midiendo los largos de los segmentos del *skeleton* [68, 69, 105].

1.2.2. Otras aplicaciones

Las aplicaciones del *skeleton* en el análisis morfológico cuantitativo no se limitan al estudio de neuronas y retículos endoplasmáticos. En el campo de la medicina, la cuantificación del *skeleton* asiste en el diagnóstico de enfermedades, siendo particularmente relevante en el diagnóstico de estenosis (el estrechamiento de algún conducto, como un vaso sanguíneo). Por ejemplo, calcular el *skeleton* es un paso importante para identificar regiones donde se reduce el diámetro de vasos sanguíneos en angiogramas [21, 35, 79] y tomografías de tráquea [90]. Siguiendo la misma idea, Zwiggellar et al. [107] clasifican los segmentos del *skeleton* según el diámetro de la sección correspondiente para detectar anomalías en mamogramas. En trabajos como el de Selle et al. [81] se calcula el *skeleton* de angiografías de vasos sanguíneos hepáticos para visualizarlos previo a procedimientos quirúrgicos.

Otra aplicación médica del *skeleton* es la detección de osteoporosis. El hueso esponjoso es un tejido poroso, por lo que en una imagen de resonancia magnética parece una red. El problema de caracterizar esta red es similar al de cuantificar la estructura del retículo endoplasmático [66]. En este caso, una baja conectividad indica deterioro del tejido.

Una aplicación donde la suavidad del *skeleton* cobra especial relevancia es la endoscopía virtual [38]. Esta técnica consiste en construir un modelo tridimensional de un órgano del paciente usando un escáner. Visualizando este modelo, el médico puede examinar el órgano de manera no invasiva en una simulación, donde la cámara pasa por el interior del órgano montada en un riel ubicado en su eje central. En este contexto, el *skeleton* del modelo se calcula para posicionar el riel que debe recorrer la cámara [30, 65]. El *skeleton* calculado debe ser suave, para evitar que en la navegación virtual haya movimientos demasiado bruscos [22]. La Figura 1.8 muestra varias etapas de una broncoscopia virtual y el *skeleton* correspondiente.

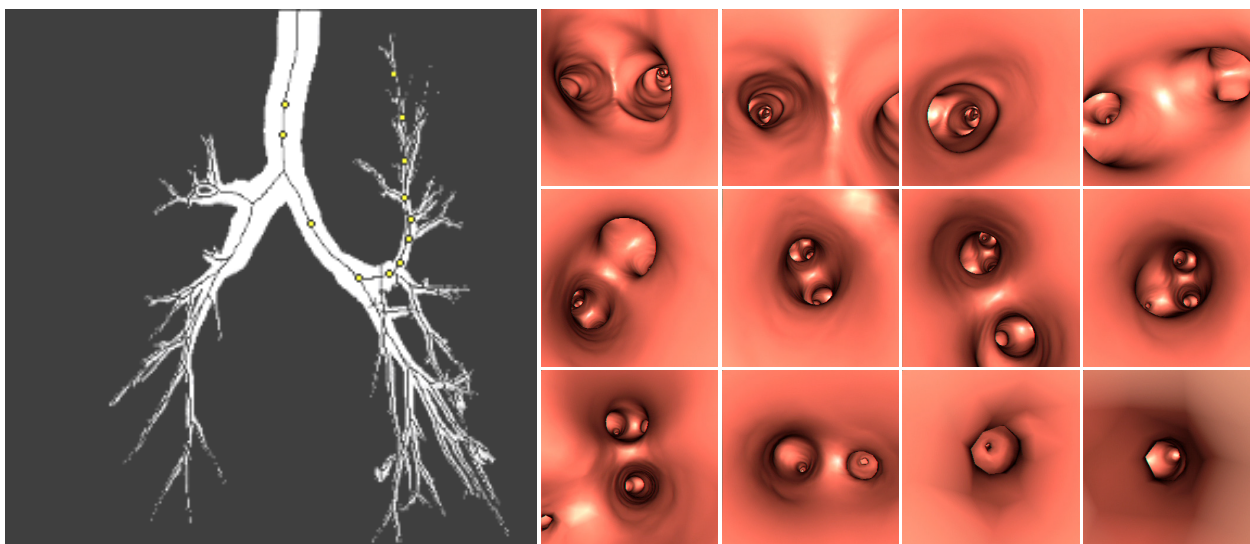


Figura 1.8: El *skeleton* en la broncoscopia virtual [65]. A la izquierda, la proyección del *skeleton* calculado a partir de la reconstrucción digital de los bronquios. A la derecha, capturas de la exploración virtual a través de la visualización del modelo digital.

Otra aplicación importante se encuentra en la animación digital [11]. En este contexto, se conoce como *IK skeleton* o *skeleton de control* a una estructura de datos que permite animar el movimiento de un modelo 3D. Se trata de una jerarquía de segmentos y articulaciones, junto la información de anclaje para vincular cada una de estas componentes a distintas partes del modelo. Por medio de este anclaje, mover las componentes del *IK skeleton* permite mover el modelo. La Figura 1.9 ilustra el *skeleton*, su deformación y su aplicación como *IK skeleton* de un modelo. Muchos programas de animación permiten construir manualmente el *IK skeleton*, pero esta construcción es un trabajo laborioso. Para subsanar este problema, se han propuesto métodos que calculan automáticamente

el *IK skeleton* a partir del *skeleton* del modelo [31, 103].

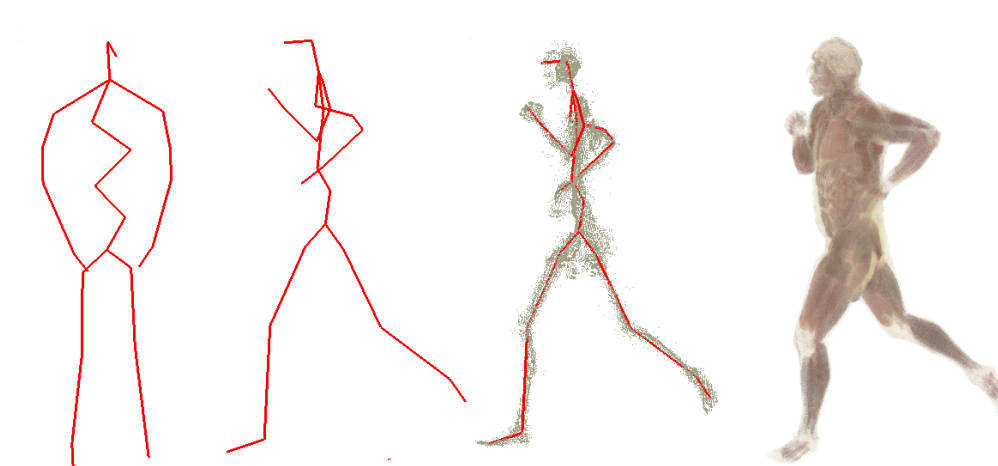


Figura 1.9: El *IK skeleton* usado para deformar un modelo, alterando su pose original (a la izquierda) por una pose de carrera (a la derecha) [31].

Hasta aquí solamente se han revisado ciertas aplicaciones del *skeleton*, abarcando algunas de las más cercanas a la Biología. Aplicaciones adicionales pueden encontrarse en los diversos trabajos de revisión en el área [22, 76, 87]. El *skeleton* también es un descriptor de amplia utilización en el reconocimiento de caracteres [2], análisis de huellas digitales [47], descomposición de figuras [82], compresión [52], captura de movimiento [70], búsqueda por forma [92] e interpolación de imágenes [20].

1.3. Representación discreta de volúmenes

El objetivo de esta sección es presentar al lector los conceptos y notaciones que se usarán en los capítulos posteriores. Esta revisión comprende los casos 2D y 3D en igual medida. Se incluye en la revisión conceptos 2D, si bien las implementaciones son en 3D, por completitud y para facilitar la explicación y comprensión de sus contrapartes tridimensionales.

1.3.1. Estructura de datos

Los algoritmos implementados en esta tesis operan sobre representaciones binarias discretas. Esto quiere decir que su entrada y salida son matrices donde cada elemento vale 0 o 1. Si esta matriz es 2D, se llama *imagen*. Si es 3D, se llama *volumen*.

Lo anterior significa que las figuras son construidas a partir de un muestreo binario. La manera de efectuar este muestreo en la práctica (el proceso llamado *segmentación*, donde se decide si un punto pertenece o no a la figura) está fuera del alcance de esta tesis, pero en el Capítulo 5 se describe

cómo se realiza para la aplicación de Biología que motiva este trabajo de tesis.

El *objeto* de una imagen o un volumen es el conjunto de sus elementos que valen 1. El *fondo* es el conjunto de sus elementos que valen 0. El *skeleton* es el objeto de la imagen o volumen que un algoritmo produce como salida.

Los elementos que constituyen una imagen se llaman *píxeles*. En el caso de un volumen, se llaman *vóxeles*. Los *píxeles de objeto* son los píxeles que pertenecen al objeto; es decir, que valen 1. Los *píxeles de fondo* son los píxeles que pertenecen al fondo, cuyo valor es 0. Los *vóxeles de objeto* y *vóxeles de fondo* se definen respectivamente de manera análoga.

1.3.2. Vecindad y Conectividad

Esta parte trata sobre las relaciones de adyacencia entre píxeles y vóxeles. El objetivo es precisar qué significa que el *skeleton* conserve la topología de la figura original.

Vecindad y conectividad 2D

En una imagen, cada píxel p tiene exactamente 8 vecinos. Estos son los píxeles que rodean a p . Interpretando la imagen como una grilla, donde cada píxel es un cuadrado, los vecinos de p son los píxeles que comparten al menos un vértice con p . El conjunto de píxeles formado por p y sus 8 vecinos se denomina la *8-vecindad de p* . Es denotado $V_8(p)$. Además, se dice que p es un *8-vecino* de los demás elementos de $V_8(p)$.

Un tipo de adyacencia más estricto considera solo los vecinos de p que comparten una arista (o, equivalentemente, dos vértices) con p . Desde un punto de vista matricial, estos son los vecinos de p que están en la misma fila o columna que p . Exactamente 4 píxeles satisfacen esta condición. Entonces, se dice que estos píxeles son *4-vecinos de p* . Unidos a p , forman la *4-vecindad de p* , denotada $V_4(p)$.

Es claro que $V_4(p) \subset V_8(p)$. Por consiguiente, si dos píxeles son 4-vecinos también son 8-vecinos. Además, todas las relaciones de vecindad son conmutativas. Si p es n -vecino de q , entonces q es n -vecino de p . La Figura 1.10 ilustra las vecindades de un píxel.

Un *8-camino* es una sucesión de píxeles (p_0, p_1, \dots, p_k) que satisface dos condiciones. En primer lugar, todos sus píxeles son de objeto. En segundo lugar, cada uno de ellos, salvo el último, es 8-vecino del siguiente. Luego, se dice que dos píxeles p y q están *8-conectados* si existe un 8-camino que empieza en p y termina en q . Análogamente, dos píxeles p y q están *4-conectados* si existe un *4-camino* que empieza en p y termina en q .

Vecindad y conectividad 3D

Extendiendo la noción de grilla a 3D, en un volumen cada vóxel se puede interpretar como un cubo dentro de una grilla tridimensional. Entonces, cualquier vóxel v puede ser ubicado en el centro de una subgrilla de $3 \times 3 \times 3$ vóxeles. El conjunto de vóxeles que pertenecen a esta subgrilla se denomina la *26-vecindad de v* , porque en ella 26 vóxeles rodean a v . Se denota $V_{26}(v)$ y se dice que v es *26-vecino* de los demás elementos de $V_{26}(v)$. Todos los vóxeles de este conjunto, y solo ellos, comparten al menos un vértice con v dentro de la grilla.

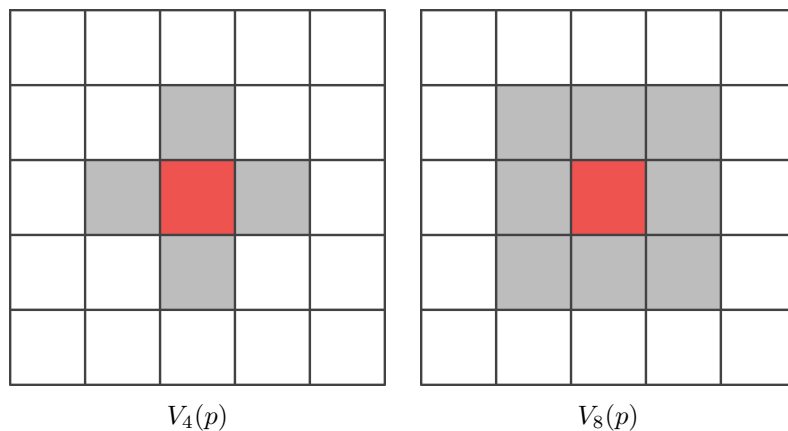


Figura 1.10: Vecindades de un píxel p .

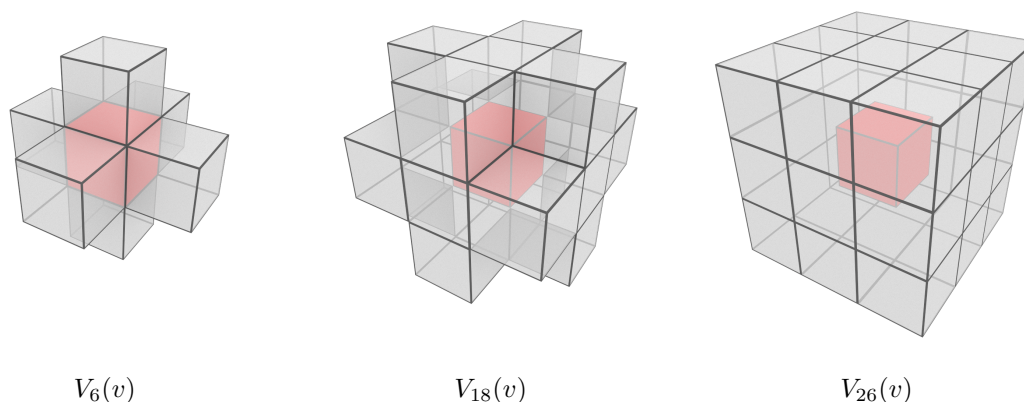


Figura 1.11: Vecindades de un vóxel v .

Un subconjunto importante de $V_{26}(v)$ son los vóxeles que comparten una cara con v . Como cada cubo de la grilla tiene 6 caras, existen 6 vóxeles con esta característica. El conjunto formado por estos vóxeles y v se denomina la *6-vecindad de v* , y se denota $V_6(v)$. Se dice que v es *6-vecino* de los demás vóxeles de $V_6(v)$.

Existe además un tercer tipo de adyacencia dentro de $V_{26}(v)$: los vóxeles que comparten alguna arista (o, equivalentemente, por lo menos dos vértices) con v . Esta condición significa excluir a los

vecinos de v que comparten exactamente 1 vértice con v . Gráficamente, quedan fuera los 8 vóxeles que ocupan las esquinas de la subgrilla de $3 \times 3 \times 3$ centrada en v . El conjunto conformado por v y sus 18 vecinos restantes se denomina la *18-vecindad de v* . Es denotado $V_{18}(v)$. Se dice que v es *18-vecino* de los demás elementos de este conjunto.

Es conveniente destacar que $V_6(v) \subset V_{18}(v) \subset V_{26}(v)$. Por lo tanto, si dos vóxeles v y w son 6-vecinos, también son 18-vecinos y 26-vecinos. La Figura 1.11 ilustra las vecindades de un vóxel.

Al igual que para imágenes, se define un tipo de camino por cada tipo de vecindad en un volumen. Así, en volúmenes la definición de camino de la sección 1.3.2 se extiende para vóxeles, dando origen a *26-caminos*, *18-caminos* y *6-caminos*.

Se dice que dos vóxeles v y w están *26-conectados* si existe un 26-camino que empieza en v y termina en w . De manera análoga, dos vóxeles pueden estar *18-conectados* y *6-conectados*.

1.4. Componentes conexas

Una *componente n -conexa*, con $n \in \{4, 8\}$ si pertenece a una imagen y $n \in \{6, 18, 26\}$ si pertenece a un volumen, es un conjunto no vacío de píxeles o vóxeles que satisface dos condiciones. En primer lugar, cada elemento del conjunto está n -conectado con todos los demás. En segundo lugar, ningún elemento del conjunto está n -conectado con algún elemento fuera del conjunto. Es decir que, usando como ejemplo el caso 3D, la componente n -conexa que contiene a un vóxel v es el conjunto maximal de vóxeles n -conectados a v . Si un conjunto no es n -conexo, se dice que es *n -disconexo*.

En la Sección 1.1.3 se ha mencionado como requisito para la correctitud de un algoritmo el que la imagen de salida sea homotópica con la imagen de entrada. En la práctica, esta preservación topológica se traduce en que la salida debe tener el mismo número de componentes conexas de objeto y de fondo que la entrada [78]. Sin embargo, estos números dependen del tipo de conectividad que se escoja. Por ejemplo, la imagen de la Figura 1.12 tiene 1 componente conexa de objeto y 1 componente conexa de fondo, independientemente de si se verifica la 4-conectividad o la 8-conectividad. Cambiar el valor del píxel marcado a 0 no altera el número de componentes 8-conexas, pero sí el de 4-conexas. Tras el cambio, la imagen quedaría con 4 componentes 4-conexas de objeto y 2 componentes 4-conexas de fondo. Por lo tanto, un algoritmo que cambiara el valor del píxel rojo a 0 cumpliría con la propiedad homotópica si se usa la 8-conectividad y no la cumpliría si se usa la 4-conectividad.

La paradoja del fondo

Casos como el ilustrado en la Figura 1.12 hacen necesario convenir cuál tipo de conectividad se usará. Una posibilidad sería fijar la 8-conectividad para imágenes y la 26-conectividad para volú-

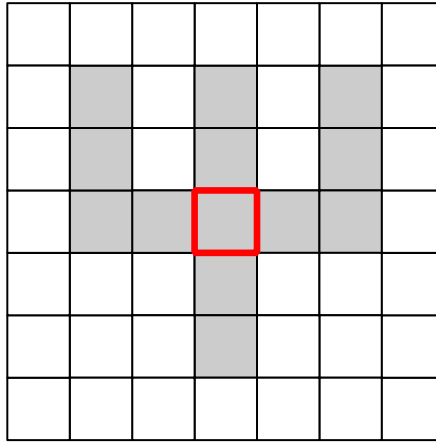


Figura 1.12: Removiendo el píxel marcado, objeto y fondo se mantienen 8-conexos, pero ambos se vuelven 4-disconexos.

menes, argumentando que esto permitiría mayor libertad en la forma del *skeleton*. Así, el *skeleton* podría reproducir la forma del objeto original con mayor fidelidad.

Sin embargo, la Figura 1.13 ilustra un problema perceptual que resulta de elegir cualquier conectividad, descrita primero por Duda et al. [28]. En la imagen de la figura, tanto fondo como objeto son componentes 8-conexas, cuando la intuición sugiere que el objeto, un “diamante” 8-conexo, debería dividir al fondo en un “interior” y un “exterior”. De examinarse la 4-conectividad, el diamante, siendo totalmente 4-disconexo, sí separaría al fondo en 2 componentes 4-conexas. El problema persiste en 3D. Basta pensar en una imagen donde el objeto sea el conjunto $V_6(v) - \{v\}$, para algún vóxel v fijo. En esa imagen, tanto objeto como fondo serían 26-conexos, 18-conexos y 6-disconexos a la vez. Sin tratarse de un problema matemático, el que visualmente una componente disconexa separe componentes conexas, y viceversa, dificulta la comprensión de la topología digital [74].

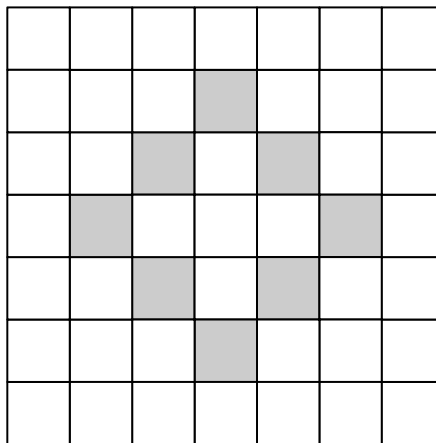


Figura 1.13: Imagen donde objeto y fondo son 8-conexos y 4-disconexos al mismo tiempo.

La ambigüedad desaparece si se escoge algún tipo de conectividad para el objeto y uno diferente

para el fondo. Para esta tesis se adopta la convención más común [50]. Se usará la 8-conectividad para el objeto y la 4-conectividad para el fondo en imágenes. En volúmenes, se usará la 26-conectividad para el objeto y la 6-conectividad para el fondo. Siguiendo esta convención, es posible afirmar que en la Figura 1.13 el diamante conexo efectivamente divide al fondo en dos componentes conexas.

Búsqueda de componentes conexas

Recorrer alguna componente conexa es un paso común dentro de los algoritmos implementados en esta tesis. Por lo general no es requisito etiquetar ni contar todas las componentes de una imagen (para lo cual existen algoritmos más apropiados [98]), sino recorrer alguna componente en particular a partir de un píxel o vóxel determinado. Esto se consigue en tiempo lineal adaptando algún algoritmo de búsqueda en grafos para que recorra elementos de imágenes en lugar de nodos [102]. La elección del algoritmo es arbitraria, porque el orden del recorrido no es relevante.

Algoritmo 1 Recorrer una componente conexa por búsqueda en profundidad

```

1: function RECORRERCCPORBEP( $I, p, n$ )
2:    $s \leftarrow \text{stack}()$ 
3:    $s.\text{push}(p)$ 
4:   while  $\neg s.\text{empty}()$  do
5:      $q \leftarrow s.\text{pop}()$ 
6:     if  $q$  no ha sido marcado como visitado then
7:       marcar  $q$  como visitado      ▷ Por lo general se hace algo con  $q$  además de marcarlo
8:       for all  $r \in V_n(q) - \{q\}$  do
9:         if  $\text{val}(q) = \text{val}(r)$  then      ▷  $\text{val}(x)$  es el valor de  $x$  en la imagen, 0 o 1
10:           $s.\text{push}(r)$ 
11:        end if
12:      end for
13:    end if
14:  end while
15: end function

```

El Algoritmo 1, implementado en esta tesis, corresponde a la búsqueda en profundidad [41] adaptada a representaciones discretas. En él, se comienza por ingresar en una pila todos los n -vecinos n -conectados a un nodo inicial p . Este proceso se repite recursivamente hasta vaciar la pila. La implementación iterativa para la recursión superó en eficiencia al uso de llamadas recursivas. Cabe señalar que la conectividad n se podría inferir a partir del número de dimensiones de la entrada y de si p es de objeto o de fondo, según la convención descrita en la sección anterior. Sin embargo, n se explicita porque en ocasiones se querrán recorrer componentes conexas sin necesariamente obedecer esa convención.

1.4.1. Criterios de preservación topológica

Un píxel o vóxel de objeto es *simple* si cambiar su valor a 0 (o *eliminarlo*) no altera la topología de la imagen o volumen al que pertenece. Para saber si un elemento es simple, se podría calcular el número de componentes conexas antes y después de eliminarlo, usando por ejemplo el Algoritmo 1, y revertir la eliminación si ese número cambia. Sin embargo, existen métodos más eficientes, capaces de determinar si un elemento es simple revisando nada más que su 26-vecindad. A continuación se describen los métodos implementados para esta tesis.

Simplicidad en 2D

Precisando lo señalado anteriormente, un píxel de objeto es simple si eliminarlo no cambia el número de componentes 8-conexas de objeto ni el número de componentes 4-conexas de fondo. En este sentido, si un algoritmo elimina solamente píxeles simples efectivamente preserva la topología de la imagen [91]. En otras palabras, un algoritmo que solamente elimine píxeles simples siempre cumplirá con la propiedad homotópica.

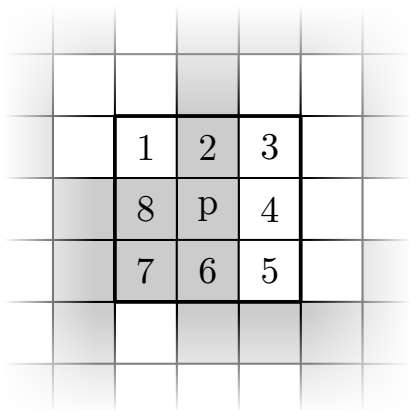


Figura 1.14: Numeración de la vecindad de p para construir su grafo de vecindad.

El criterio para determinar la simplicidad en imágenes implementado en esta tesis es el explicado en el artículo de Siddiqi et al. [86]. Dado un píxel p , se numera su 8-vecindad como se muestra en la Figura 1.14. Nótese que p no se numera. El *grafo de vecindad de p* se construye situando primero un nodo sobre cada píxel de objeto numerado y luego una arista entre cada par de nodos situados sobre píxeles 8-vecinos entre sí. Si alguna de las 3-tuplas $(2, 3, 4)$, $(4, 5, 6)$, $(6, 7, 8)$ u $(8, 1, 2)$ corresponde a nodos del grafo, se elimina la respectiva arista diagonal $(2, 4)$, $(4, 6)$, $(6, 8)$ u $(8, 2)$. Con esto se remueven los posibles ciclos de largo 3.

Examinar las relaciones de adyacencia entre los nodos del grafo de vecindad de p indica qué sucedería al eliminar p . Si la eliminación de p desconectara el objeto, el grafo de vecindad de p

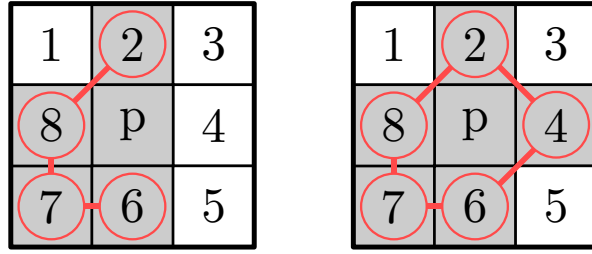


Figura 1.15: Grafos de vecindad para dos píxeles distintos. En ambos casos la arista que conectaba los nodos 6 y 8 ha sido removida.

sería desconexo. Si la eliminación de p originara una componente de fondo (un “agujero”), el grafo de vecindad de p tendría un ciclo. Un grafo conexo sin ciclos es un árbol. Por lo tanto, se cumple el siguiente lema:

Lema 1. *Un píxel p es simple si su grafo de vecindad es un árbol.*

Un criterio directo para determinar si el grafo de vecindad es un árbol consiste en comprobar si su característica de Euler, el número de vértices menos el número de aristas, es igual a 1. La Figura 1.15 muestra ejemplos donde p es simple y donde no.

Simplicidad en 3D

Además de componentes conexas, los volúmenes pueden presentar un elemento topológico adicional. La Figura 1.16 ilustra un ejemplo. Difíciles de definir y encontrar [93], los *túneles* también deben considerarse al evaluar la preservación topológica de un algoritmo [22]. Informalmente, un túnel es un agujero que atraviesa un objeto sin desconectarlo. Una definición formal de túnel fue propuesta por Kong y Rosenfeld [49], pero no es necesaria para esta tesis. Solo es preciso considerar que la eliminación de un vóxel puede originar un túnel, alterando con ello la topología de la imagen sin cambiar el número de componentes conexas. Adicionalmente, la unidimensionalidad del *skeleton* no le permite preservar las *cavidades*, que son componentes 6-conexas de fondo completamente rodeadas por una componente 26-conexa de objeto.

Tomando en cuenta todo lo anterior, se dice que un algoritmo satisface la propiedad homotópica si el *skeleton* que calcula preserva el número de componentes 26-conexas de objeto, el número de componentes 6-conexas de fondo y además tiene un túnel por cada túnel y por cada cavidad del volumen original [53].

Se han propuesto varios criterios para determinar la simplicidad de un vóxel [49, 59, 100]. En

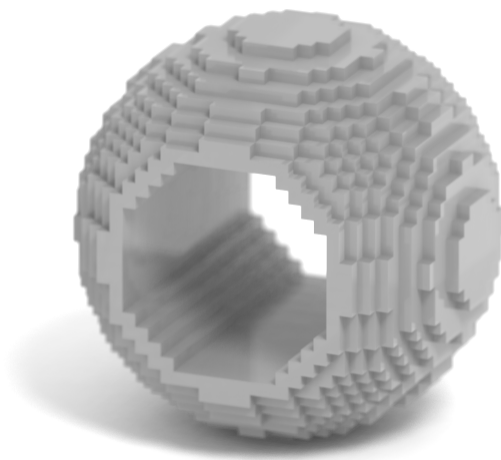


Figura 1.16: Ejemplo de volumen con un túnel.

esta tesis se utiliza la caracterización provista por Bertrand y Malandain [12]. Esta caracterización comienza por definir dos números:

- $c(v)$ como el número de componentes 26-conexas de objeto en $V_{26}(v) - \{v\}$ y
- $c^*(v)$ como el número de componentes 6-conexas de fondo en $V_{18}(v) - \{v\}$ con algún vóxel 6-vecino a v .

Luego, se cumple el siguiente teorema:

Teorema 1. *Un vóxel v es simple si y solo si $c(v) = 1$ y $c^*(v) = 1$.*

Tener en cuenta la noción de punto simple al diseñar un algoritmo basta para satisfacer la propiedad homotópica. Como se ha dicho antes, la topología podría conservarse imponiendo la restricción de eliminar exclusivamente puntos simples. Sin embargo, esta estrategia no asegura en ninguna medida el resto de las propiedades del *skeleton*. Los píxeles de los extremos de una línea, por ejemplo, son simples, pero eliminarlos significaría perder información de la forma de la figura (es más, la línea podría ser transformada en un punto aplicando sucesivamente este criterio).

Píxeles y vóxeles de término

Un *píxel de término* es un píxel de objeto que tiene exactamente 1 píxel de objeto 8-vecino. Similarmente, un *vóxel de término* es un vóxel de objeto que tiene exactamente 1 vóxel de objeto 26-vecino. La Figura 1.17 ilustra los principales tipos de vóxeles.

El que un elemento sea de término significa que está en el extremo de una línea. En principio, todas las líneas son necesarias para que el *skeleton* exprese la forma del objeto original con fidelidad, por lo que un algoritmo debería preservar todos los elementos de término. No obstante, algunas líneas

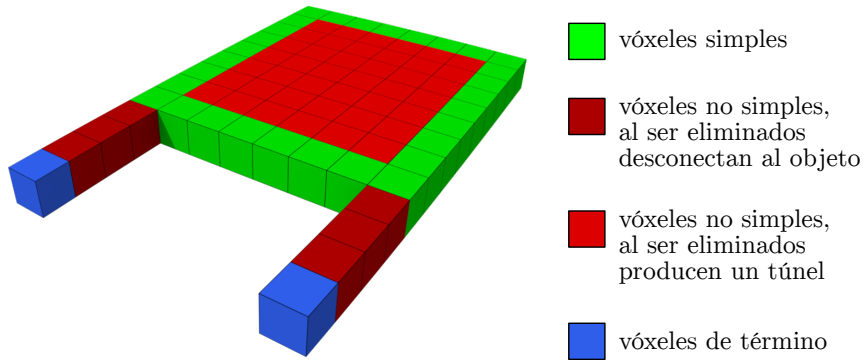


Figura 1.17: Principales tipos de vóxeles.

pueden ser poco relevantes, por lo que todo algoritmo debe obedecer algún criterio para decidir si un elemento de término debe ser eliminado. En cuán relajado sea este criterio radica en qué medida el resultado producido por un algoritmo favorecerá la propiedad de reconstructibilidad (en perjuicio de otras propiedades, como la suavidad y la robustez).

1.5. Clases de algoritmos de cálculo del *skeleton*

En principio, los algoritmos de cálculo del *skeleton* pueden dividirse en dos, según el tipo de dato que reciben como entrada. Están los algoritmos que operan sobre mallas geométricas y los que operan sobre volúmenes de vóxeles. Para esta tesis se implementaron exclusivamente algoritmos que operan sobre volúmenes de vóxeles, que pueden subdividirse en tres clases [22, 76, 88, 96]. Cada algoritmo implementado en esta tesis es representativo de alguna de las tres clases descritas a continuación. Por completitud, se agregan en una cuarta clase los algoritmos que operan sobre mallas geométricas.

1.5.1. Algoritmos de adelgazamiento topológico

El enfoque más intuitivo y divulgado para este problema usa la definición de punto simple para erosionar progresivamente el borde de un objeto de vóxeles [51, 59, 78], capa por capa, hasta lograr el ancho unitario [5, 16, 39, 55]. Abordar el problema de esta forma es cercano a simular la transformada del incendio propuesta por Blum. Con el fin de reducir el costo computacional, también se ha propuesto hacer la eliminación de vóxeles en paralelo [34, 63, 77].

En el Capítulo 2 se detalla la implementación del algoritmo de adelgazamiento paralelo de Palágyi y Kuba [63].

1.5.2. Algoritmos basados en la distancia

Una manera alternativa de adelgazar la figura es incorporar en el cálculo información de la distancia de cada vóxel al borde [67]. Siguiendo este enfoque, existen varios algoritmos que buscan cimas en la transformada de distancia para calcular el *skeleton*. Esto equivale a encontrar los centros de las esferas maximales inscritas [26, 65, 67].

Por lo general estos algoritmos calculan primero el *skeleton* de superficie para el caso 3D. Luego, el *skeleton* curvilíneo es calculado como un subconjunto centrado en el *skeleton* de superficie. Esto les permite un mayor grado de centralidad en comparación con las demás clases de algoritmos [95]. El algoritmo de Arcelli et al [8], cuya implementación es detallada en el Capítulo 4, es el representante implementado de esta clase.

1.5.3. Algoritmos de campos generales

Otros algoritmos usan campos más elaborados, generalmente construidos a partir de la transformada de distancia. Esto permite calcular el *skeleton* a partir de un número menor de singularidades en comparación a las cimas de la transformada de distancia. El libro de Siddiqi et al. [87] proporciona una buena recopilación de algoritmos de esta categoría. Se ha calculado el *skeleton* usando varios campos, tanto de orden escalar [3, 33] como vectorial [23, 37, 85].

Para esta tesis se implementó el algoritmo de Siddiqi et al. [86], uno de los primeros algoritmos de campos generales propuestos [88]. Este algoritmo se basa en el laplaciano de la transformada de distancia para detectar los puntos de choque, siguiendo principios similares a los expuestos en los trabajos de Kimmel y Leymarie [29, 48]. Los detalles se exponen en el Capítulo 3.

1.5.4. Algoritmos para representaciones geométricas

Como se mencionó anteriormente, el *skeleton* también se puede calcular a partir de representaciones geométricas. Ejemplos de esto son el uso de diagramas de Voronoi para extraer *skeletons* 2D [54, 61, 62] y 3D [25, 83], colapso de mallas geométricas [9, 46] y otras operaciones de centralidad [19, 42].

Aunque no se implementó un representante de esta clase de algoritmos para esta tesis, se incluye en la comparación la implementación del algoritmo de Au et. al [9] fruto de los trabajos de memoria de Liliana Alcayaga e Iván Rojas [4, 73].

La clasificación anterior no es estricta, puesto que varios algoritmos rescatan elementos de más de una clase. Por ejemplo, el algoritmo del Capítulo 4 usa adelgazamiento para asegurar la delgadez del *skeleton*. Sin embargo, el objetivo de este trabajo es proponer una evaluación cuantitativa de

propósito general. Con esto en mente, implementar un método de cada clase busca ilustrar los alcances generales de las métricas de comparación propuestas.

1.6. Comparaciones de algoritmos de cálculo del *skeleton*

La evaluación del desempeño de los algoritmos de cálculo del *skeleton* es un desafío que todavía no tiene una solución aceptada [76]. En esta sección se presentan algunos de los esfuerzos realizados con este fin, tanto en Ciencias de la Computación como en la aplicación específica del análisis de neuronas.

| | <i>Thinning</i> | Campo Distancia | Geométricos | Campos Generales |
|--------------------------------|-----------------|--------------------|-------------|---------------------|
| Homotopía | Sí | | Sí | No |
| Invariabilidad ante rotaciones | | Sí | | Sí |
| Reconstructibilidad | No | | No | No |
| Delgadez | | | | Sí |
| Centrado | | | | |
| Confiabilidad | | | | |
| Detección de juntas | | | Sí | Sí |
| Conectividad | Sí | | | |
| Robustez | No | No | No | Sí |
| Suavidad | | | | Sí |
| Jerarquía | No | | | Sí |
| Eficiencia | Sí | Sí | Sí | No |

Tabla 1.1: Propiedades del *skeleton* obtenido por cada clase de algoritmos según Cornea et al. [22] (vacío indica que la propiedad es alcanzable solamente por algunos algoritmos de la clase).

Los tres artículos que se describen a continuación ofrecen una comparación cualitativa para algoritmos de cálculo de *skeleton* 3D, formulada según el cumplimiento de las propiedades descritas en la Sección 1.1.3.

Cornea et al. (2007) [22]

Los autores dividen los algoritmos en clases similares a las señaladas en la Sección 1.5, implementando un algoritmo de cada clase. Luego, aplican los algoritmos a un conjunto de objetos 3D estándar [84] y a figuras reales de colon, describiendo las propiedades de los *skeletons* resultantes. La Tabla 1.1 resume los resultados. La única métrica cuantitativa que se provee es el tiempo de ejecución de los algoritmos. Es importante recalcar que para la publicación de este artículo aún no se habían propuesto los algoritmos de colapso de mallas geométricas.

Sobiecki et al. (2013) [89]

En este artículo se compara 6 algoritmos de colapso de mallas geométricas, señalando esta subclase como la más popular desde la publicación del algoritmo de Au et al. [9]. La comparación es netamente cualitativa, concentrándose en observar las propiedades garantizadas por cada algoritmo. Las conclusiones son interesantes: el desempeño de los algoritmos fue inferior al esperado, motivando el siguiente trabajo.

Sobiecki et al. (2014) [88]

Este artículo se concentra en algoritmos que se aplican a volúmenes de vóxeles. Se comparan, según las mismas propiedades, 6 algoritmos de extracción de *skeleton* curvilíneo y 4 de extracción de *skeleton* de superficie. Esta vez los autores incluyen además el tiempo de ejecución y el máximo de memoria para cada algoritmo. Además, cuantifican la diferencia entre los *skeletons* extraídos a distintas resoluciones.

Estos son los únicos trabajos encontrados de comparación general propuestos desde la Ciencia de la Computación [76]. Iniciativas provenientes de otros campos han comparado algoritmos para aplicaciones más específicas. Ejemplos de esto son el concurso DIADEM [32], donde se definió una métrica para comparar algoritmos de segmentación y extracción de *skeletons* de neuronas, o el criterio de evaluación de algoritmos de extracción de arterias propuesto por Schaap et al. [79]. No obstante, la alta especificidad de las métricas formuladas limita su generalización. Por ejemplo, la métrica DIADEM compara la conectividad del *skeleton* calculado contra un *gold standard* dibujado por biólogos expertos en neuronas. Esto vuelve la métrica difícil de generalizar, pues en general no existen *gold standards* para otros problemas.

Es posible encontrar otras comparaciones en artículos que proponen un algoritmo nuevo de extracción de *skeleton*. Es muy común que los autores de estos artículos comparen sus resultados con los de algoritmos populares o similares, mediante la ilustración gráfica de los *skeletons* calculados a partir de figuras estándar [3, 9, 42, 45, 95, 97]. Este es el enfoque más antiguo para validar estos algoritmos [76]. La Figura 1.18 muestra un ejemplo de esta aproximación.

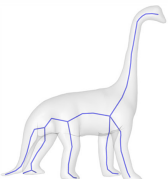
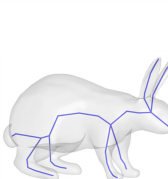

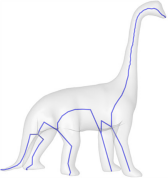
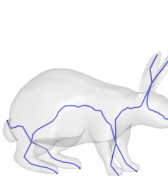

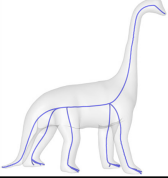
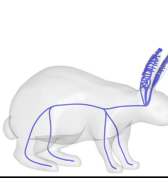

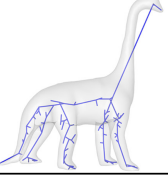
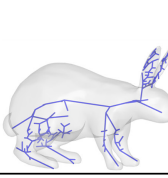

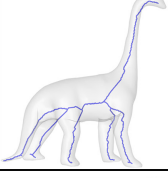
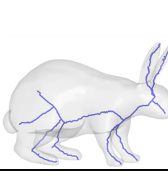

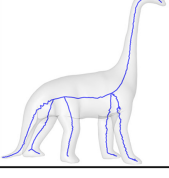
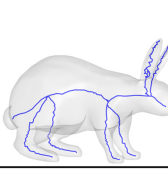

| | | | |
|------------------------|-------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Our Method |  |  |  |
| Reeb Graph |  |  |  |
| Potential Field |  |  |  |
| Distance Field |  |  |  |
| Thinning |  |  |  |
| Medial Surface |  |  |  |

Figura 1.18: Ejemplo de validación gráfica [9].

Por otro lado, parámetros como el tiempo de ejecución o el uso de memoria también suelen usarse como indicadores de desempeño [88]. Sin embargo, estos indicadores no son útiles para cuantificar la calidad del *skeleton* como descriptor de forma.

Finalmente, algunos artículos proponen cuantificar alguna propiedad deseable del *skeleton* para validar su correctitud. Saha et al. [77] proponen una distancia para medir la diferencia entre el *skeleton* de un objeto y el *skeleton* del mismo objeto con ruido y sometido a rotaciones. Arcelli et al. [8] miden el nivel de compresión al reducir un objeto a su *skeleton* y el porcentaje de reconstructibilidad del objeto a partir de su *skeleton*. Así como las propiedades que miden, varias de estas mediciones son incompatibles: reducir la incidencia del ruido (es decir, suavizar el *skeleton*) o comprimirlo atenta contra la capacidad para reconstruir íntegramente el objeto original. En el contexto de esta tesis, las mediciones generales de esta índole son de poca utilidad, puesto que en aplicaciones no constituyen

características de interés.

1.7. Hipótesis y objetivos

La hipótesis de este trabajo de tesis se enuncia a continuación:

Hipótesis Algoritmos de cálculo del *skeleton* diferentes producen resultados significativamente distintos para la misma figura, de acuerdo a métricas relevantes en el estudio morfológico cuantitativo de estructuras biológicas.

Para indagar esta hipótesis, este trabajo de tesis contempla satisfacer los siguientes objetivos:

Objetivo 1 Implementar tres algoritmos de cálculo del *skeleton*, uno por cada una de las siguientes tres clases: adelgazamiento topológico, basados en la distancia y basados en campos generales.

Objetivo 2 Definir e implementar el cálculo de métricas relevantes en Biología que permitan caracterizar numéricamente un *skeleton*.

Objetivo 3 Identificar el algoritmo de cálculo del *skeleton* más apropiado para cada métrica propuesta, mediante la cuantificación de figuras biológicas simuladas y reales.

1.8. Estructura de esta tesis

Los capítulos 2, 3 y 4 describen cada uno un algoritmo de cálculo del *skeleton* implementado. Estos capítulos tienen la misma estructura. Primero se exponen algunos fundamentos teóricos que respaldan el algoritmo, para luego dar paso al detalle de la implementación. Cada capítulo termina con una discusión respecto del costo computacional del algoritmo en función del número de vóxeles de la entrada y la factibilidad de su paralelización. En el Capítulo 5 se describen las métricas de comparación de propuestas y se utilizan para comparar los algoritmos implementados. Finalmente, en el Capítulo 6 se presentan las conclusiones y discusión de este trabajo.

Capítulo 2

Cálculo del *Skeleton* por Adelgazamiento Topológico (Palágyi y Kuba)

El primer algoritmo implementado para esta tesis fue presentado por Palágyi y Kuba [63]. Este algoritmo simula la metáfora del incendio de Blum eliminando vóxeles del contorno del objeto hasta transformarlo en su *skeleton*. La secuencia de eliminación no es arbitraria, sino que está determinada por máscaras.

Un vóxel es eliminado si su 26-vecindad calza con alguna máscara, es decir, si satisface cierto patrón de unos y ceros. La Figura 2.1 muestra una de estas máscaras, donde el vóxel central, etiquetado con una v , es el que se elimina de haber calce con la máscara. En general, cualquier algoritmo que siga una estrategia similar es referido como “algoritmos de adelgazamiento topológico”, porque cada operación de eliminación va haciendo más delgado el objeto sin alterar su topología. Esto significa que una condición mínima para las máscaras es que solamente vóxeles simples sean eliminados.

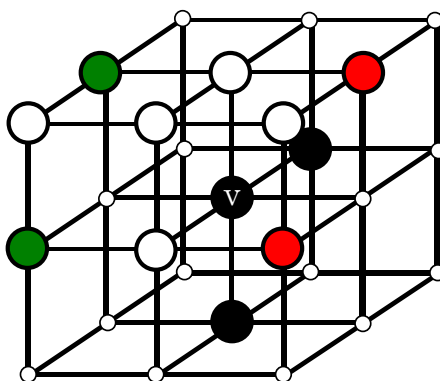


Figura 2.1: Ejemplo de máscara usada en este algoritmo.

2.1. Fundamentos teóricos

2.1.1. Adelgazamiento topológico en 2D

Revisar un algoritmo de adelgazamiento topológico de imágenes resulta útil para explicar el algoritmo de este capítulo, porque los principios son similares.

En procesamiento de imágenes, una operación morfológica llamada *hit-miss* puede ser usada para calcular el *skeleton* [44]. Esta operación busca patrones de unos y ceros en los píxeles de una imagen. Los patrones se especifican mediante máscaras, que en el caso del cálculo del *skeleton* corresponden a regiones de 3×3 píxeles. La Figura 2.2 muestra las máscaras necesarias para calcular el *skeleton* de una imagen.

| | | |
|---|---|---|
| 0 | 0 | 0 |
| | 1 | |
| 1 | 1 | 1 |

| | | |
|---|---|---|
| | 0 | 0 |
| 1 | 1 | 0 |
| | 1 | |

Figura 2.2: Máscaras para el cálculo del *skeleton* en 2D. El *skeleton* se calcula aplicando sucesivamente *hit-miss* con la máscara de la izquierda en sus 4 rotaciones de 90° y luego con la de la derecha en sus 4 rotaciones de 90° , hasta que no se observen cambios.

En estas máscaras, un 0 indica calce con un píxel de fondo, un 1 indica calce con un píxel de objeto y vacío indica indiferencia. La operación *hit-miss* consiste en recorrer todos los píxeles p de la imagen y ver si $V_8(p)$ calza con la máscara. Si hay un calce, se marca con un 1 la posición de p en la imagen de salida, que inicialmente solo tiene ceros. Con esto, los píxeles de objeto de la imagen de salida corresponden a los píxeles de la imagen de entrada donde hubo calce.

El cálculo del *skeleton* utilizando *hit-miss* es iterativo. En cada iteración se calcula *hit-miss* de la imagen usando las máscaras de la Figura 2.2, secuencialmente, en todas las rotaciones de 90° posibles (8 máscaras en total). Los píxeles marcados como consecuencia de esta operación se eliminan de la imagen, y se repite el proceso. El ciclo se detiene cuando *hit-miss* produce una imagen vacía, es decir, cuando no hay calce con ningún patrón. La Figura 2.3 ilustra el resultado de este algoritmo.

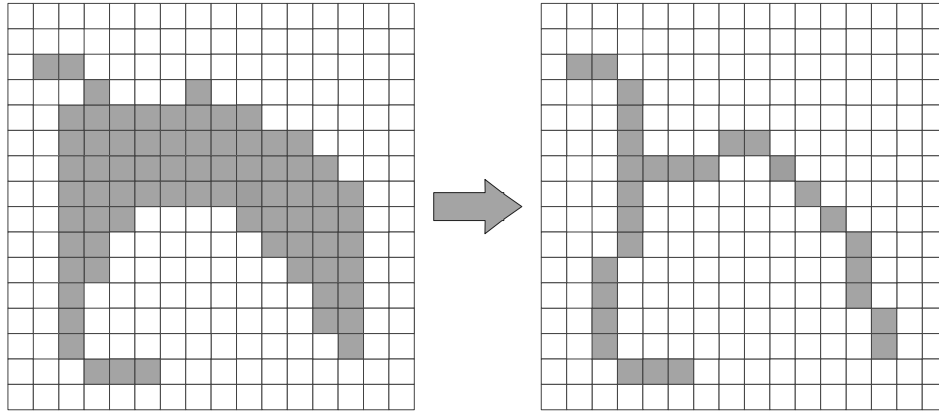


Figura 2.3: Resultado del adelgazamiento topológico de una imagen (luego de 3 iteraciones).

El algoritmo 3D de este capítulo tiene exactamente la misma estructura lógica. La diferencia está en el número de máscaras que se necesita para calcular el *skeleton* de un volumen. Solo existen $2^8 = 256$ posibles configuraciones para la 8-vecindad de un píxel de objeto y 4 rotaciones, mientras que para la vecindad de un vóxel de objeto existen $2^{26} \approx 67$ millones y 24 rotaciones.

2.2. Pseudocódigo del algoritmo

Algoritmo 2 Adelgazamiento topológico paralelo de Palágyi y Kuba

```

1: function ADELGAZAMIENTO_TOPOLOGICO_PARALELO( $V$ )
2:    $mascaras \leftarrow InicializarMascaras()$ 
3:    $skel \leftarrow V$ 
4:   do
5:      $eliminables \leftarrow Ceros(Size(skel))$ 
6:     for all  $r \in rotaciones$  do
7:       for all  $m \in mascaras$  do
8:          $mascaraRotada \leftarrow RotarMascara(m, r)$ 
9:          $eliminables \leftarrow eliminables \cup AplicarMascara(mascaraRotada, skel)$ 
10:      end for
11:    end for
12:     $skel \leftarrow skel - eliminables$ 
13:  while  $Contar(eliminables) > 0$ 
14:  return  $skel$ 
15: end function

```

Cada iteración de este algoritmo consiste en marcar los vóxeles cuya 26-vecindad calce con alguna de las máscaras, que son 14, rotada en alguna de 12 rotaciones predefinidas. Al final de la iteración, se eliminan los vóxeles marcados. Este proceso se repite hasta que no se marque ningún vóxel. Por lo tanto, en principio cada iteración consiste en revisar $12 \times 14 = 168$ veces la vecindad de cada vóxel del volumen. A continuación se describe la implementación de este proceso en detalle.

2.3. Implementación

2.3.1. Inicialización

La única entrada para este algoritmo es el volumen cuyo *skeleton* se quiere calcular. Esto significa que, a diferencia de los otros algoritmos implementados en esta tesis, este algoritmo no depende de parámetros que deban ser ajustados.

Esta parte del algoritmo se implementó de dos maneras, con el objetivo de asegurar la consistencia de sus resultados. Esto porque se encontró que es fácil cometer errores al especificar las máscaras. En principio, como cada máscara tiene 27 valores, es necesario escribir $14 \times 27 = 378$ valores en el código. Sin embargo, en la primera implementación se quiso usar la operación morfológica *hit-miss*, pensando en aprovechar que MATLAB dispone de la operación *hit-miss* para volúmenes en la función `bwhitmiss`. Como la operación *hit-miss* solamente admite máscaras cuyos valores calcen estrictamente con 1, estrictamente con 0, o con cualquier valor, no es posible especificar condiciones de calce que involucren solo a cierto subconjunto de vóxeles (como, por ejemplo, que al menos un vóxel de entre los marcados con rojo en la máscara de la Figura 2.4 sea de fondo). Debido a esto, para poder usar *hit-miss* varias de las 14 máscaras de este algoritmo debieron ser divididas, como ejemplifica la Figura 2.4, proceso que originó 63 máscaras. Esto aumentó la posibilidad de error, puesto que ahora debieron escribirse $63 \times 27 = 1701$ valores.

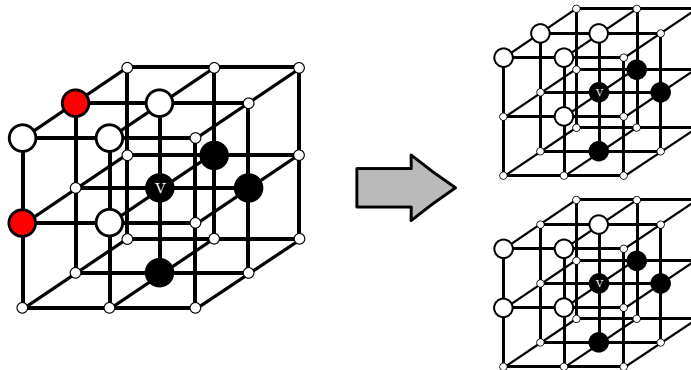


Figura 2.4: Ejemplo de máscara que debe dividirse para poder usar *hit-and-miss*.

Estimando que resultaba muy difícil asegurar la correctitud del código de la primera implementación, se programó el algoritmo por segunda vez, representando las máscaras de manera distinta. Esta segunda implementación adopta las sugerencias que aparecen en el artículo de Palágyi y Kuba [63]. Siguiendo estas sugerencias, cada máscara se representa como una función que recibe la 26-vecindad del vóxel que se quiere revisar. En el cuerpo de estas funciones, los vóxeles de objeto se consideran

true y los vóxeles de fondo *false*, y el calce con la máscara se realiza mediante operaciones booleanas elementales.

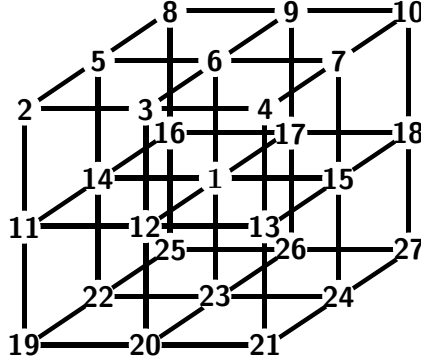


Figura 2.5: Etiquetado de la 26-vecindad.

Para ilustrar esta implementación, cada vóxel en la 26-vecindad es identificado con una etiqueta según su posición, como se muestra en la Figura 2.5. Luego, el calce de un vóxel v (cuya etiqueta en el cuerpo de la función es v_1) con la máscara de la Figura 2.4 puede ser revisado usando la función:

$$F_{M7}(V_{26}(v)) = v_1 \wedge v_{15} \wedge v_{17} \wedge v_{23} \wedge \neg(v_2 \vee v_3 \vee v_6) \wedge \neg(v_5 \wedge v_{11})$$

Al no usar funciones nativas, este enfoque tiene mayor costo computacional en MATLAB, por lo que su tiempo de ejecución es mayor. La ventaja es que las máscaras a ingresar se mantienen en 14, reduciendo la probabilidad de errores.

En definitiva, ambas implementaciones produjeron resultados idénticos para todas las figuras de prueba. Para revisar la equivalencia entre los resultados se construyó un tercer volumen a partir de la conjunción lógica entre los valores de índices correspondiente en los *skeletons* obtenidos mediante ambas implementaciones. Luego, se verificó que este volumen tuviera el mismo número de vóxeles de objeto que ambos *skeletons*.

La Figura 2.6 ilustra las máscaras orientadas en su primera rotación.

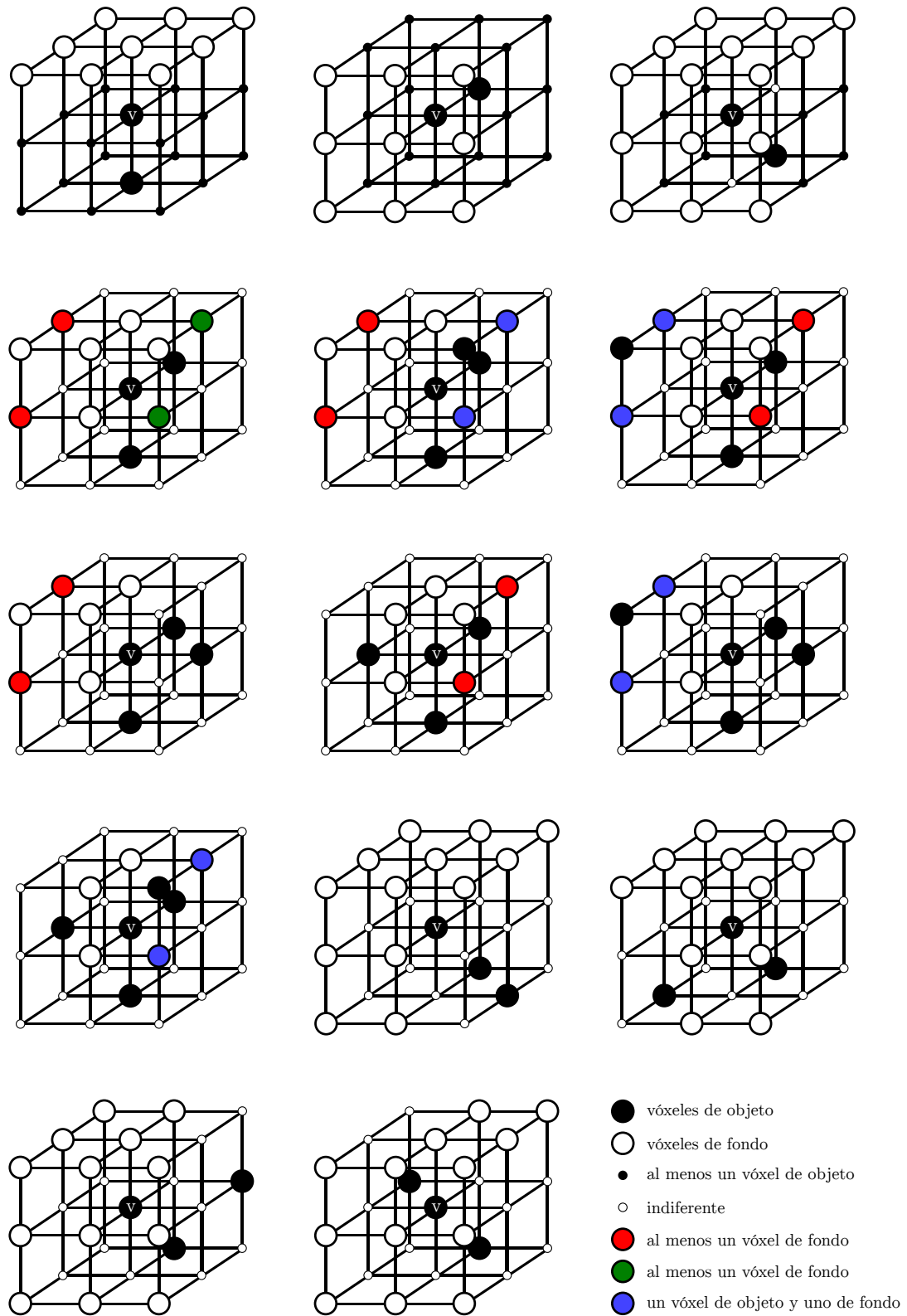


Figura 2.6: Las 14 máscaras usadas en este algoritmo, en su primera rotación (US).

2.3.2. Rotación y aplicación de las máscaras

Algoritmo 2 Adelgazamiento Topológico Paralelo - Aplicación de las máscaras

```
1:  $skel \leftarrow V$ 
2: do
3:    $eliminables \leftarrow Ceros(Size(skel))$ 
4:   for all  $r \in rotaciones$  do
5:     for all  $m \in mascaras$  do
6:        $mascaraRotada \leftarrow RotarMascara(m, r)$ 
7:        $eliminables \leftarrow eliminables \cup AplicarMascara(mascaraRotada, skel)$ 
8:     end for
9:   end for
10:   $skel \leftarrow skel - eliminables$ 
11: while  $Contar(eliminables) > 0$ 
12: return  $skel$ 
```

La variable $skel$, inicializada como una copia del volumen de entrada, almacena el resultado parcial de cada iteración. Se espera que $skel$ tenga menos vóxeles de objeto luego de cada iteración.

El primer paso de cada iteración consiste en crear una matriz de ceros del mismo tamaño que el volumen de entrada. Esta matriz se usará para marcar los vóxeles que se eliminarán al final de cada iteración.

En la Figura 2.6 las máscaras aparecen orientadas en la primera de las 12 rotaciones (US). Las rotaciones de las máscaras, identificadas a partir del etiquetado de la Figura 2.7, deben seguir el orden: $US, NE, WD, ES, UW, ND, SW, UN, UD, NW, UE$ y SD . Este orden debe obedecerse estrictamente. Un orden diferente produciría un *skeleton* distinto.

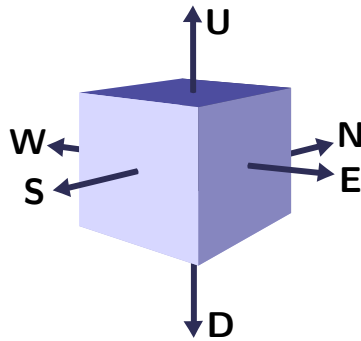


Figura 2.7: Etiquetado de las caras de una máscara.

Para facilitar el cálculo iterativo, la operación de rotar una máscara fue implementada de manera tal que recibe el índice r de la rotación como parámetro, con $r = 1, \dots, 12$. Cada rotación se programó

como una permutación a partir de la rotación US . Es decir, para $i = 1$ efectúa la rotación $US \rightarrow US$, para $i = 2$ la rotación $US \rightarrow NE$, para $i = 3$ la rotación $US \rightarrow WD$, etcétera.

Habiendo rotado la máscara, la función *AplicarMascara* recorre cada vóxel del volumen para determinar el calce con la máscara rotada. Debido a esto, cada aplicación de una máscara tiene costo $\Theta(n)$. Los índices donde haya calce son marcados con un 1 en la matriz *eliminables*.

Al final de cada iteración se eliminan del resultado parcial los vóxeles marcados en *eliminables*. Para esto, basta con una resta entre matrices, operación de costo asintótico dentro de $\Theta(n)$. El ciclo termina si no se marcaron vóxeles para eliminar. Cuando se cumple esta condición, se ha calculado el *skeleton*.

2.4. Discusión

El costo computacional de este algoritmo es altamente dependiente de la entrada. La operación de calzar una máscara con un volumen de n vóxeles tiene costo asintótico lineal en n , por lo que entre más vóxeles se eliminen por calce, mejor será su rendimiento. El peor caso corresponde a una figura donde cada iteración elimine una cantidad de vóxeles muy pequeña en comparación al número de vóxeles de objeto del resultado parcial. En ese caso, eliminar cada vóxel tendría un costo asintótico lineal en n , quedando la complejidad del algoritmo en $\mathcal{O}(n^2)$. En el mejor caso, la entrada es un *skeleton*, o se calcula su *skeleton* en una iteración. En ese caso, la complejidad del algoritmo pertenece a $\Theta(n)$.

Este algoritmo es altamente paralelizable. En primer lugar, las operaciones de calce con máscaras en cada iteración podrían dividirse en 168 hilos de ejecución, uno por cada máscara en cada rotación. Esto porque no existen condiciones de carrera (*data race*) entre el calce con máscaras distintas sino hasta el final de cada iteración, cuando debe actualizarse la matriz *skel* antes de pasar a la iteración siguiente. Por otro lado, determinar el calce con alguna máscara es un proceso arbitrariamente paralelizable, particionando la matriz *skel* y asignando un hilo por partición.

La Figura 2.8 ilustra los resultados de este algoritmo para un conjunto de volúmenes de prueba.

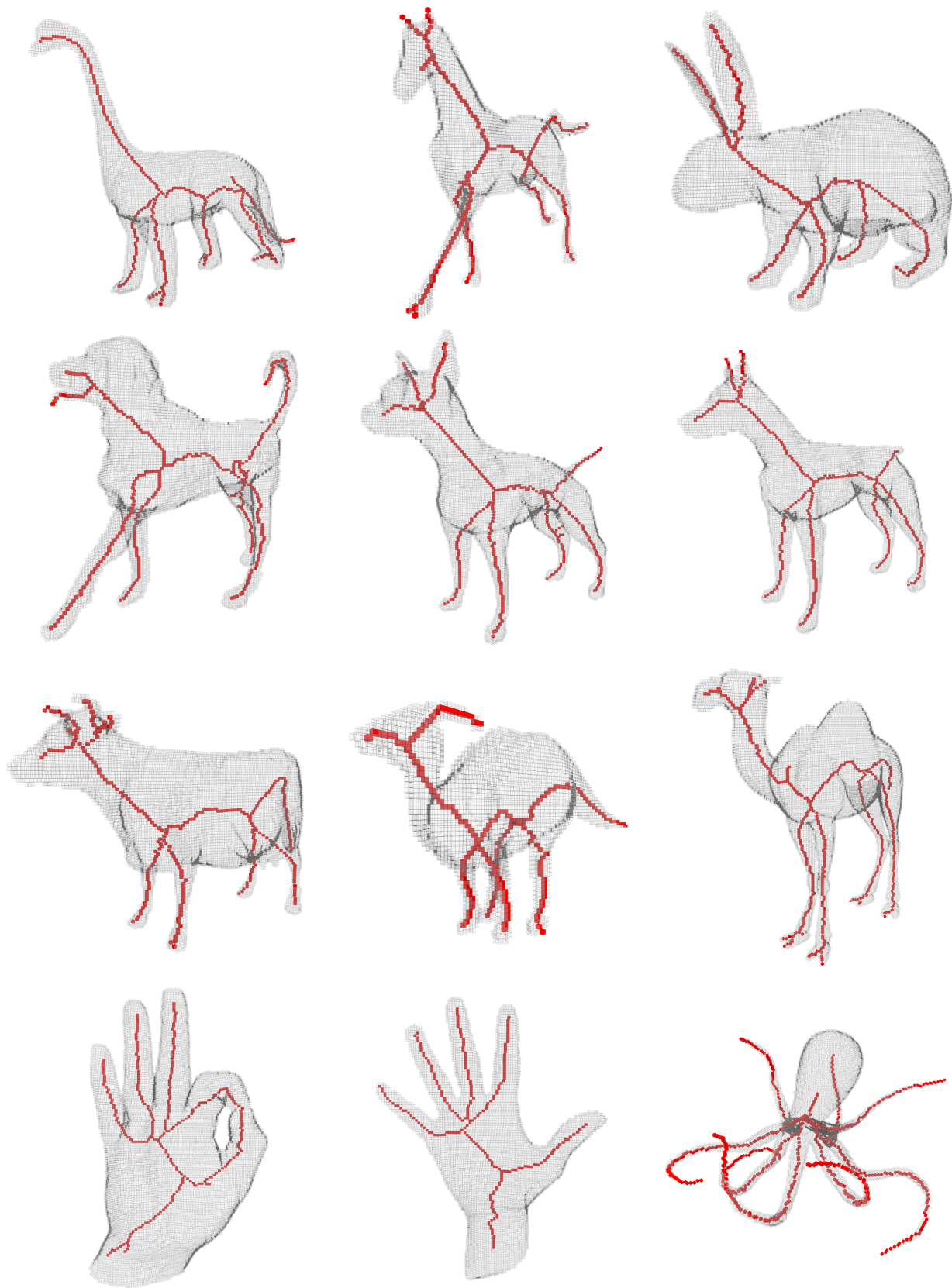


Figura 2.8: Resultados del algoritmo de Palágyi y Kuba para los volúmenes de prueba.

Capítulo 3

Cálculo del *Skeleton* Basado en la Divergencia (Siddiqi et al.)

Recordando lo señalado en el Capítulo 1, la metáfora del incendio de Blum [14] dice que el *skeleton* de un objeto se puede obtener encendiéndole fuego a toda su frontera simultáneamente. Asumiendo que los frentes de llamas se propagan a velocidad constante hacia el interior del objeto, el *skeleton* queda formado en los lugares donde dos frentes distintos se encuentran. En otras palabras, el *skeleton* corresponde a los puntos de extinción, donde el fuego no pudo seguir avanzando por haberse encontrado consigo mismo. Estos puntos se conocen hoy en día como *puntos de choque* [29].

Es posible simular directamente el frente de propagación de Blum como un flujo geométrico homogéneo [48]. Sin embargo, enfrentar el problema específico de encontrar los puntos de choque (y por lo tanto, el *skeleton*) permite simplificar los cálculos. Con ese propósito, los autores del algoritmo descrito en este capítulo [86] se basan en una cantidad similar al laplaciano de la transformada de distancia euclidiana.

3.1. Fundamentos teóricos

La idea central de este algoritmo es calcular el *skeleton* a partir de una cantidad escalar llamada *flujo emergente promedio*. Esta cantidad se define para una región cerrada R y un campo vectorial \mathbf{F} como:

$$\frac{\int_{\delta R} \langle \mathbf{F}, \mathcal{N} \rangle ds}{\text{largo}(\delta R)}, \quad (3.1)$$

donde ds es el diferencial del contorno de δR y \mathcal{N} es la normal exterior en cada punto del contorno.

En palabras simples, el flujo emergente promedio es un indicador de cómo varía la forma de la región δR al ser deformada por el campo vectorial \mathbf{F} . El flujo emergente promedio es negativo si el área encerrada por δR se encoje ante la acción de \mathbf{F} , positivo si crece y cero si se mantiene.

Entonces, el *skeleton* puede ser detectado a partir del flujo emergente promedio escogiendo un campo vectorial \mathbf{F} adecuado, es decir, que presente singularidades para el flujo emergente promedio en los puntos que corresponden al *skeleton*. El campo utilizado en este algoritmo es el gradiente de la transformada de distancia euclidiana. Esta transformada es un mapeo donde a cada vóxel de objeto se le asigna la distancia euclidiana al vóxel de fondo más cercano.

La Figura 3.1 ilustra los pasos que este algoritmo utiliza para calcular el flujo emergente promedio en una imagen. El caso de volúmenes es análogo.

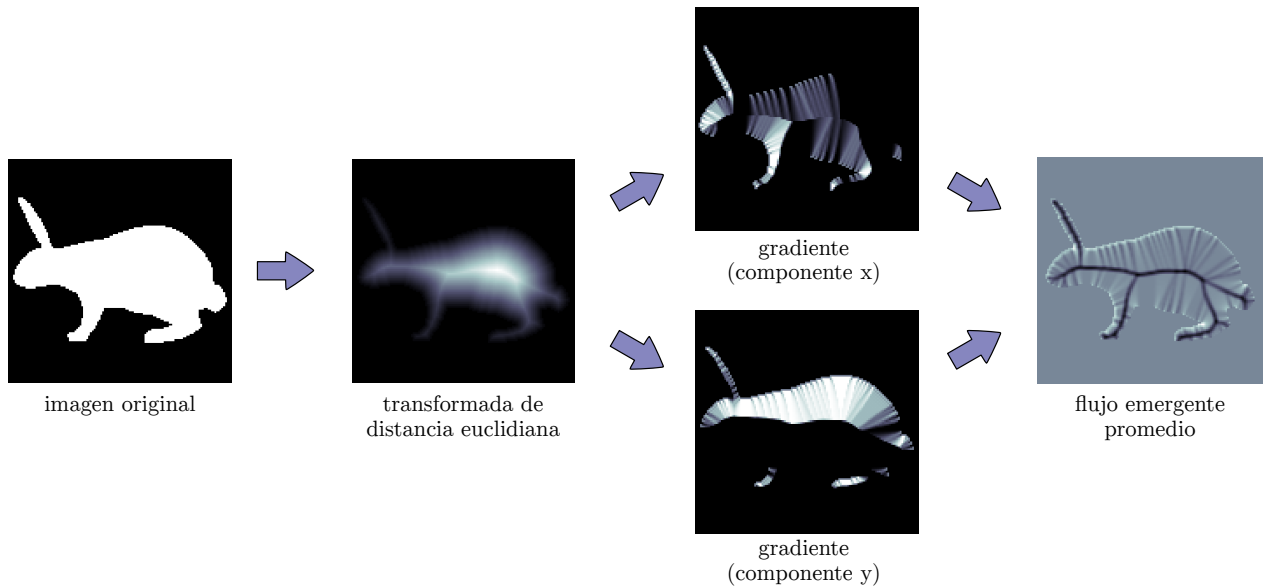


Figura 3.1: Pasos en el cálculo del flujo emergente promedio.

Puede verse gráficamente que cada componente del gradiente de la transformada de distancia euclidiana presenta discontinuidades relevantes en los ejes medios de partes distintas del objeto. El flujo emergente promedio se vuelve negativo si se toma una región que encierre un punto de estos ejes medios, porque el campo vectorial gradiente cambia de dirección, encogiendo esa región. La región utilizada en este algoritmo es la 26-vecindad de cada vóxel. Los vóxeles del *skeleton* se determinan en base a un umbral para el valor del flujo emergente promedio en cada 26-vecindad.

3.2. Pseudocódigo del algoritmo

Algoritmo 3 Extracción del *skeleton* de Hamilton-Jacobi.

```

1: function SKELETONDEHAMILTONJACOBI( $V, \tau$ )
2:    $tde \leftarrow TDE(V)$ 
3:    $\mathbf{F} \leftarrow \text{Gradiente}(tde)$ 
4:    $\mathcal{F} \leftarrow \text{FlujoEmergentePromedio}(\mathbf{F})$ 
5:    $C \leftarrow \text{ColaDePrioridad}()$ 
6:   for all  $v$  en el borde del objeto de  $V$  do
7:     if  $v$  es simple then
8:        $C.\text{insertar}(v, \mathcal{F}(v))$             $\triangleright v$  se inserta con el flujo como clave de ordenamiento
9:     end if
10:  end for
11:  while  $C.\text{contar}() > 0$  do
12:     $v \leftarrow C.\text{extraerPrimero}()$ 
13:    if  $v$  es simple then
14:      if  $v$  no es vóxel de término o  $\mathcal{F}(v) > \tau$  then
15:        Eliminar  $v$  de  $V$ 
16:        for all  $w \in V_{26}(v)$  do
17:          if  $w$  es simple y eliminable then
18:             $C.\text{insertar}(w, \mathcal{F}(w))$ 
19:          end if
20:        end for
21:      else
22:        Marcar  $v$  como no eliminable
23:      end if
24:    end if
25:  end while
26:  return  $V$ 
27: end function

```

Según lo señalado hasta ahora, el primer paso de este algoritmo consiste en calcular el flujo emergente promedio del gradiente de la transformada de distancia euclidiana. Sin embargo, basar íntegramente el cálculo del *skeleton* en un umbral para este flujo no garantiza la propiedad homotópica. Para preservar la topología y la conectividad del volumen, el resto del algoritmo emplea la noción de punto simple para erosionar el objeto.

Luego del cálculo del flujo emergente promedio, los vóxeles del borde del objeto se introducen en una cola de prioridad, donde la prioridad es el flujo emergente promedio en su 26-vecindad. El paso final consiste en la eliminación iterativa de los vóxeles simples de la cola que no sean vóxeles de término ni cuyo flujo emergente promedio supere el umbral τ . Si un vóxel simple no puede ser eliminado, se marca como no eliminable. Cada vez que se elimina un vóxel, se agregan a la cola sus 26-vecinos simples que no hayan sido marcados como no eliminables. El proceso termina cuando no quedan vóxeles en la cola.

3.3. Implementación

3.3.1. Cálculo del flujo emergente promedio

Algoritmo 3 Parte 1

```
1: function SKELETONDEHAMILTONJACOBI( $V, \tau$ )
2:    $tde \leftarrow TDE(V)$ 
3:    $\mathbf{F} \leftarrow \text{Gradiente}(tde)$ 
4:    $\mathcal{F} \leftarrow \text{FlujoEmergentePromedio}(\mathbf{F})$ 
```

La transformada de distancia euclidiana se calcula usando la función de MATLAB `bwdist`. La función `bwdist` calcula para cada vóxel de fondo la distancia euclidiana hasta el vóxel de objeto más cercano. Por lo tanto, antes de invocar a `bwdist` los valores del volumen V deben invertirse usando el operador de negación y así obtener la transformada de distancia adecuada.

El gradiente de la transformada de distancia \mathbf{F} se calcula mediante una convolución con un *kernel* de diferenciación aproximada, similar a un operador Sobel suavizado. Esta operación produce tres matrices que representan el gradiente en cada dirección.

El flujo emergente promedio \mathcal{F} se calcula para cada vóxel v a partir del gradiente y los vectores normales los vóxeles $w \in V_{26}(v)$:

$$\mathcal{F}(v) = \frac{1}{26} \sum_{w \in V_{26}(v)} \langle \mathcal{N}_{vw}, \mathbf{F}(w) \rangle. \quad (3.2)$$

Primero se suman los productos internos entre la normal desde v y el vector gradiente correspondiente a cada vóxel $w \in V_{26}(v)$. Finalmente, estos 26 valores se promedian.

3.3.2. Erosión guiada por el flujo emergente promedio

Algoritmo 3 Parte 2

```
5:    $C \leftarrow \text{ColaDePrioridad}()$ 
6:   for all  $v$  en el borde del objeto de  $V$  do
7:     if  $v$  es simple then
8:        $C.\text{insertar}(v, \mathcal{F}(v))$             $\triangleright v$  se inserta con el flujo como clave de ordenamiento
9:     end if
10:  end for
```

Para la cola de prioridad se utilizó una implementación orientada a objetos [36], que realiza las operaciones manipulando un *heap*.

Los primeros vóxeles que se insertan en la cola de prioridad son los vóxeles del borde del objeto. El artículo de Siddiqi et al. [86] no es específico en cuanto a la detección de los vóxeles del borde

del objeto. Para esta implementación se consideraron vóxeles del borde los vóxeles de objeto v con al menos un vóxel de fondo $w \in V_{26}(v)$.

Algoritmo 3 Parte 3

```

11:   while  $C.contar() > 0$  do
12:      $v \leftarrow C.extraerPrimero()$ 
13:     if  $v$  es simple then
14:       if  $v$  no es vóxel de término o  $\mathcal{F}(v) > \tau$  then
15:         Eliminar  $v$  de  $V$ 
16:         for all  $w \in V_{26}(v)$  do
17:           if  $w$  es simple y eliminable then
18:              $C.insertar(w, \mathcal{F}(w))$ 
19:           end if
20:         end for
21:       else
22:         Marcar  $v$  como no eliminable
23:       end if
24:     end if
25:   end while
26:   return  $V$ 
27: end function

```

Estas líneas corresponden al proceso de erosión de vóxeles que da origen al *skeleton*. En cada iteración se examina el vóxel con máximo flujo promedio emergente de entre los de cola de prioridad. Si no es simple, se ignora. Si es simple, es eliminado solo si no es vóxel de término o si su flujo emergente promedio supera el umbral τ . De ser eliminado, se insertan en la cola todos sus 26-vecinos simples que no han sido marcados como no eliminables. Si no es eliminado, es marcado como no eliminable.

El criterio utilizado para determinar cuándo un vóxel es de término es el propuesto por Pudney et al. [67], según el cual un vóxel es de término si tiene un único 26-vecino en alguno de los 9 planos de la Figura 3.2.

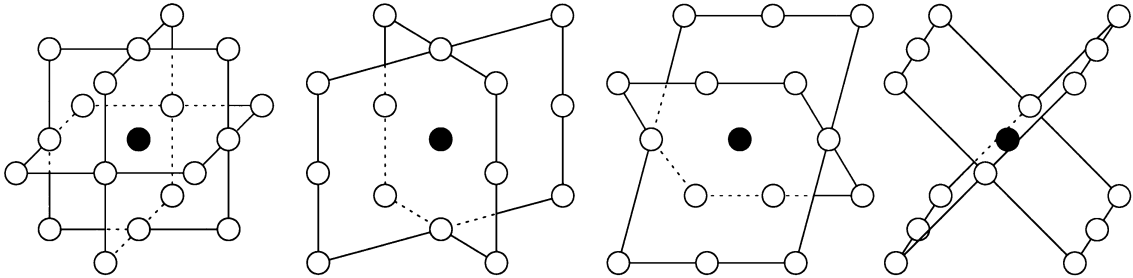


Figura 3.2: Los 9 planos para determinar si un vóxel es de término [67].

El *skeleton* calculado por este algoritmo contiene, por lo tanto: los vóxeles que en alguna iteración

satisfacen el criterio para ser de término, los vóxeles cuyo flujo emergente promedio es menor que τ y los vóxeles que en ninguna iteración satisfacen el criterio de simplicidad.

3.4. Discusión

El cálculo de la transformada de distancia de MATLAB usa el algoritmo de complejidad en $\Theta(n)$ de [57]. La complejidad de calcular el gradiente usando el operador Sobel también está en $\Theta(n)$, al igual que la del cálculo del flujo emergente promedio. Por lo tanto, la operación más costosa de este algoritmo es el proceso de erosión de vóxeles simples, que depende de la implementación para la cola de prioridad utilizada. En esta tesis se utilizó un *heap*, por lo que los costos de inserción y extracción del máximo tienen una complejidad en $\mathcal{O}(n \log(n))$ en el peor caso. Como cada vóxel es insertado en la cola un número constante de veces (a lo más 26, una vez por cada vecino), la complejidad del algoritmo también pertenece a $\mathcal{O}(n \log(n))$.

En cuanto a la factibilidad de paralelizar de este algoritmo, su proceso más costoso, el de erosión de vóxeles, no es paralelizable. Los vóxeles deben ser revisados siguiendo un orden estricto, puesto que el resultado de cada iteración determina el estado de los vóxeles (simplicidad o posible eliminación) para las iteraciones posteriores.

Finalmente, los resultados de este algoritmo son altamente sensibles a la elección del umbral superior τ para el flujo emergente promedio. La Figura 3.3 ilustra cómo pequeñas desviaciones del “óptimo” (en este caso, $\tau \approx -11,5$) originan ramas poco significativas en el *skeleton*. Por lo tanto, obtener el mejor *skeleton* para cada volumen depende de un proceso de calibración de este parámetro.

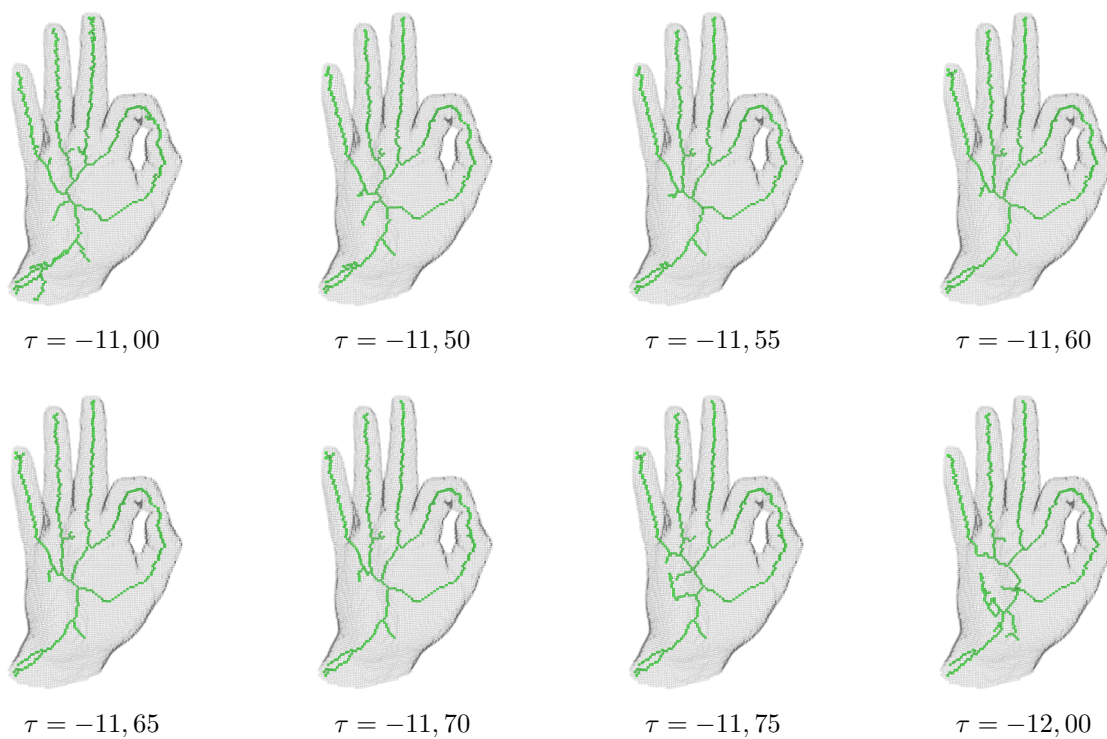


Figura 3.3: Resultados del algoritmo de Siddiqi et al. para distintos valores del umbral τ .

La Figura 3.4 ilustra los resultados para $\tau = -15$ en varios volúmenes de prueba.

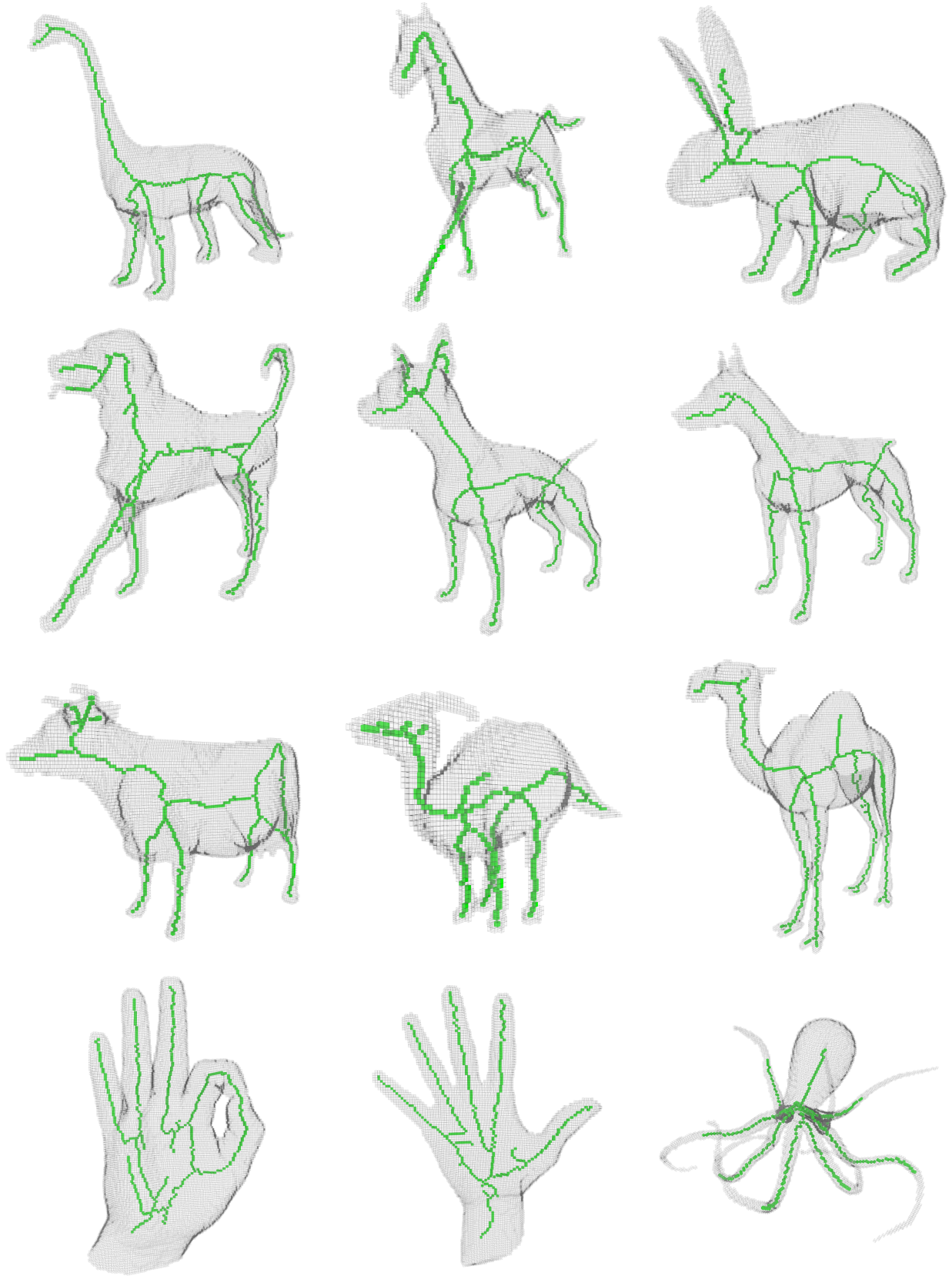


Figura 3.4: Resultados del algoritmo de Siddiqi et al. para los volúmenes de prueba, con $\tau = -15$.

Capítulo 4

Cálculo del *Skeleton* Basado en la Distancia (Arcelli et al.)

El tercer algoritmo implementado en esta tesis fue presentado por Arcelli et al. [8]. Este algoritmo difiere en mayor medida de los dos anteriores porque utiliza la segunda definición del *skeleton* presentada en el Capítulo 1. Es decir, el *skeleton* es calculado tomando como referencia los centros de las esferas maximales inscritas en el objeto.

Este algoritmo fue diseñado para calcular el *skeleton* de volúmenes. Sin embargo, para adquirir familiaridad con las técnicas utilizadas, se comenzó por implementar el algoritmo 2D basado en discos maximales propuesto por Di Baja et al. [27]. Este capítulo trata exclusivamente sobre el algoritmo 3D. En el Apéndice D se incluyen los detalles para el algoritmo 2D.

4.1. Fundamentos teóricos

4.1.1. Transformada de distancia

Una *transformada de distancia* de un volumen es un mapeo donde cada vóxel de objeto se etiqueta con la distancia al vóxel de fondo más cercano. Los vóxeles de fondo se etiquetan con el valor 0. En otras palabras, la transformada de distancia es una matriz del mismo tamaño que su volumen correspondiente. Si un vóxel vale 0, en su índice dentro de la transformada de distancia aparecerá un 0. Si vale 1, en su índice aparecerá un valor que indique la distancia entre ese vóxel y el vóxel con valor 0 más cercano. Este valor depende de cómo se mida la distancia entre vóxeles.

Fue necesario esbozar el concepto de transformada de distancia para explicar el algoritmo del Capítulo 3. Dentro de ese algoritmo, el cálculo de la transformada de distancia euclidiana era el primer paso para el cálculo del *skeleton*. Las operaciones subsecuentes eran de carácter global. En contraste, la métrica euclidiana no permite determinar con facilidad si un vóxel determinado corresponde al centro de una esfera maximal. La métrica euclidiana es exacta, por lo que no existe relación alguna entre los valores de vóxeles vecinos [17]. A pesar de esto, se han propuesto algunos

algoritmos que calculan el *skeleton* a partir de ella, como el de Remy y Thiel [71], que se basa en tablas de consulta con todos los valores posibles.

El algoritmo de este capítulo emplea una transformada más estable, donde los valores que pueden aparecer son siempre números enteros. Es parte de la familia de transformadas de distancia que se describe a continuación.

4.1.2. Transformada de distancia con pesos

Una *transformada de distancia con pesos* es una transformada de distancia donde cada vóxel de objeto v se etiqueta con el largo del 26-camino más corto que empieza en v y termina en un vóxel de fondo. Este tipo de transformada se calcula a partir de una 3-tupla $\langle p_c, p_a, p_v \rangle$, donde p_c es la distancia entre dos vóxeles que comparten una cara, p_a es la distancia entre dos vóxeles que comparten exactamente una arista y p_v es la distancia entre dos vóxeles que comparten exactamente un vértice.

La idea básica de las transformadas de distancia con pesos es aproximar la transformada de distancia euclidiana, simplificando al mismo tiempo los cálculos. La transformada de distancia con pesos que interesa para este algoritmo es la $\langle 3, 4, 5 \rangle$, cuyo error máximo con respecto a la euclidiana es del 12 % [15].

4.1.3. Centros de esferas maximales en la transformada de distancia $\langle 3, 4, 5 \rangle$

La sencillez de las transformadas de distancia con pesos permite encontrar máximos locales con facilidad. Dada una transformada de distancia con pesos d , puede decirse que un vóxel v es el centro de una esfera maximal, CEM en adelante, comparando su etiqueta $d(v)$ con las de sus vecinos, tomando en cuenta el peso según su relación de vecindad. En particular, para la transformada de distancia $\langle 3, 4, 5 \rangle$ un vóxel v es CEM si se cumplen las siguientes condiciones:

$$\begin{aligned} \forall w \in V_6(v), d(w) - d(v) < 3, \\ \forall w \in V_{18}(v), d(w) - d(v) < 4 \quad \text{y} \\ \forall w \in V_{26}(v), d(w) - d(v) < 5. \end{aligned} \tag{4.1}$$

Estas condiciones buscan medir el grado de relevancia de la esfera maximal centrada en v . La relevancia de la esfera centrada en v será mayor cuanto mayor sea el grado de solapamiento con las esferas centradas en sus vecinos (esto es, cada esfera de radio $d(w)$ centrada en un vecino w). Así, cuanto mayor sea alguna diferencia $d(w) - d(v)$, menor será la relevancia de v como CEM.

La noción de relevancia permite refinar la detección de CEM. Así, se dice que un vóxel v es un *CEM altamente relevante* si cumple las condiciones:

$$\begin{aligned} \forall w \in V_6(v), d(w) - d(v) < 2, \\ \forall w \in V_{18}(w), d(w) - d(v) < 3 \quad \text{y} \\ \forall w \in V_{26}(w), d(w) - d(v) < 4. \end{aligned} \tag{4.2}$$

4.1.4. Capas en la transformada de distancia $\langle 3, 4, 5 \rangle$

Dado un vóxel v y su correspondiente etiqueta $d(v)$, sus vecinos más “profundos” dentro del objeto pueden tener los valores $d(v) + 3$, $d(v) + 4$, o $d(v) + 5$, dependiendo de su relación de vecindad con v . Siguiendo lo mostrado por Svensson et al. [94], esto permite particionar los vóxeles del objeto en subconjuntos llamados *capas*, tal que los vóxeles q de la capa k -ésima C_k satisfacen

$$k = \left\lceil \frac{d(q)}{p_c} \right\rceil. \tag{4.3}$$

La Figura 4.1 ilustra este concepto. En ella, cada color en el corte axial del objeto representa una capa distinta.

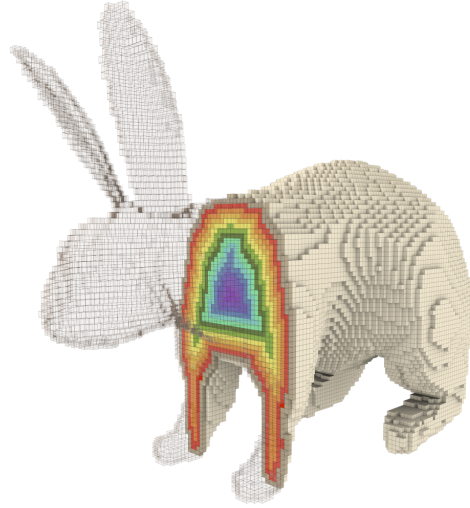


Figura 4.1: Capas en la transformada de distancia.

Las capas pueden ser interpretadas como un frente que se propaga discretamente hacia el interior del objeto. Esta interpretación indica que el concepto de capas podría ser utilizado para detectar el *skeleton*, pero debido a su inexactitud sería fácil detectar CEM erróneos. Sin embargo, tiene valor como un segundo criterio de refinamiento para definir el conjunto de vóxeles más importante para este

algoritmo. Este conjunto es el de los *CEM esenciales*, que se eligen de acuerdo a un procedimiento de dos pasos. En primer lugar, se dice que un vóxel v de la capa C_k es una CEM esencial si cumple que

1. v es un CEM altamente relevante y
2. todos los 26-vecinos de v en C_k son CEM altamente relevantes.

En segundo lugar, por cada vóxel v marcado como CEM esencial según este criterio, todos los CEM altamente relevantes de C_k en $V_{26}(v)$ son también marcados como CEM esenciales.

El procedimiento para elegir los CEM esenciales es ligeramente distinto en el artículo donde se presenta este algoritmo. En el artículo, los autores replican la definición de Svensson et al. [94] para definir los CEM esenciales, según la cual un CEM es CEM esencial si pertenece a una componente 26-conexa de CEM en la capa C_k que contiene al menos un CEM sin 26-vecinos en la capa C_{k+1} . El procedimiento anterior contradice esta definición, porque en ningún caso impone una restricción según la adyacencia con vóxeles en capas distintas. No obstante, se comprobó que usar la definición de Svensson et al. produce resultados diferentes a los del artículo, como se muestra en la Figura 4.2. Por otro lado, inmediatamente después de la definición de Svensson et al. en el artículo aparece el procedimiento anterior, salvo que no se menciona la restricción de que todo CEM esencial debe ser altamente relevante. Omitir esta restricción, determinada mediante experimentos, también produce resultados distintos a los mostrados en el artículo, como ilustra la Figura 4.2.

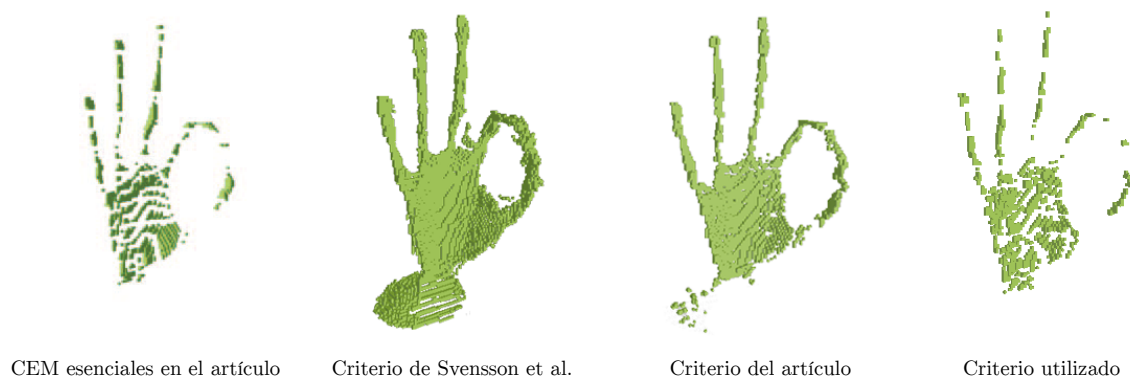


Figura 4.2: Comparación de detección de CEM esenciales según distintos criterios, calculados a partir de una figura similar a la del artículo.

4.2. Pseudocódigo del algoritmo

Algoritmo 4 Skeletonización basada en distancia

```
1: function SKELETONIZACIONBASADAENDISTANCIA( $V, \theta_1, \theta_2$ )
2:    $td \leftarrow TD345(V)$ 
3:    $cem \leftarrow ExtraerCEM(td)$ 
4:    $anclas \leftarrow Filtrar(cem)$ 
5:    $pssCasiDelgado \leftarrow ErosionarYConectar(anclas)$ 
6:    $pssDelgado \leftarrow Adelgazar(pssCasiDelgado)$ 
7:    $pss \leftarrow Podar(pssDelgado, \theta_1, \theta_2)$ 
8:    $vc \leftarrow ClasificarVoxeles(pss)$ 
9:    $tdd \leftarrow TD345Delimitada(pss, vc)$ 
10:   $cemPss \leftarrow ExtraerCEM(pss, tdd)$ 
11:   $anclasPss \leftarrow Filtrar(cemPss)$ 
12:   $skeletonCasiDelgado \leftarrow Erosionar(PSS, anclasPss)$ 
13:   $skeleton \leftarrow Adelgazar(skeletonCasiDelgado)$ 
14:   $skeleton \leftarrow Podar(skeleton, \theta_1, \theta_2)$ 
15:  return  $skeleton$ 
16: end function
```

Solamente los primeros pasos de este algoritmo corresponden a la detección de los distintos tipos de CEM en el volumen de entrada. El resto del algoritmo consiste en procedimientos que eliminan vóxeles según criterios basados en los CEM detectados. En la siguiente sección se detallan estos procedimientos. En resumen, tras determinar el conjunto de vóxeles que anclará el *skeleton* se sigue con la erosión, adelgazamiento y poda del volumen. Esto resulta en un *pseudo-skeleton* de superficie. Luego, se detectan los CEM en el *pseudo-skeleton* de superficie, y el proceso se repite de manera similar para obtener el *skeleton*.

4.3. Implementación

4.3.1. Cálculo de la transformada de distancia $\langle 3, 4, 5 \rangle$

Algoritmo 4 Parte 1

```
1: function SKELETONIZACIONBASADAENDISTANCIA( $V, \theta_1, \theta_2$ )
2:    $td \leftarrow TD345(V)$ 
```

Además del volumen de entrada V , este algoritmo requiere los parámetros numéricos θ_1 y θ_2 , que permiten calibrar el nivel de detalle del *skeleton*. Estos parámetros se utilizan para las operaciones de poda, de las líneas 7 y 14 del pseudocódigo. Se trata de umbrales que determinan la eliminación de líneas poco relevantes del *skeleton*. Por defecto, $\theta_1 = 4$ y $\theta_2 = 0.25$.

Para calcular la transformada $\langle 3, 4, 5 \rangle$ se implementó el algoritmo de Borgfors et al. [15]. Este

algoritmo permite calcular una transformada de distancia con pesos en tiempo lineal en el número de vóxeles. Consiste en recorrer el volumen dos veces, propagando los costos mínimos desde esquinas opuestas en cada pasada. El siguiente pseudocódigo describe este algoritmo:

Algoritmo 5 Cálculo de la transformada de distancia (3, 4, 5)

```

1: function TD345( $V$ )
2:    $d \leftarrow Ceros(V)$ 
3:   for  $i \leftarrow 1, Ancho(V)$  do ▷ Recorrido hacia adelante
4:     for  $j \leftarrow 1, Alto(V)$  do
5:       for  $k \leftarrow 1, Largo(V)$  do
6:         if  $(v \leftarrow V_{i,j,k}) = 1$  then
7:            $d(v) \leftarrow \min_{w \in V_F} d(w) + p_w$ 
8:         end if
9:       end for
10:    end for
11:  end for
12:  for  $i \leftarrow Ancho(V), 1$  do ▷ Recorrido hacia atrás
13:    for  $j \leftarrow Alto(V), 1$  do
14:      for  $k \leftarrow Largo(V), 1$  do
15:        if  $(v \leftarrow V_{i,j,k}) = 1$  then
16:           $d(v) \leftarrow \min_{w \in V_B} d(w) + p_w$ 
17:        end if
18:      end for
19:    end for
20:  end for
21:   $d \leftarrow ReemplazarValoresEquivalentes(d)$ 
22:  return  $d$ 
23: end function

```

La transformada de distancia d se inicializa como una matriz de ceros del mismo tamaño que V . El primer recorrido comienza desde la esquina donde empiezan los índices en el volumen. Para cada vóxel de objeto v , $V_F(v)$ es el subconjunto ya recorrido de $V_{26}(v)$. La distancia $d(v)$ se calcula como el mínimo de los valores de d para cada vóxel $w \in V_F(v)$ más el peso p_w correspondiente a la relación de vecindad entre w y v ; por ejemplo, hay que sumar 5 si w comparte solo un vértice con v . El segundo recorrido se efectúa en orden inverso. La única diferencia con el primer recorrido es que $V_B(v)$ contiene además a v . Es decir, teniendo el largo del camino encontrado en el primer recorrido, el segundo recorrido verifica si existe un camino más corto en alguna dirección no revisada. Por último, la función *ReemplazarValoresEquivalentes()* del final del algoritmo simplemente sustituye todos los valores 3 de la transformada por el valor 1. Este reemplazo permite la identificación correcta de máximos locales cercanos a los bordes [6].

4.3.2. Filtrado de CEM

Algoritmo 4 Parte 2

3: $cem \leftarrow \text{ExtraerCEM}(V)$
4: $anclas \leftarrow \text{Filtrar}(cem)$

La primera línea corresponde a detectar cada vóxel que cumple las condiciones para ser un CEM, un CEM altamente relevante o un CEM esencial. Las posiciones de los vóxeles que cumplen las condiciones se guardan en tres matrices booleanas, una por cada tipo de CEM. El volumen debe ser recorrido dos veces, una para detectar los CEM normales y altamente relevantes, y otra para determinar cuáles son CEM esenciales, por lo que la complejidad de este paso está en $\Theta(n)$.

En la segunda línea se detectan los *vóxeles ancla*. Estos son los vóxeles que en adelante servirán de guías para construir el *skeleton*. En principio, se marcan en una matriz booleana las posiciones de todos los CEM esenciales como vóxeles ancla. De entre los vóxeles que no son CEM esenciales, se marcan también como vóxeles ancla los CEM altamente relevantes con una alta *convexidad local*. La convexidad local de un vóxel v se calcula como el número de 26-vecinos de v con un valor mayor al de v en la transformada de distancia. Entonces, los vóxeles v que sean CEM altamente relevantes y para los cuales además su convexidad local sea mayor a un umbral τ son marcados como vóxeles ancla. De acuerdo al artículo de Arcelli et al. [8], $\tau = 7$ produce buenos resultados, por lo que τ se fija en ese valor. Este paso requiere recorrer la imagen una vez, estando su costo por lo tanto en $\Theta(n)$. Se marcan además como vóxeles ancla los CEM altamente relevantes que pertenezcan a una componente 26-conexa de CEM con al menos un CEM esencial. Este segundo criterio busca recalcar la importancia que se le da a los CEM esenciales. Como ningún vóxel se revisa más de una vez, la complejidad de este paso también está en $\Theta(n)$.

La Figura 4.3 muestra el volumen que se usa para ilustrar los pasos de este algoritmo y los vóxeles ancla seleccionados.

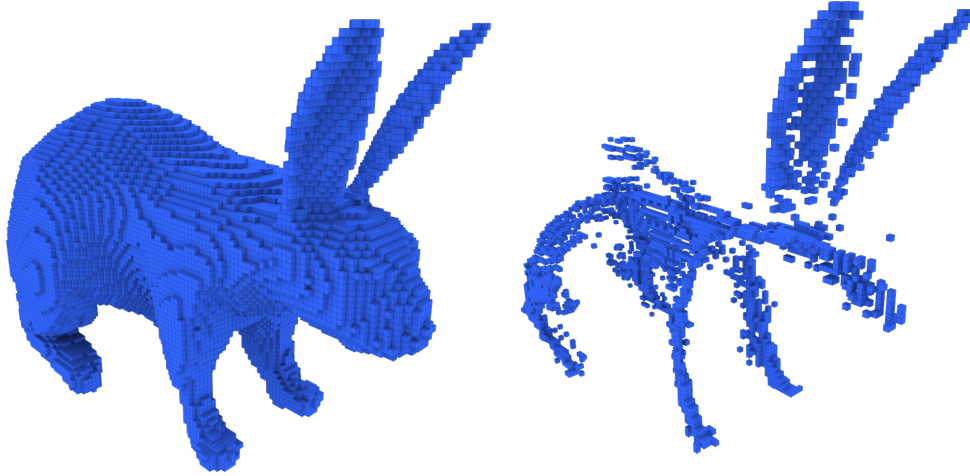


Figura 4.3: Volumen de prueba y vóxeles ancla seleccionados.

4.3.3. Obtención del PSS casi delgado

Algoritmo 4 Parte 3

```

5:   $pssCasiDelgado \leftarrow ErosionarYConectar(anclas)$ 
6:   $pssDelgado \leftarrow Adelgazar(pssCasiDelgado)$ 
7:   $pss \leftarrow Podar(pssDelgado, \theta_1, \theta_2)$ 

```

En el artículo de este algoritmo [8] se explica cómo pequeños cambios en la implementación permiten obtener un *skeleton* de superficie ajustado a la definición formal. Sin embargo, para obtener el *skeleton* curvilíneo es suficiente calcular primero lo que los autores llaman el *pseudo-skeleton de superficie* (PSS), una estructura formada por líneas y parches de superficie, centrada y homotópica, pero más sencilla que el *skeleton* de superficie calculado por un algoritmo diseñado con ese fin.

El primer paso para obtener el PSS es una erosión de vóxeles simples, ordenados ascendentemente según su valor en la transformada de distancia. A diferencia de la erosión en el algoritmo del Capítulo 3, en este algoritmo se revisan todos los vóxeles con el mismo valor en la transformada de distancia a la vez. Esto es posible gracias a que la transformada de distancia es discreta.

En principio, un vóxel simple es eliminado del PSS (que se inicializa como una copia del volumen de entrada) si es que no fue marcado como vóxel ancla. Sin embargo, cada vez que en el proceso de erosión se encuentra un vóxel ancla v , también deben marcarse como parte del PSS los vecinos más apropiados para conectar v con el interior del objeto. Estos vóxeles tampoco son eliminados en iteraciones posteriores, sean simples o no. El criterio para seleccionar estos vecinos consiste en buscar el o los vóxeles $w \in V_{26}(v)$ tales que $d(w) > d(v)$ y que además maximicen la *derivada direccional* en $V_{26}(v)$, definida como:

$$\mathcal{D}_{(v,w)} = \frac{d(w) - d(v)}{p_w}, \quad (4.4)$$

donde p_w es el peso correspondiente a la relación de vecindad entre w y v . Los vóxeles marcados según este criterio se denominan *vóxeles de enlace*. Cuando el proceso de erosión pasa de examinar el valor de distancia m al valor $m + 1$, el criterio de la derivada direccional debe usarse para encontrar nuevos vóxeles de enlace que conecten hacia el interior del objeto tanto vóxeles ancla como vóxeles de enlace. La Figura 4.4 muestra el resultado de esta operación.

En la operación de erosión solo se revisa una vez cada vóxel de objeto y su 26-vecindad. Por lo tanto, la operación de mayor costo es el ordenamiento previo a la erosión, donde los vóxeles de objeto se ordenan según su valor en la transformada de distancia. Entonces, la complejidad de esta parte está en $\mathcal{O}(n \log n)$.

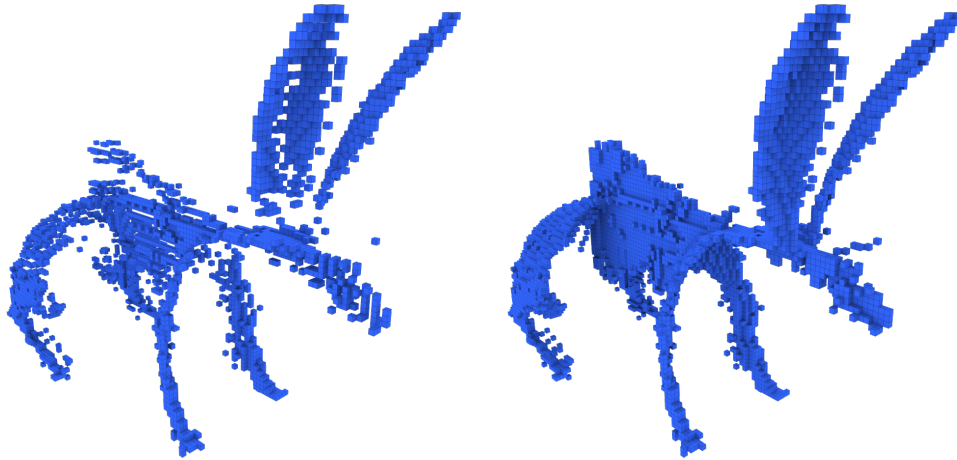


Figura 4.4: Vóxeles ancla conectados a través de vóxeles de enlace, formando el PSS casi delgado.

El *PSS casi delgado* es el objeto obtenido luego de haber procesado todos los valores de la transformada de distancia. Se le llama “casi delgado” porque, como se aprecia en la Figura 4.4, puede contener regiones de algunos vóxeles de ancho. Para reducir estas regiones, el PSS casi delgado es adelgazado usando máscaras, que se aplican de manera similar a las del algoritmo del Capítulo 2. En este caso, se usan las dos máscaras de la Figura 4.5, en cada una de sus 6 rotaciones posibles. El vóxel marcado con una flecha es el que se elimina de existir calce.

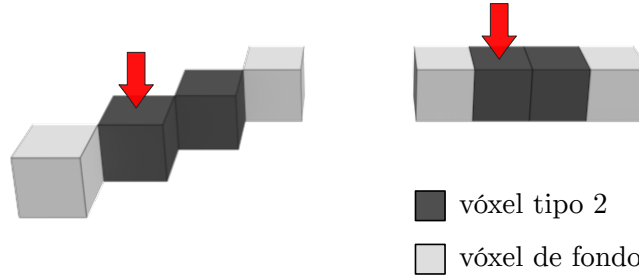


Figura 4.5: Máscaras usadas para adelgazar el PSS.

Los *vóxeles tipo 2* de las máscaras son los vóxeles que satisfacen la segunda condición para la simplicidad señalada en la Sección 1.4.1. Es decir, un vóxel es tipo 2 si es 6-adyacente con una única componente 6-conexa de vóxeles de fondo en su 26-vecindad. Esto asegura que la eliminación de los vóxeles en el proceso de adelgazamiento no produzca túneles. Sin embargo, no basta que un vóxel sea tipo 2 para su eliminación. Cuando una máscara calza, el vóxel revisado v es eliminado si satisface las siguientes 3 condiciones adicionales:

1. v es simple,
2. existe una única componente 26-conexa de vóxeles tipo 2 en $V_{26}(v)$ y
3. v es 26-vecino de dos o más vóxeles de objeto.

Las máscaras se aplican hasta que no se eliminen vóxeles (en todas las pruebas, basta con 3 iteraciones como máximo). El orden de aplicación no se especifica en el artículo. En la implementación de esta tesis se aplica la máscara recta en todas sus rotaciones y luego la máscara diagonal en todas sus rotaciones. El *PSS delgado* es el objeto que se obtiene terminado el proceso de adelgazamiento. La Figura 4.6 muestra el resultado de este proceso.

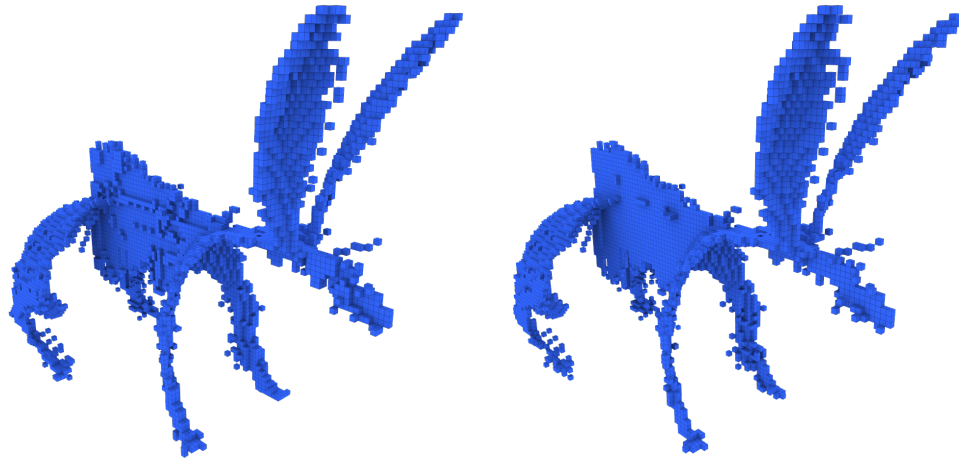


Figura 4.6: El PSS antes y después del adelgazamiento.

Luego del adelgazamiento se prosigue con la poda del PSS delgado. La poda es una operación presente en muchos algoritmos de cálculo del *skeleton* [22]. Recibe el nombre de poda porque consiste en eliminar *ramas* del PSS, que corresponden a los 26-caminos (v_1, \dots, v_f) tales que v_1 es un vóxel de término, v_f es un *vóxel de ramificación* (un vóxel con más de dos 26-vecinos) y $\{v_2, \dots, v_{f-1}\}$ contiene solamente *vóxeles de línea* (vóxeles con exactamente dos 26-vecinos). La Figura 4.8 muestra los vóxeles de término y los vóxeles de línea para el PSS delgado del volumen de prueba.

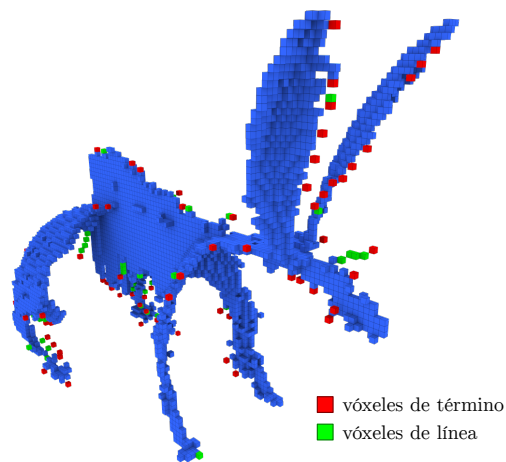


Figura 4.7: Tipos de vóxeles que forman las ramas del PSS delgado.

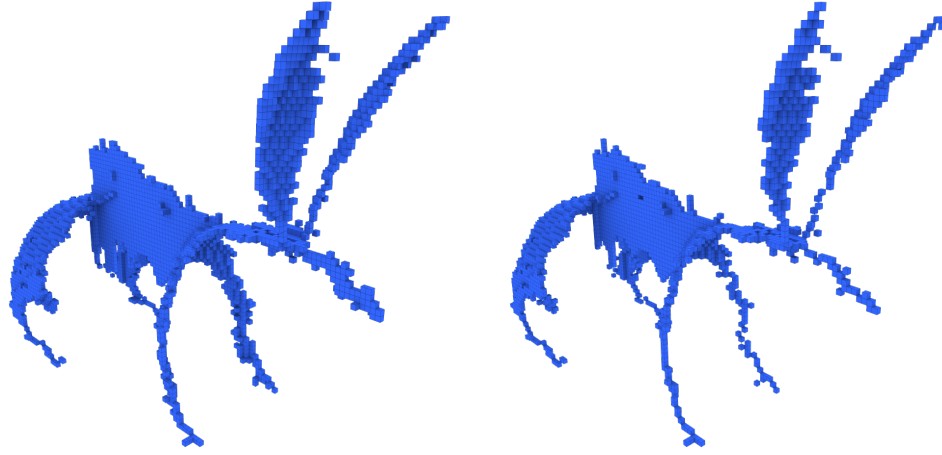


Figura 4.8: Resultados de ambas operaciones de poda del PSS.

Una rama es eliminada si no satisface cierto criterio. En este algoritmo se hacen dos podas, cada una basada en un criterio distinto, en ambos casos basado en el número de vóxeles ancla de la rama examinada. Para cada rama, se define L como el número vóxeles de línea y N como el número de vóxeles ancla. Luego, en la primera poda, una rama se elimina si $N < \theta_1$. En la segunda poda, una rama se elimina si $N/L < \theta_2$.

La detección y eliminación de las ramas se puede hacer recorriendo el volumen una vez por cada poda, por lo que en principio la complejidad del proceso de poda está en $\Theta(n)$. Sin embargo, el último vóxel de las ramas se trata de manera diferente. Si tras eliminar el resto de la rama el último vóxel se convierte en un vóxel de término, no se elimina. En caso contrario, se elimina, pero además se eliminan todos los vóxeles simples de la componente 26-conexa de vóxeles de ramificación a la cual pertenece. Esta componente se recorre usando un *stack*, como se ha descrito en la Sección 1.4. Entonces, por rama potencialmente debe recorrerse gran parte de los vóxeles de objeto del PSS, por lo que el proceso de poda tiene complejidad asintótica dentro de $\mathcal{O}(n^2)$ en el peor caso. Luego de completar ambas podas, se obtiene el PSS.

4.3.4. Cálculo de la transformada de distancia $\langle 3, 4, 5 \rangle$ del PSS

Algoritmo 4 Parte 4

```

8:  clasificacionDeVoxeles  $\leftarrow$  ClasificarVoxeles(pss)
9:  tddPss  $\leftarrow$  TD345Delimitada(pss, clasificacionDeVoxeles)
10: cemPss  $\leftarrow$  ExtraerCEM(pss, tddPss)
11: anclasPss  $\leftarrow$  Filtrar(cemPss)

```

El primer paso para calcular el *skeleton* a partir del PSS es calcular su transformada de distancia $\langle 3, 4, 5 \rangle$. Esta transformada no se puede calcular de la misma manera que para el volumen original, puesto que todos los vóxeles de objeto del PSS son 6-adyacentes con un vóxel de fondo. Entonces, en este caso la transformada de distancia se propaga a partir de un conjunto de *vóxeles de borde* que delimita los planos del PSS, ilustrado en la Figura 4.9.

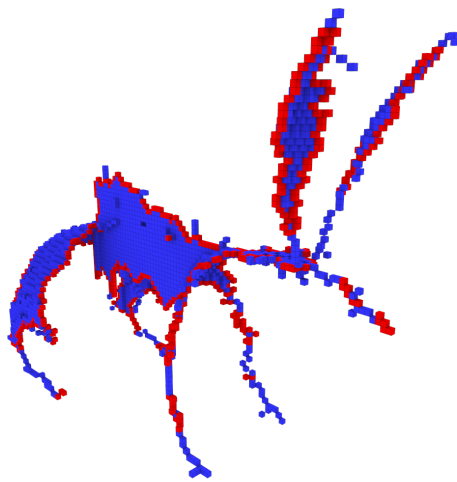


Figura 4.9: Vóxeles de borde en el PSS.

El conjunto de vóxeles de borde se determina usando el método de clasificación de vóxeles de Arcelli et al. [7]. Este método consiste en particionar el conjunto de vóxeles de objeto del volumen en 7 subconjuntos. En el Apéndice D se describe el procedimiento para efectuar esta partición.

Teniendo la clasificación, los valores para la transformada de distancia del PSS se propagan desde los vóxeles de borde, que se inicializan con el valor 3. La propagación se hace mediante un algoritmo similar al Algoritmo 5, como se describe en el artículo de Arcelli [8].

Habiendo calculado la transformada de distancia, los CEM del PSS se determinan según el mismo criterio que se utilizó para determinar los CEM altamente relevantes en el volumen original. Luego, los vóxeles ancla del PSS son los CEM detectados que al mismo tiempo fueron vóxeles ancla del volumen original, junto con los vóxeles de línea, los vóxeles de ramificación y los vóxeles de juntura.

La clasificación de vóxeles puede hacerse en 9 lecturas del volumen, por lo que su complejidad está en $\Theta(n)$. El cálculo de la transformada de distancia se hace en dos lecturas, por lo que su complejidad también está en $\Theta(n)$.

4.3.5. Extracción del *skeleton* a partir del PSS

Algoritmo 4 Parte 5

```
12: skeletonCasiDelgado  $\leftarrow$  Erosionar(pss, anclasPSS)
13: skeleton  $\leftarrow$  Adelgazar(skeletonCasiDelgado)
14: skeleton  $\leftarrow$  Podar(skeleton,  $\theta_1$ ,  $\theta_2$ )
15: return skeleton
16: end function
```

Teniendo los vóxeles ancla del PSS, los procesos de erosión, adelgazamiento y poda se repiten sobre el PSS de manera equivalente a como se hizo para obtener el PSS, salvo por las siguientes diferencias:

- En el proceso de erosión, siendo el PSS una estructura 26-conectada, no se definen vóxeles de enlace.
- En el proceso de adelgazamiento, las máscaras ahora calzan con vóxeles simples en lugar de vóxeles tipo 2.
- En el proceso de poda, los vóxeles ancla son los CEM del PSS que habían sido marcados como vóxeles ancla en el volumen original.

El *skeleton* es el resultado de este proceso.

4.4. Discusión

La operación más costosa en este algoritmo es el primer proceso de poda. Como se ha señalado anteriormente, la complejidad de este proceso es cuadrática en el número de vóxeles de objeto del volumen. Las pruebas realizadas corroboran que esta operación toma la mayor cantidad de tiempo.

Varias de las operaciones de este algoritmo no son paralelizables. Por ejemplo, el cálculo de la transformada de distancia $\langle 3, 4, 5 \rangle$ debe ser secuencial si se realiza utilizando el algoritmo presentado, porque los valores calculados en cada iteración dependen de los valores calculados en iteraciones anteriores. Por otro lado, las máscaras utilizadas para adelgazar el volumen no están diseñadas para ser aplicadas en paralelo. El adelgazamiento podría de todas formas paralelizarse mediante una estrategia basada en particionar el objeto, tratando de manera especial las juntas entre particiones. Finalmente, las operaciones de poda tampoco pueden hacerse fácilmente en paralelo, debido a que la posible examinación de la misma componente 26-conexa de vóxeles de ramificación viniendo desde ramas distintas significaría un *data race*.

La Figura 4.10 muestra los resultados para este algoritmo para algunas figuras de prueba.

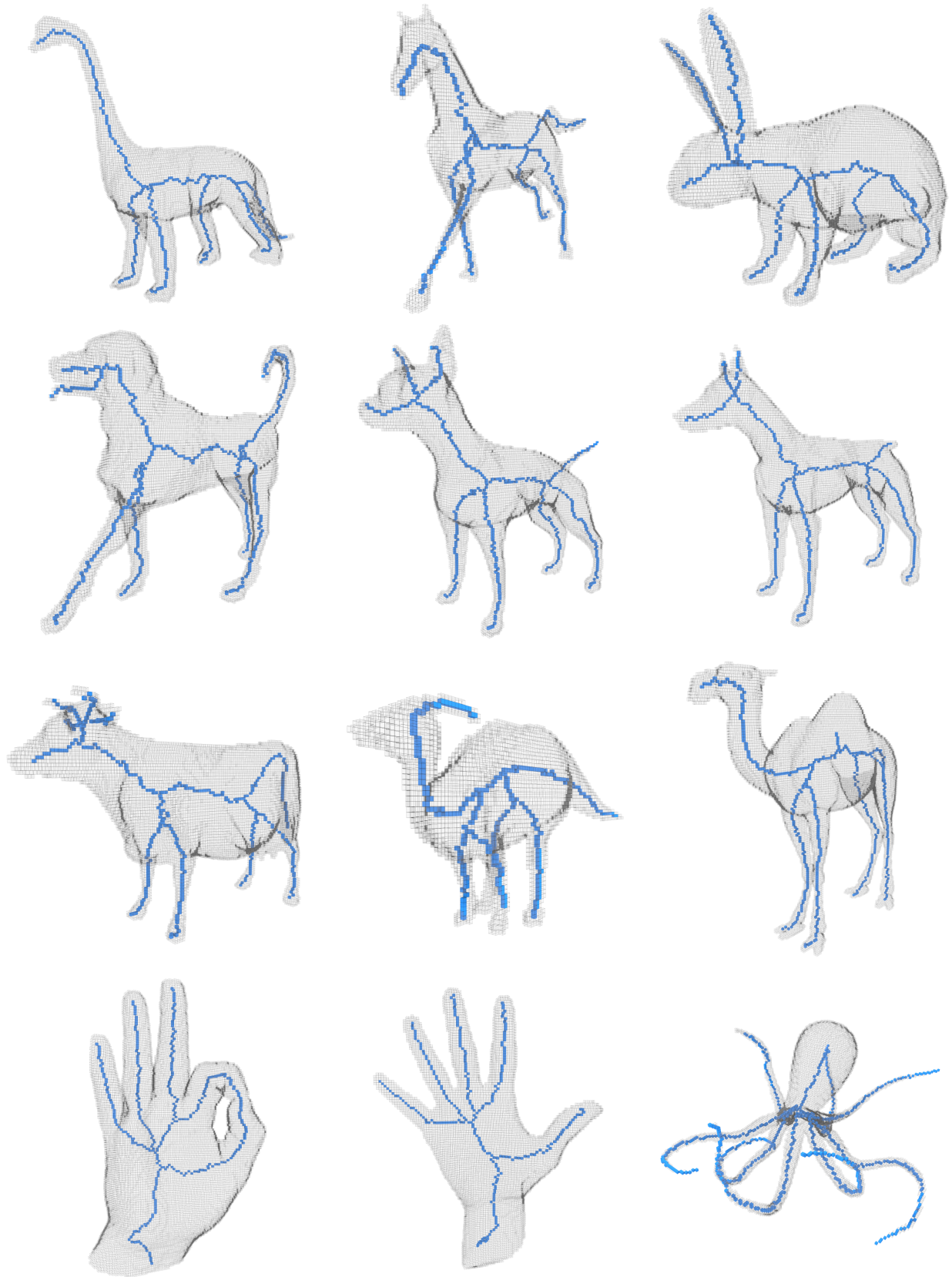


Figura 4.10: Resultados del algoritmo de Arcelli et al. para los volúmenes de prueba con $\theta_1 = 7$ y $\theta_2 = 0.25$.

Capítulo 5

Comparación

En este capítulo se presentan las métricas de comparación de algoritmos de cálculo del *skeleton* propuestas. Posteriormente, los resultados de las implementaciones descritas en los capítulos anteriores son evaluados y comparados utilizando estas métricas. Además de los algoritmos implementados para esta tesis, la implementación del algoritmo de contracción de mallas de Au et al. [9] descrita en los trabajos de Alcayaga y Rojas [4,73] también es evaluada.

La comparación se efectúa en datos de tres dominios distintos: volúmenes de figuras estándar, volúmenes que simulan la forma de figuras biológicas y volúmenes de figuras biológicas reales, neuronas y retículos endoplasmáticos, obtenidos mediante microscopía óptica. Los volúmenes simulados son de particular interés, puesto que su construcción consiste en “inflar” un *skeleton* generado. De esta forma, permiten medir la desviación de cada algoritmo respecto del valor ideal para cada métrica.

5.1. Métricas de comparación

Las métricas propuestas pueden dividirse en dos categorías según su origen teórico. En primer lugar están las métricas de grafos, que surgen de interpretar el *skeleton* como una estructura compuesta por nodos y aristas. Por otro lado están las métricas que buscan capturar aspectos geométricos del *skeleton* como objeto tridimensional.

En cada aplicación del *skeleton* solamente algunas de estas métricas pueden ser relevantes. En el caso de las aplicaciones que motivan esta tesis, el histograma de ángulos de bifurcación solamente es relevante en el análisis morfológico cuantitativo de neuronas, mientras que el número de ciclos inducidos solamente es relevante en el análisis morfológico cuantitativo de retículos endoplasmáticos. El resto de las métricas propuestas son relevantes para ambas aplicaciones, como se describe a continuación.

5.1.1. Métricas de grafos

Precisando lo señalado anteriormente, a partir del *skeleton* de un objeto puede construirse un grafo con pesos no dirigido [10], como se ilustra en la Figura 5.1. En el grafo construido, cada nodo corresponde a un vóxel de término o a una componente 26-conexa de vóxeles de ramificación (vóxeles con más de dos 26-vecinos), y cada arista representa la existencia de un 26-camino entre dos nodos. El peso de cada arista corresponde al largo del 26-camino correspondiente, calculado utilizando una métrica pseudo-euclidiana donde la distancia entre dos vóxeles 26-vecinos es la distancia euclidiana entre sus índices dentro del volumen. Cuando el nodo corresponde a una componente conexa de vóxeles de ramificación (marcadas en azul en la Figura 5.1), la distancias se miden desde el o los vóxeles con la mayor cantidad de vecinos.

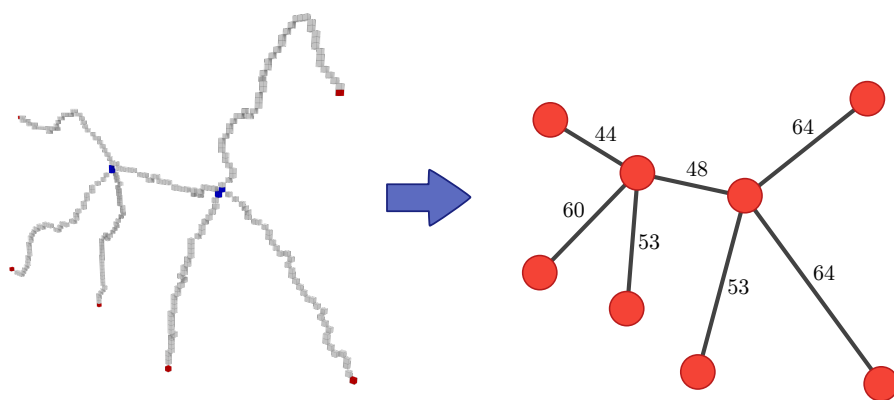


Figura 5.1: Ejemplo de construcción de grafo a partir del *skeleton* de un volumen genérico. Los pesos del grafo representan la distancia entre cada par de nodos en el *skeleton*.

En estructuras biológicas, esta representación del *skeleton* se utiliza para calcular las siguientes métricas:

- Número de nodos

Corresponde al total de nodos del grafo. En el análisis morfológico cuantitativo de neuronas, el número de nodos de terminación (nodos de grado 1) y de ramificación (nodos de grado 3) caracterizan la topología de las estructuras [24,40]. En el caso de los retículos endoplasmáticos, representa un indicador directo de la formación de la red [69].

- Número de aristas

Corresponde al total de aristas del grafo, es decir, el número de conexiones entre nodos.

- Largo total

Es la suma de los largos de todas las conexiones entre nodos, que se calcula sumando los pesos del grafo. Es uno de los principales indicadores de tamaño en estudios morfológicos de neuronas y retículos endoplasmáticos [40,68].

- Número de ciclos inducidos

Es el número de ciclos del grafo que satisfacen la propiedad de que todo nodo dentro del ciclo no está conectado con algún nodo fuera del ciclo. Las neuronas son estructuras acíclicas, por lo que esta métrica no se utiliza en su estudio. En contraste, los retículos endoplasmáticos son redes altamente cíclicas, representando el número de ciclos inducidos una métrica importante en el estudio de su morfología [106].

- Histograma de largos

Corresponde a la distribución de los pesos del grafo. Esta distribución es utilizada tanto en el estudio morfológico de neuronas como de retículos endoplasmáticos para cuantificar regiones donde el largo representa alguna propiedad biológica de interés [24,69].

- Histograma de grados

Corresponde a la distribución del conteo de aristas que contiene a cada nodo. Como se ha señalado anteriormente, en el estudio de neuronas interesa contar el número de nodos de grado 3 (bifurcaciones) y nodos de grado 1 (terminaciones) [24]. El caso del retículo endoplasmático es similar.

5.1.2. Métricas geométricas

Otras métricas adicionales

- Histograma de ángulos de bifurcación

En el estudio de neuronas es de particular interés medir los ángulos de bifurcación de la estructura. Los ángulos se miden recorriendo la estructura jerárquicamente, comenzando desde la región de la neurona denominada soma. El soma es el cuerpo celular de la neurona, desde donde se originan sus ramificaciones. Por lo tanto, medir los ángulos exige la orientación de la estructura, por lo que no tiene sentido práctico calcular esta métrica para objetos sin una raíz definida.

- Índice de reconstructibilidad

Esta métrica no tiene una relación directa con el análisis morfológico cuantitativo, sino que busca medir la centralidad del *skeleton* calculado por cada algoritmo [8]. Corresponde a la razón entre el número de vóxeles del objeto original y el número de vóxeles de su reconstrucción a partir del *skeleton* y la transformada de distancia euclidiana. La reconstrucción corresponde a la unión de las esferas centradas en cada vóxel del *skeleton* de radio igual al valor en la posición correspondiente de la transformada de distancia. Un *skeleton* es más centrado cuanto más cercano a 1 sea su índice de reconstructibilidad.

5.2. Comparación en volúmenes estándar

5.2.1. Especificación de los datos

A lo largo de esta tesis se utilizaron volúmenes de figuras sencillas para ilustrar los resultados de los algoritmos. Estos modelos provienen de bases de datos de objetos estándar, puestas a disposición en Internet por la comunidad de computación gráfica. La obtención y procesamiento de estos modelos se detalla en el Apéndice B. El tipo de dato final de estos objetos de prueba corresponde a volúmenes binarios de $128 \times 128 \times 128$ vóxeles.

5.2.2. Resultados

Las Figuras 5.2 y 5.3 muestran los *skeletons* calculados por cada algoritmo para el conjunto de volúmenes de prueba. La primera columna corresponde a los resultados del algoritmo de Palágyi y Kuba [63], la segunda a los del algoritmo de Siddiqi et al. [86] con $\tau = -15$ y la tercera a los del algoritmo de Arcelli et al. [8] con $\theta_1 = 7$ y $\theta_2 = 0.25$. La cuarta columna corresponde a los resultados de la implementación de Iván Rojas [73] del algoritmo de Au et al. [9]. Como este último algoritmo opera sobre mallas geométricas, es necesario transformar cada volumen a esa representación para poder calcular su *skeleton*. Para esta transformación de vóxeles a mallas se utilizó el programa de procesamiento de imágenes ImageJ [1] y el programa de corrección de mallas no conformes MeshFix.

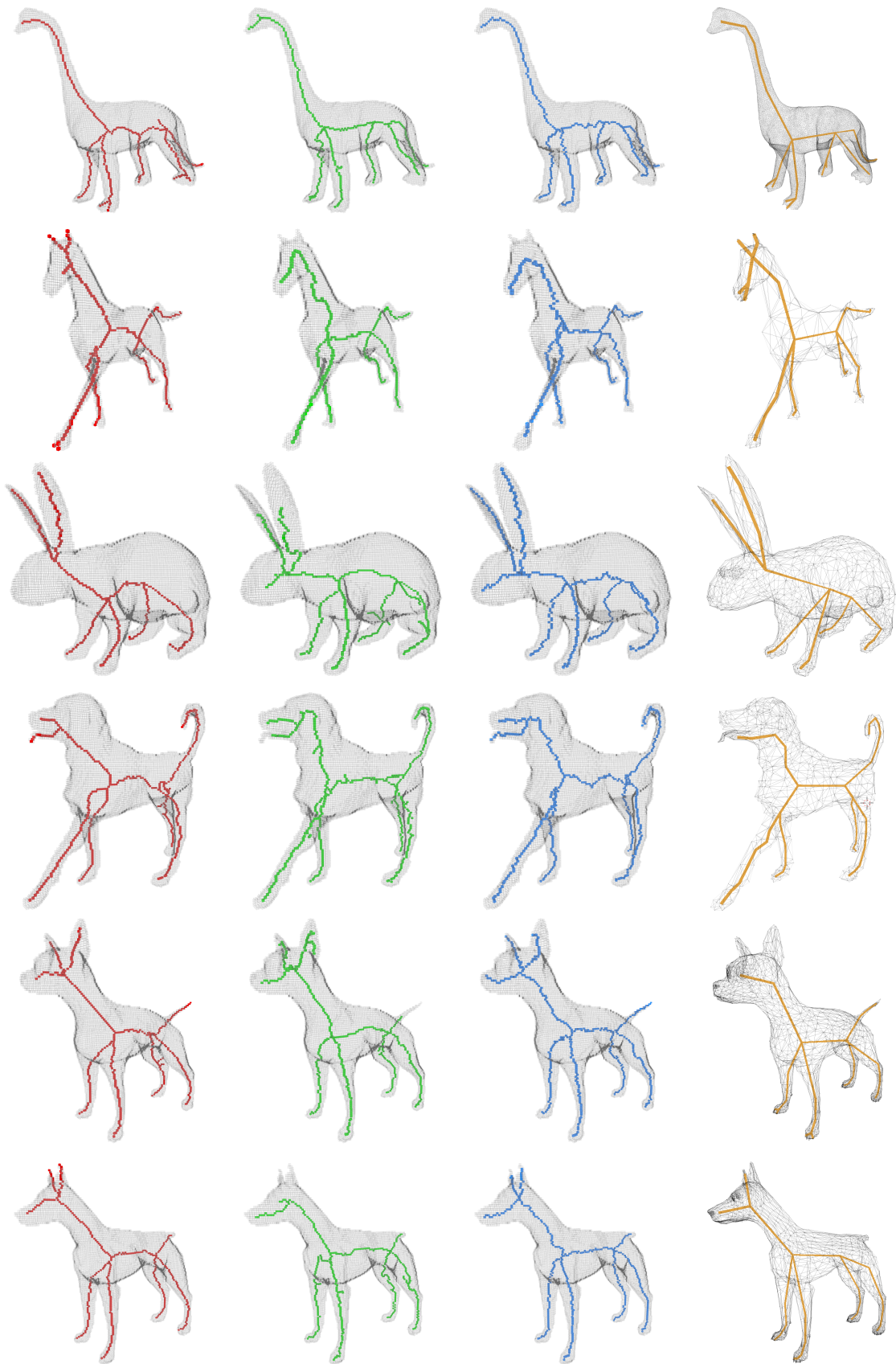


Figura 5.2: Comparación visual de los resultados para modelos de la base de datos Princeton Shape Benchmark.

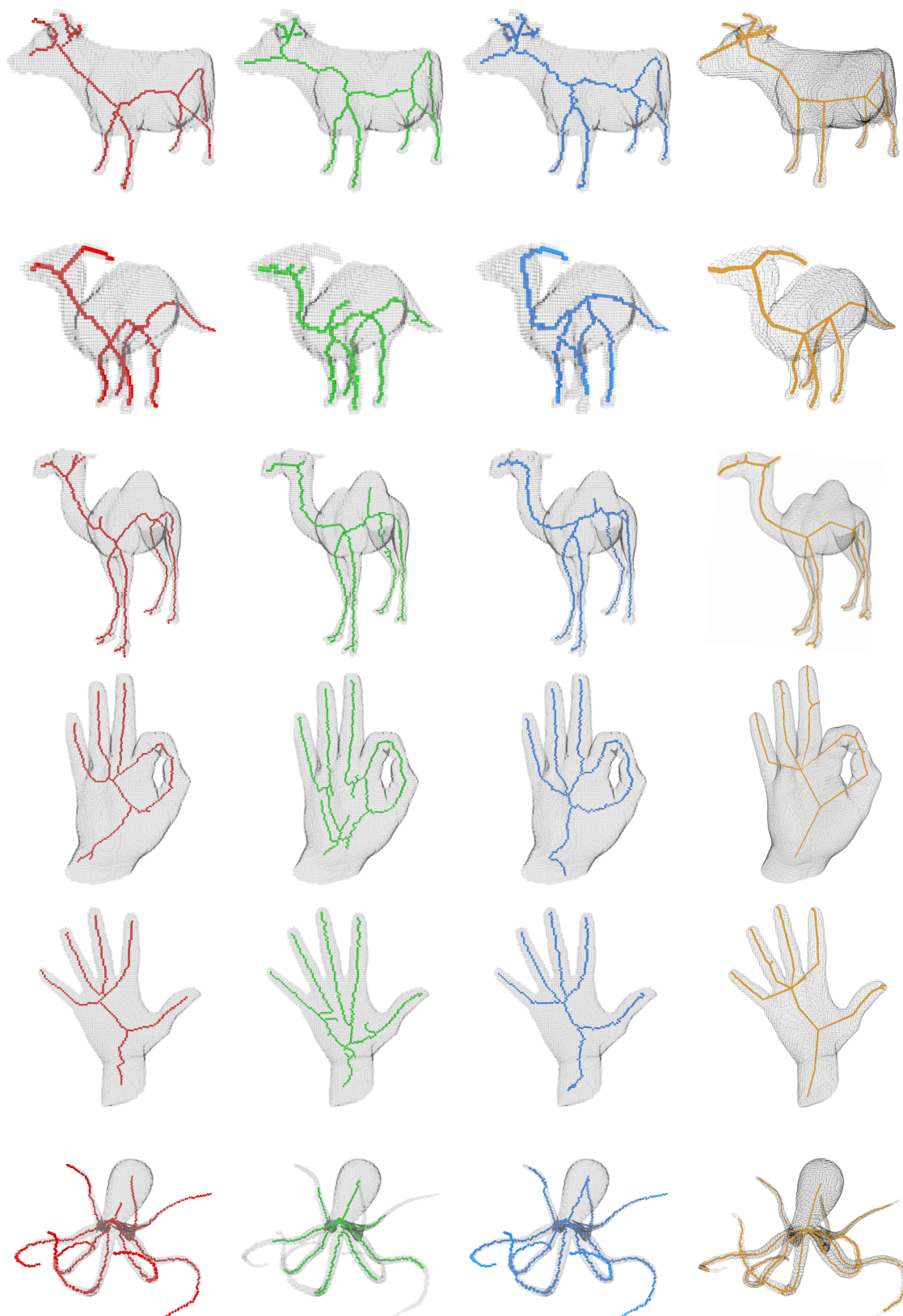


Figura 5.3: Comparación visual de los resultados para modelos de la base de datos Aim@Shape.

Para brindar una primera noción del análisis que es posible efectuar a partir de las métricas, estas fueron calculadas para los *skeletons* de los modelos “conejo” y “mano”. El parámetro del algoritmo de Siddiqi et al. [86] fue optimizado visualmente para cada modelo ($\tau = -15$ para “conejo” y $\tau = -11,55$ para “mano”). En el caso del algoritmo de Arcelli et al. [8], los parámetros $\theta_1 = 7$ y $\theta_2 = 0.25$ produjeron un *skeleton* satisfactorio. Los resultados se presentan a continuación.

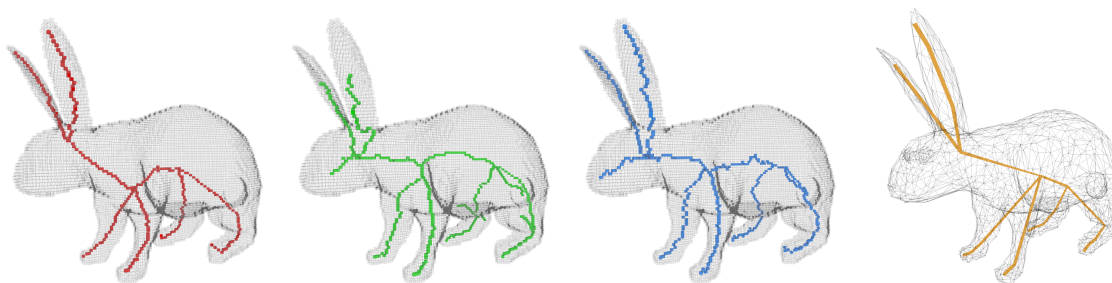


Figura 5.4: Comparación visual de los resultados para el modelo “conejo”.

| | Palágyi y Kuba | Siddiqi et al. | Arcelli et al. | Au et al. |
|-------------------------------|----------------|----------------|----------------|-----------|
| Número de nodos | 10 | 18 | 11 | 9 |
| Número de aristas | 9 | 17 | 10 | 8 |
| Largo total | 373,5 | 484,6 | 470,5 | 419,5 |
| Número de ciclos inducidos | 0 | 0 | 0 | 0 |
| Índice de reconstructibilidad | 0.63 | 0.71 | 0.83 | 0.58 |

Tabla 5.1: Métricas escalares para el modelo “conejo”.

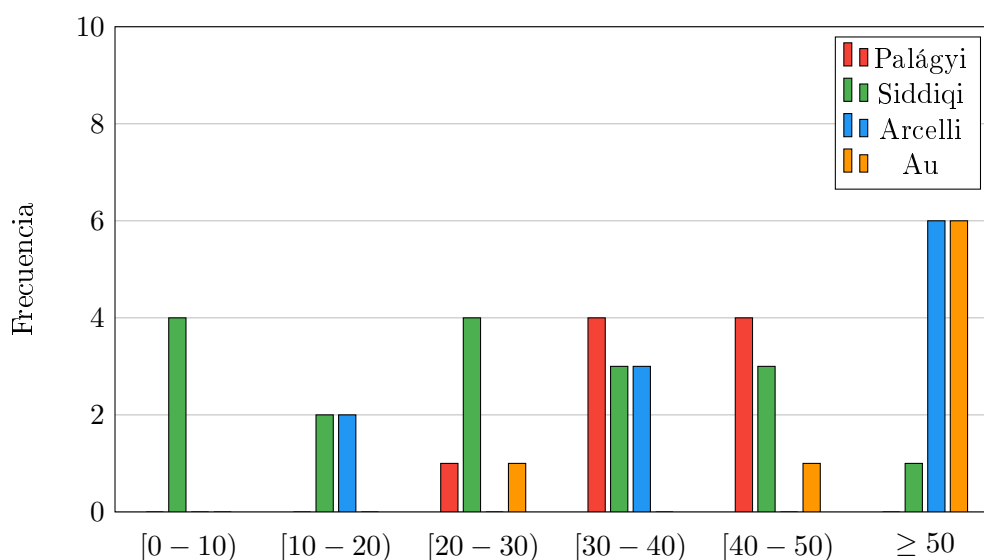


Figura 5.5: Histograma de largos para el modelo “conejo”.

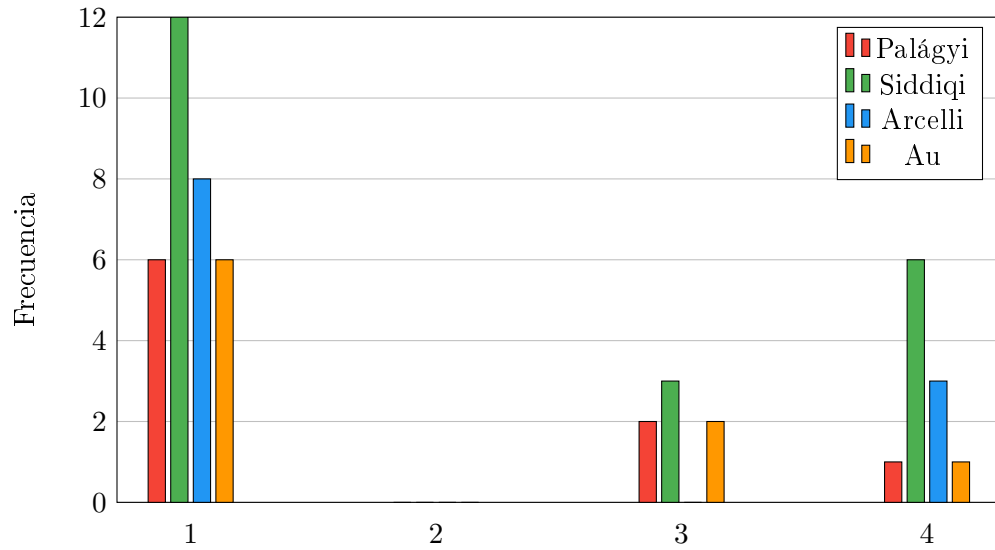


Figura 5.6: Histograma de grados para el modelo “conejo”.

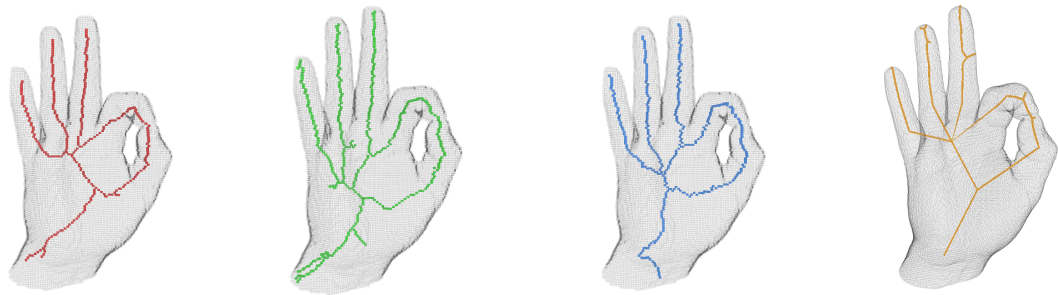


Figura 5.7: Comparación visual de los resultados para el modelo “mano”.

| | Palágyi y Kuba | Siddiqi et al. | Arcelli et al. | Au et al. |
|-------------------------------|----------------|----------------|----------------|-----------|
| Número de nodos | 17 | 22 | 12 | 16 |
| Número de aristas | 17 | 22 | 12 | 16 |
| Largo total | 439,7 | 533,6 | 521,8 | 465,0 |
| Número de ciclos inducidos | 1 | 1 | 1 | 1 |
| Índice de reconstructibilidad | 0.71 | 0.84 | 0.85 | 0.68 |

Tabla 5.2: Métricas escalares para el modelo “mano”.

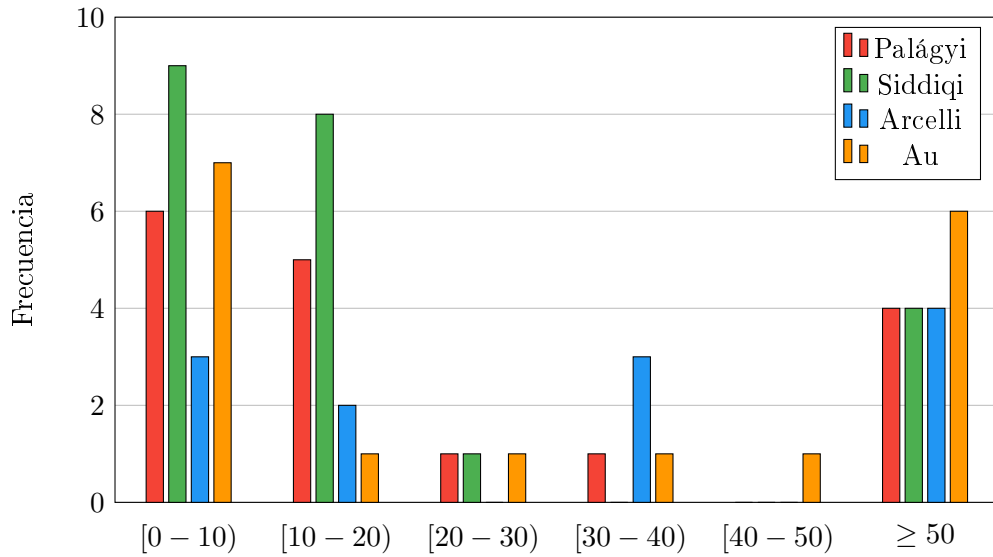


Figura 5.8: Histograma de largos para el modelo "mano".

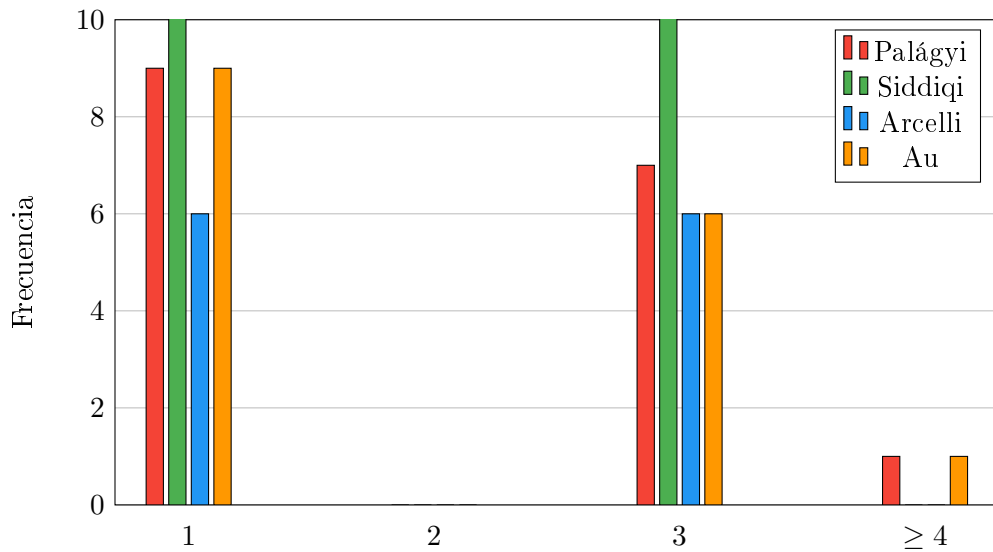


Figura 5.9: Histograma de grados para el modelo "mano".

5.2.3. Discusión

Existen diferencias significativas en varias de las métricas calculadas. En particular, para ambos modelos existe más de un 20 % de diferencia para el largo total entre los algoritmos de Palágyi y Kuba [63] y Siddiqi et al. [86], al igual que entre Palágyi y Kuba [63] y Arcelli et al. [8]. El número de nodos y el número de aristas también varían significativamente, viéndose aumentados casi al doble comparando los algoritmos de Arcelli et al. [8] y Siddiqi et al. [86].

Analizar el histograma de grados del modelo "conejo" revela que los algoritmos de Palágyi y Kuba [63] y Au et al. [9] calculan un *skeleton* de conectividad similar. Sin embargo, la conectividad

de los *skeletons* calculados por los otros dos algoritmos difiere. En particular, el algoritmo de Siddidi et al. [86] tiende a favorecer la aparición de segmentos pequeños en el *skeleton* en comparación con los demás algoritmos. Esto se ve reflejado en el mayor número nodos de grado 4, pero también en el mayor número de nodos y aristas y en el histograma de largos concentrado a la izquierda para ambos modelos.

5.3. Comparación en volúmenes simulados

5.3.1. Especificación de los datos

Los volúmenes simulados se generaron utilizando dos rutinas de MATLAB, una para neuronas y otra para retículos endoplasmáticos. Estas rutinas no forman parte de este trabajo de tesis, sino que fueron desarrolladas para un trabajo anterior [101]. Además de producir el dato generado en formato TIFF, cada rutina produce un archivo de texto con las métricas ideales para el *skeleton* de la estructura. Como se ha mencionado anteriormente, esto es posible gracias a que ambos tipos de estructuras se generan a partir de un *skeleton* perfecto.

La generación de neuronas se realiza construyendo una estructura ramificada a partir de una raíz. Esta raíz consiste en un segmento de 60 unidades de largo. Este segmento raíz es extendido bifurcando uno de sus extremos, dando origen a dos segmentos de 30 unidades de largo cada uno. Luego, cada uno de estos segmentos a su vez se bifurca desde su extremo libre, produciendo un segundo nivel de segmentos de 30 unidades de largo. El proceso de bifurcación se repite 5 veces en total, y está determinado por ángulos de bifurcación seleccionados mediante sorteos en rangos fijos. El resultado de este proceso es un *skeleton* que siempre tiene 31 nodos y 32 segmentos (1 segmento de largo 60 y 31 segmentos de largo 30). Este *skeleton* es luego dilatado utilizando un *kernel* cúbico de lado 5, como se ilustra en la Figura 5.10, proceso que origina juntas inexistentes entre segmentos, fenómeno similar a la superposición producida al capturar imágenes en el microscopio debido a limitaciones de resolución. La salida de este proceso es una serie de 151 imágenes binarias de 500×500 píxeles, que al ser apiladas permiten reconstruir tridimensionalmente la neurona.

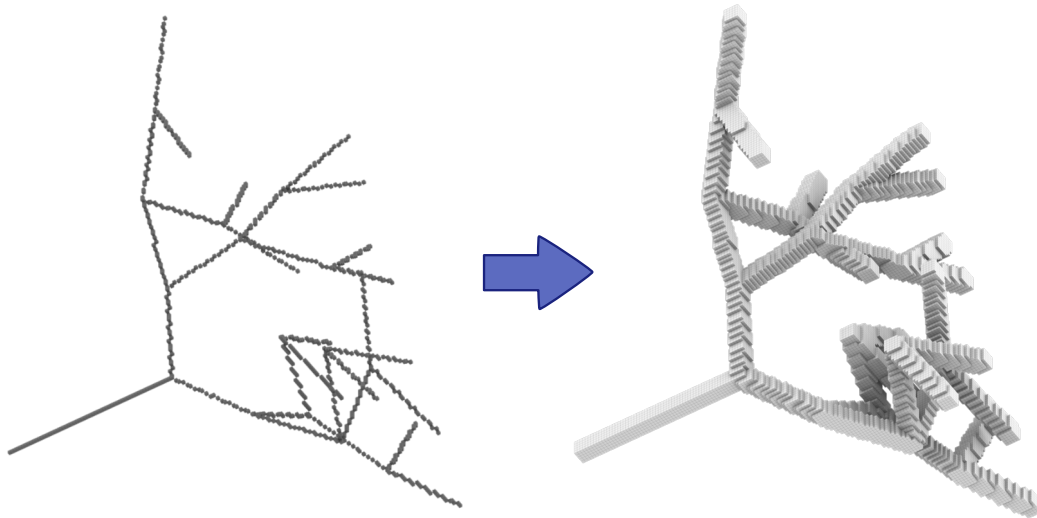


Figura 5.10: Generación de una neurona simulada a partir de un *skeleton* perfecto.

La generación de cada retículo endoplasmático consiste en sembrar 10 puntos en coordenadas aleatorias y conectarlos con las líneas correspondientes a su triangulación de Delaunay. Las líneas de esta triangulación conforman el *skeleton* ideal para la estructura. A diferencia del generador de neuronas, el generador de retículos endoplasmáticos produce una única imagen binaria de 1000×1000 píxeles. Esto se condice con el hecho de que para el estudio morfológico de retículos endoplasmáticos comúnmente se eligen células donde este organelo presenta una geometría plana [106]. La Figura 5.11 muestra la generación de un retículo endoplasmático mediante la rutina descrita.

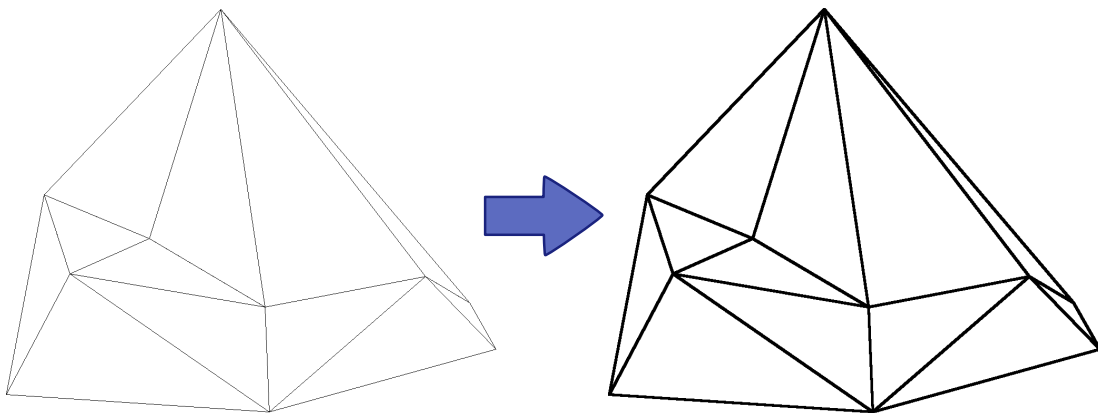


Figura 5.11: Generación de un retículo endoplasmático simulado a partir de un *skeleton* perfecto.

5.3.2. Resultados para neuronas generadas

Utilizando la rutina para generar neuronas descrita anteriormente, se generó un conjunto de datos consistente en 10 volúmenes. Luego, se calculó el *skeleton* de cada neurona utilizando las cuatro implementaciones y las métricas se calcularon para estos 40 *skeletons*. El resto de esta sección presenta los resultados agregados por algoritmo para cada métrica.

Los argumentos para los algoritmos parametrizados fueron esta vez fijados observando los resultados para una estructura del conjunto de datos, estimando demasiado costosa una optimización caso a caso. Para el algoritmo de Siddiqi et al. [86] se fijó $\tau = -19$, mientras que para el algoritmo de Arcelli et al. [8] se fijó $\theta_1 = 11$ y $\theta_2 = 0.25$.

En la Figura 5.12 se presentan en un diagrama de caja los resultados para la primera métrica: el número de nodos. En todos los diagramas de este tipo se consideraron *outliers* los valores alejados de la mediana en una cantidad mayor a 1,5 veces el rango intercuartílico (la diferencia entre el tercer y el primer cuartil). En el caso de los histogramas, como el de la Figura 5.15, los resultados se presentan agregados para entregar una idea de su distribución.

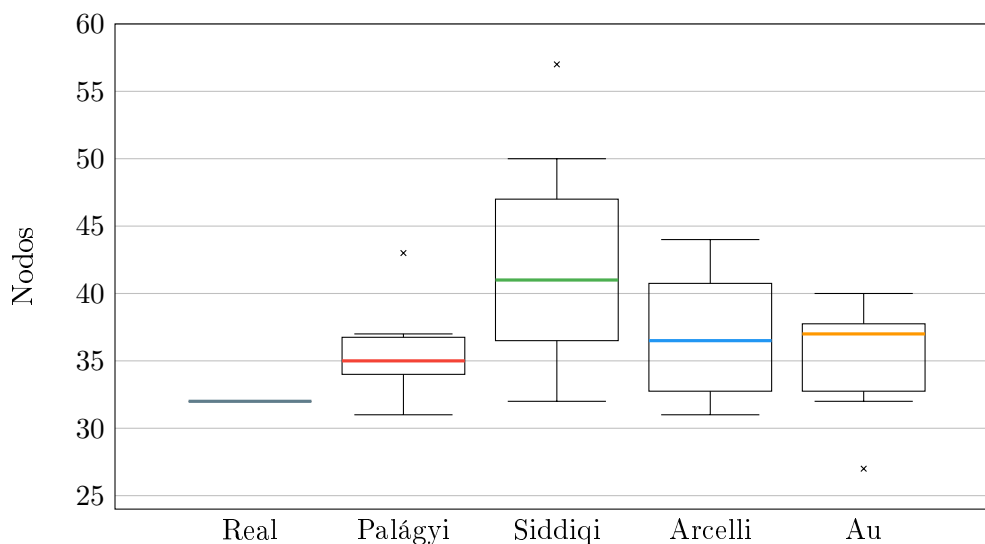


Figura 5.12: Número de nodos en los *skeletons* de neuronas simuladas calculados por cada implementación.

Para medir la significación de la diferencia entre la distribución de cada métrica y su valor ideal se utilizó la prueba no paramétrica de Kruskal-Wallis. Todos los algoritmos producen un *skeleton* con un número de nodos significativamente distinto del valor ideal ($p < 0,05$ en todos los casos, comparando la distribución uniforme ideal con la distribución de cada algoritmo).

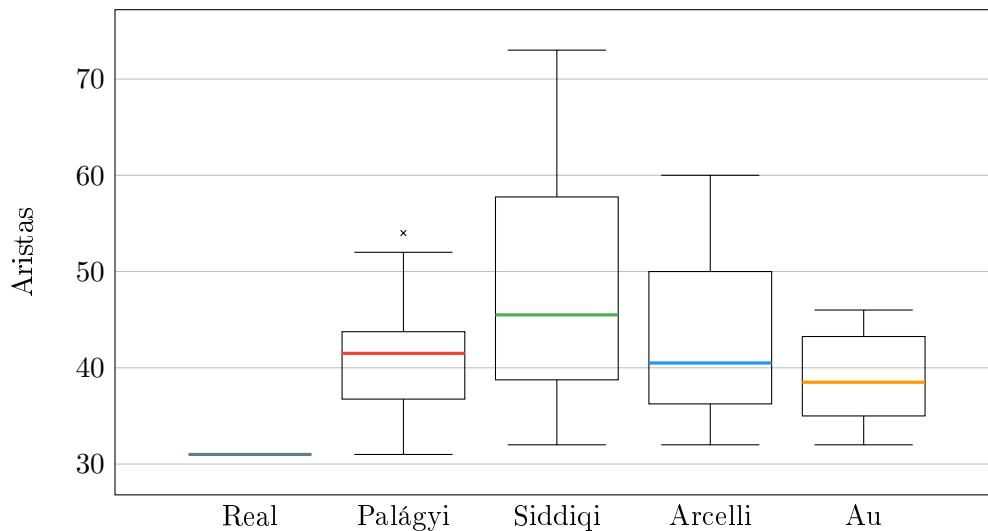


Figura 5.13: Número de aristas en los *skeletons* de neuronas simuladas calculados por cada implementación.

Consecuente con el resultado anterior, la Figura 5.13 muestra que todos los algoritmos producen un *skeleton* con un número de aristas significativamente distinto del valor ideal ($p < 0,05$ en todos los casos, comparando la distribución uniforme ideal con la distribución de cada algoritmo).

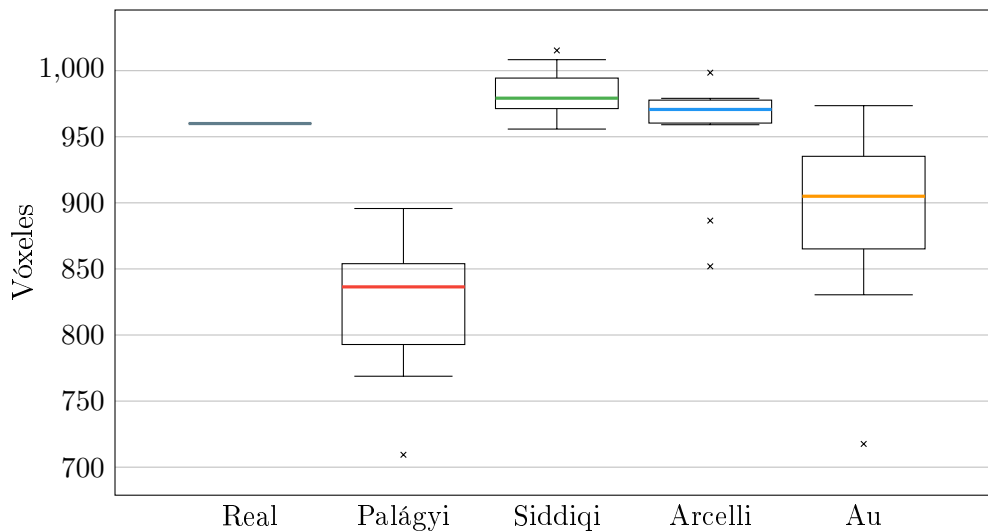


Figura 5.14: Largo total en los *skeletons* de neuronas simuladas calculados por cada implementación.

El algoritmo de Arcelli et al. [8] produce un resultado sin diferencias significativas con el valor ideal ($p = 0,42$ comparando su distribución con la distribución uniforme ideal). El resto de los algoritmos difiere significativamente ($p < 0,05$ comparando la distribución ideal contra todos demás los casos).

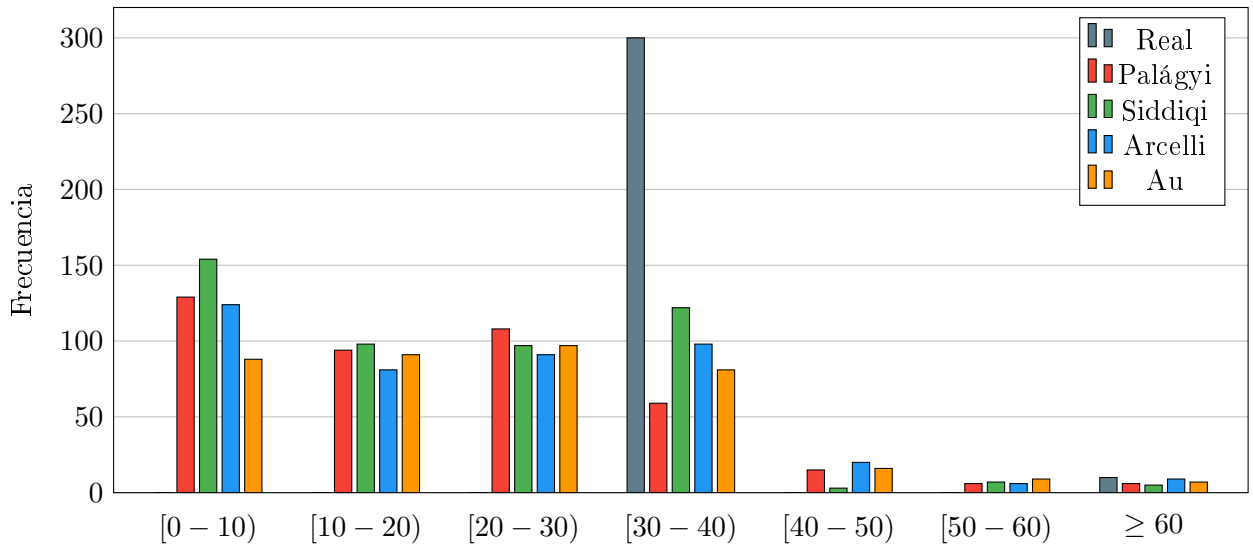


Figura 5.15: Histograma de largos de segmentos en los *skeletons* de neuronas simuladas calculados por cada implementación.

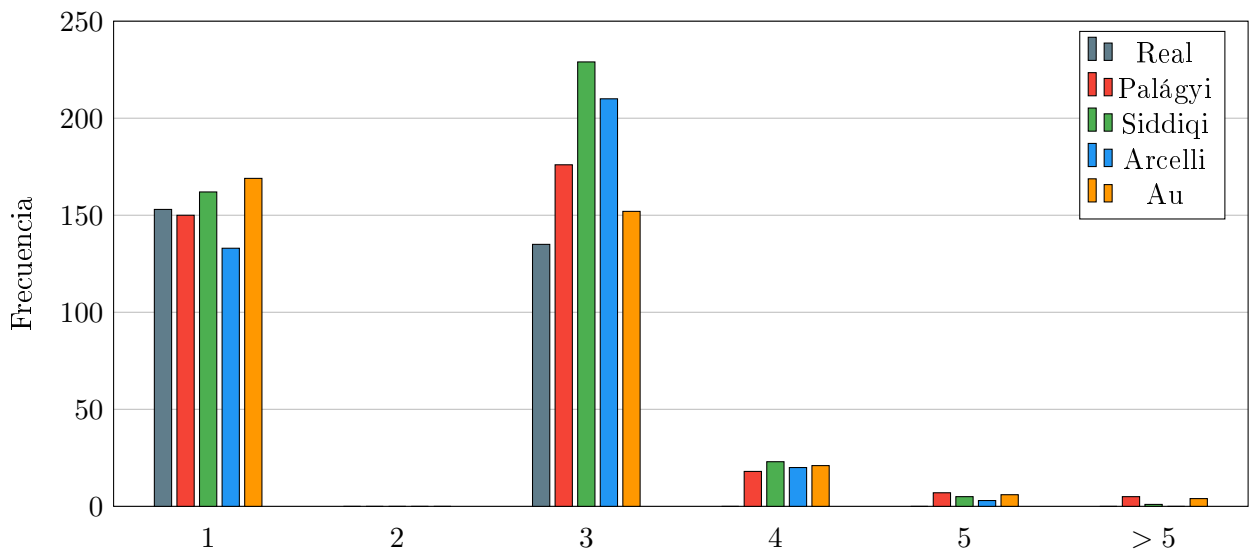


Figura 5.16: Histograma de grados en los *skeletons* de neuronas simuladas calculados por cada implementación.

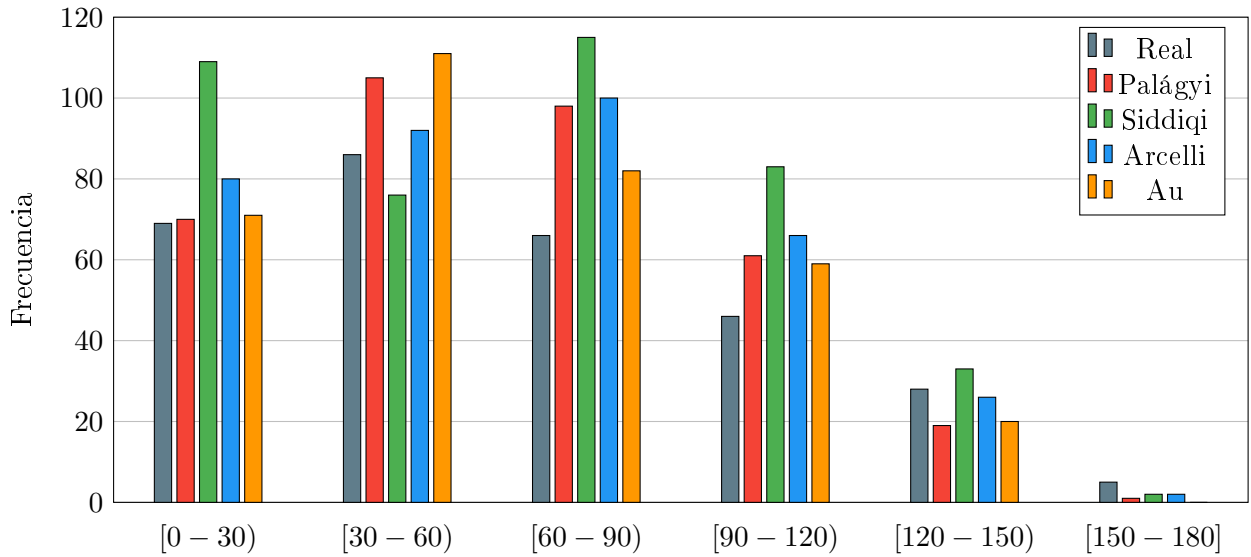


Figura 5.17: Histograma de ángulos en los *skeletons* de neuronas simuladas calculados por cada implementación.

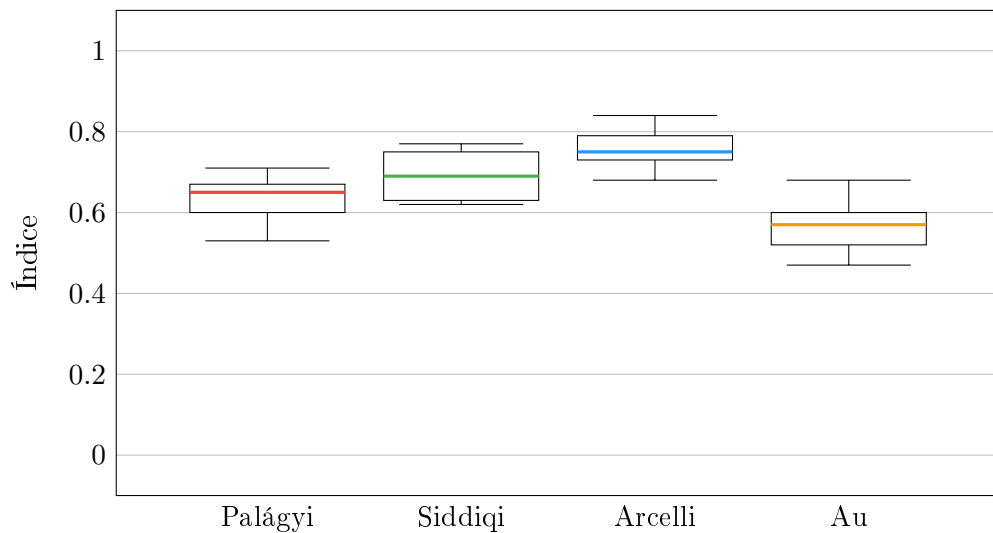


Figura 5.18: Índice de reconstructibilidad en los *skeletons* de neuronas simuladas calculados por cada implementación.

5.3.3. Discusión

Tal como fue el caso para los modelos estándar, el algoritmo de Siddiqi et al. [86] produce un mayor número de nodos y aristas falsas en las neuronas generadas. A pesar de esto, su estimación del largo total es más precisa que la de los algoritmos de Palágyi y Kuba [63] y Au et al. [9], posiblemente debido a que estos algoritmos tienden a producir un *skeleton* demasiado simplificado al ser aplicados sobre figuras de geometría sencilla. Sin embargo, el resultado de Arcelli et al. [8] es el único que produjo resultados estadísticamente satisfactorios para la métrica del largo total.

Por otro lado, la alta ramificación de los resultados del algoritmo de Siddiqi et al. [86] produce además una distorsión significativa en la distribución de los ángulos y un alto número de nodos de grado 3.

5.3.4. Resultados para retículos endoplasmáticos generados

Al igual que para las neuronas, se generaron 10 retículos endoplasmáticos utilizando la rutina descrita anteriormente. Para el algoritmo de Siddiqi et al. [86] se fijó $\tau = -40$, mientras que para el algoritmo de Arcelli et al. [8] se fijó $\theta_1 = 7$ y $\theta_2 = 0.25$. Los resultados agregados por algoritmo se presentan a continuación.

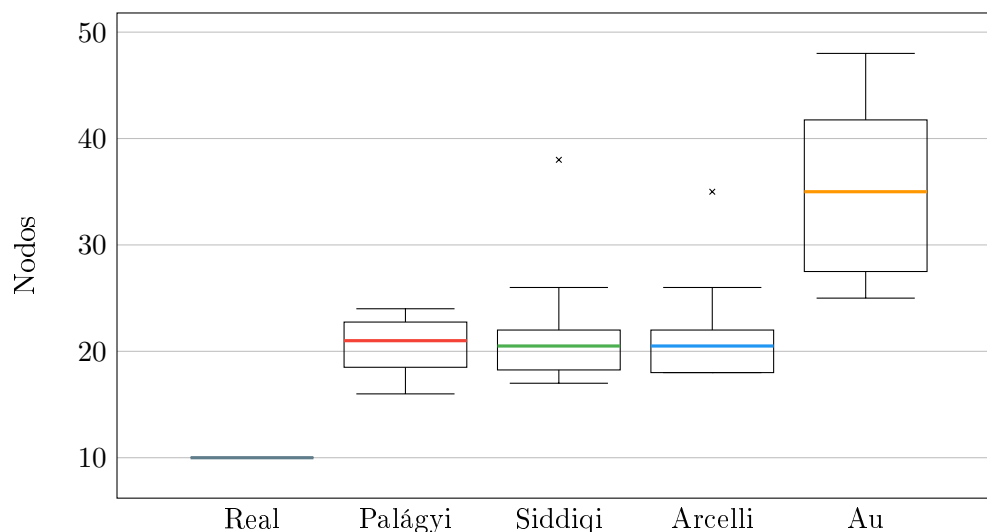


Figura 5.19: Número de nodos en los *skeletons* de retículos endoplasmáticos simulados calculados por cada implementación.

Todos los algoritmos producen un *skeleton* con un número de nodos significativamente distinto del valor ideal ($p < 0,05$ comparando la distribución uniforme ideal con la distribución con cada algoritmo). Sin embargo, los algoritmos de Palágyi y Kuba [63], Siddiqi et al. [86] y Arcelli et al. [8] producen en este caso resultados similares ($p = 0,88$ comparando las tres distribuciones).

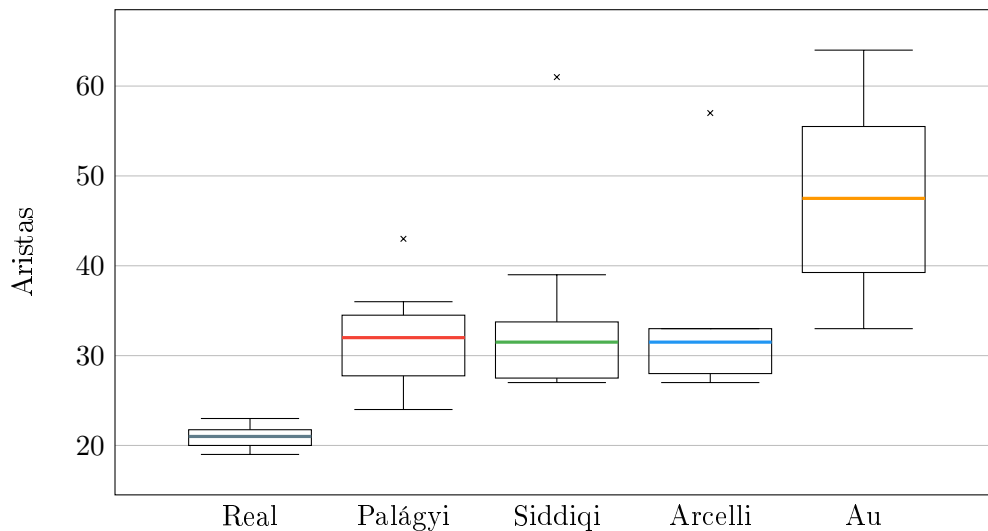


Figura 5.20: Número de aristas en los *skeletons* de retículos endoplasmáticos simulados calculados por cada implementación.

El número de aristas también difiere del real para todos los algoritmos ($p < 0,05$ comparando la distribución ideal con la de cada algoritmo). Sin embargo, entre los algoritmos de Palágyi y Kuba [63], Siddiqi et al. [86] y Arcelli et al. [8] son similares ($p = 0,99$ comparando sus distribuciones).

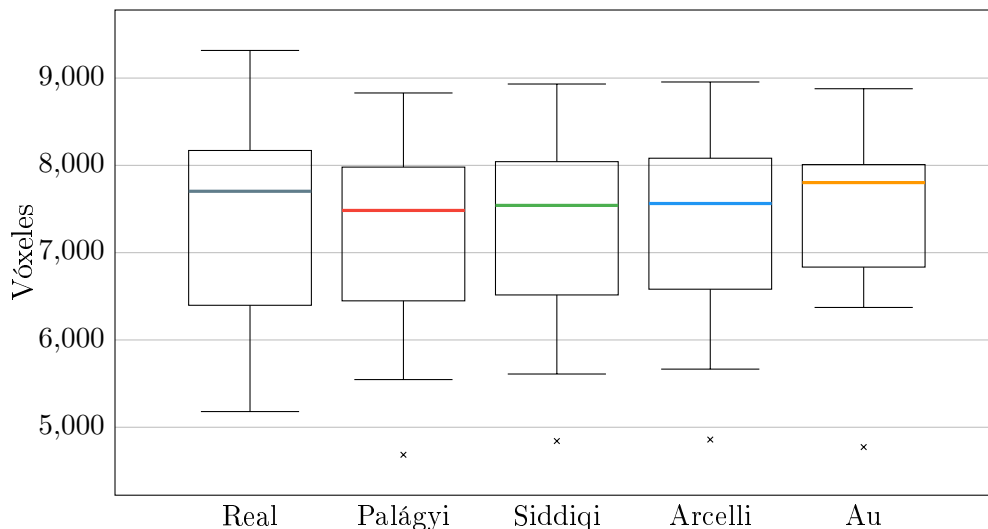


Figura 5.21: Largo total de los *skeletons* de retículos endoplasmáticos simulados calculados por cada implementación.

El largo total resulta con una distribución similar a la real para los 4 algoritmos según la prueba de Kruskal-Wallis (comparando la distribución ideal con la de cada algoritmo se obtiene $p = 0,65$ para Palágyi y Kuba [63], $p = 0,71$ para Siddiqi et al. [86], $p = 0,88$ para Arcelli et al. [8] y $p = 0,93$ para Au et al. [9]).

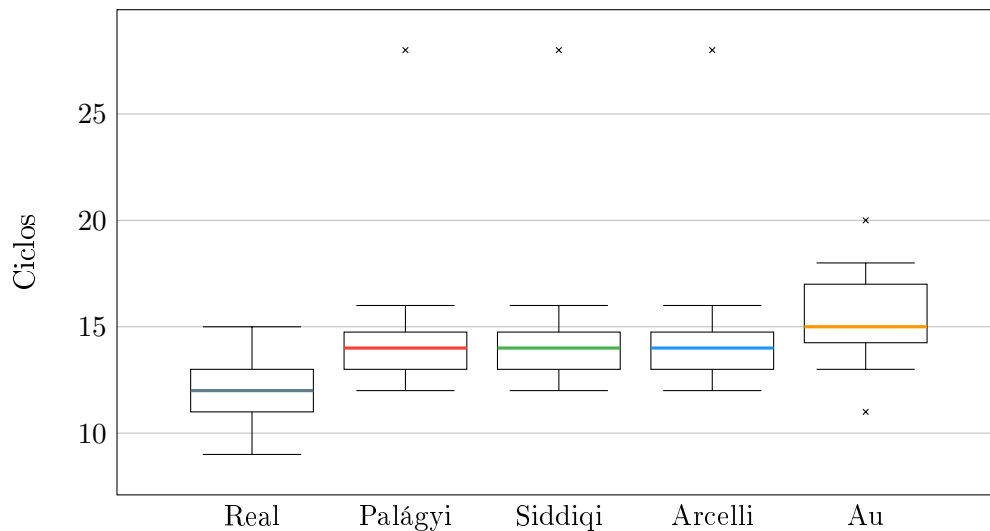


Figura 5.22: Número de ciclos inducidos en los *skeletons* de retículos endoplasmáticos simulados calculados por cada implementación.

El número de ciclos inducidos difiere del real para todos los algoritmos ($p < 0,05$ en todos los casos). Los algoritmos de Palágyi y Kuba [63], Siddiqi et al. [86] y Arcelli et al. [8] calculan exactamente el mismo número de ciclos para cada entrada.

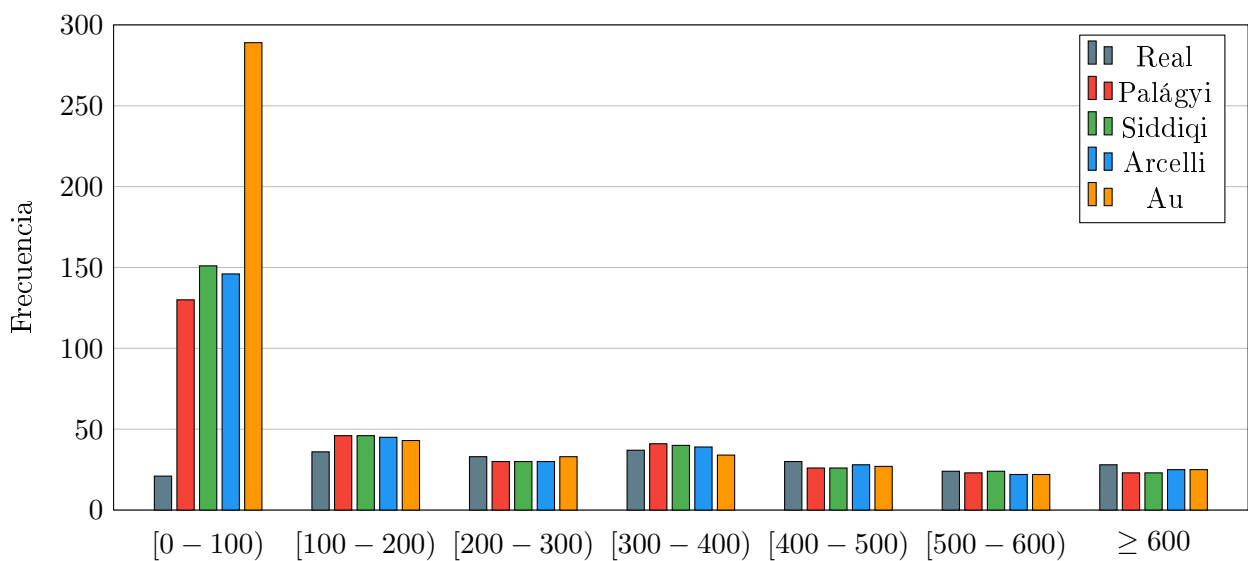


Figura 5.23: Histograma de largos en los *skeletons* de retículos endoplasmáticos simulados calculados por cada implementación.

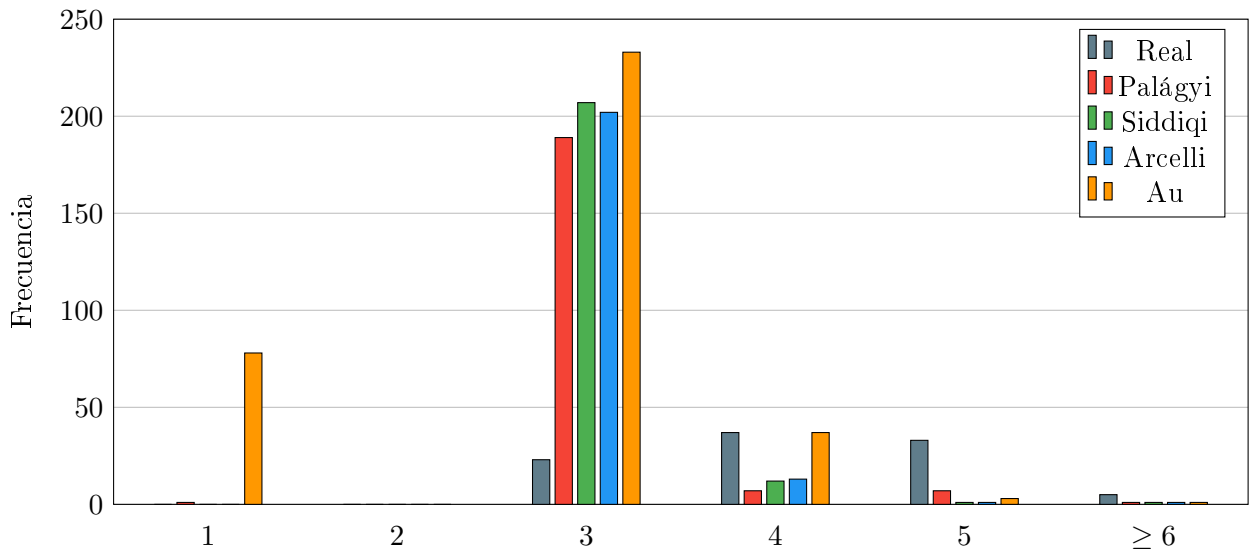


Figura 5.24: Histograma de grados en los *skeletons* de retículos endoplasmáticos simulados calculados por cada implementación.

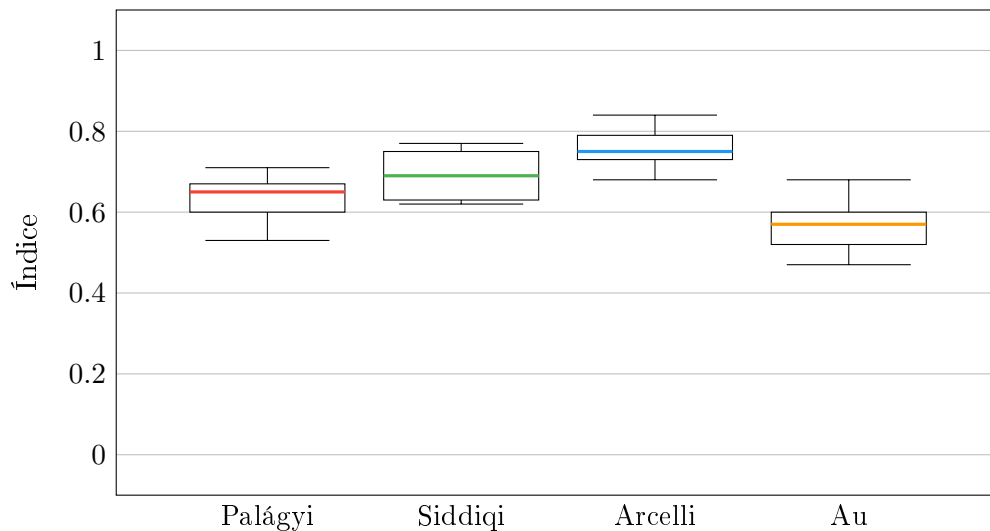


Figura 5.25: Índice de reconstructibilidad en los *skeletons* de retículos endoplasmáticos simulados calculados por cada implementación.

5.3.5. Discusión

En el caso de los retículos endoplasmáticos, los cuatro algoritmos tienen un rendimiento muy similar en cuanto a la preservación del largo total de las estructuras. Sin embargo, el algoritmo de Au et al. [9] calcula un largo ligeramente mayor, lo cual se condice con sus resultados en el resto de las métricas. En particular, el número de ciclos, una cantidad que debería ser exactamente igual para todos los algoritmos debido a la propiedad homotópica, es distinto para el algoritmo de Au et al. [9]. La introducción de ciclos adicionales en los datos tiene su origen en el procesamiento

necesario para calcular el *skeleton* (transformar cada imagen en malla utilizando ImageJ y corregirla con MeshFix) y las métricas (transformar cada malla contraída en una imagen). La necesidad de procesar las imágenes y volúmenes de esta manera menoscaba la usabilidad del algoritmo de Au et al. [9] para las aplicaciones que motivan esta tesis.

5.4. Comparación en volúmenes de biología

5.4.1. Especificación de los datos

Finalmente, con el objetivo de evaluar las métricas en un caso real, se presenta la cuantificación de los *skeletons* de dos volúmenes obtenidos mediante microscopía óptica, correspondientes a una neurona y un retículo endoplasmático, siguiendo los métodos descritos en la Sección 1.2.1. La Figura 5.26 muestra la reconstrucción tridimensional de la neurona segmentada ($1344 \times 1024 \times 305$ vóxeles) y la segmentación del retículo endoplasmático ($672 \times 512 \times 1$ vóxeles).

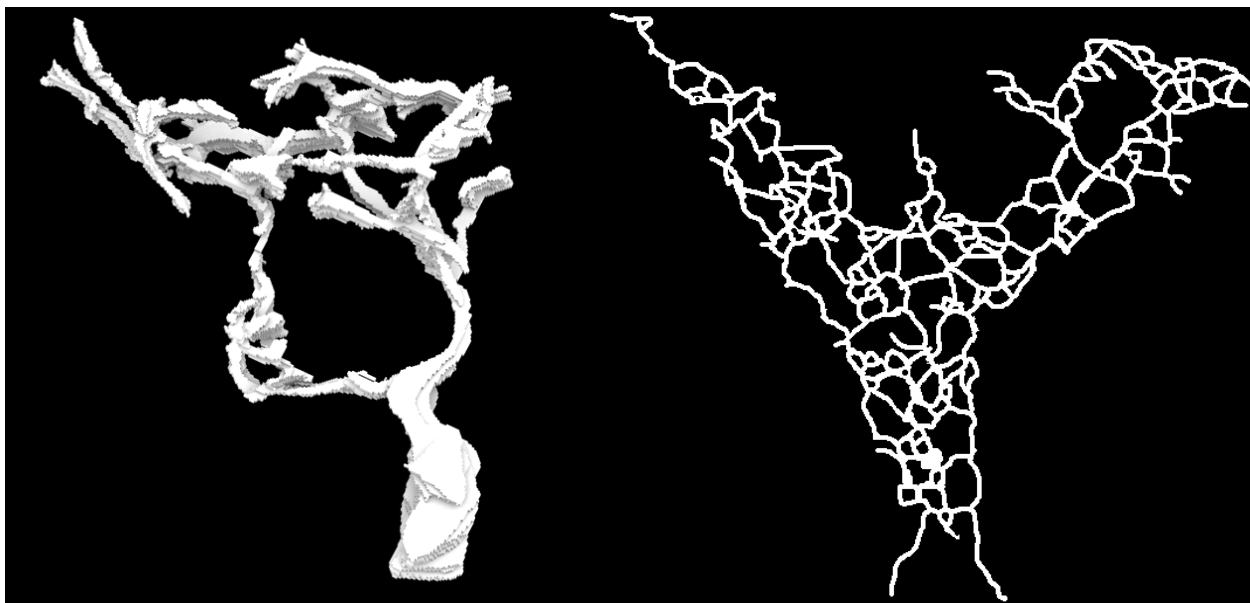


Figura 5.26: Estructuras biológicas segmentadas. A la izquierda, una neurona de pez cebra. A la derecha, un retículo endoplasmático de mono verde.

5.4.2. Resultados

En esta parte se presentan las métricas para la neurona y el retículo endoplasmático de la Figura 5.26. En ambos casos se optimizaron los parámetros de los algoritmos de Siddiqi et al. [86] y Arcelli et al. [8] para obtener el mejor resultado posible. En el caso de la neurona, se utilizó $\tau = -22$ para el algoritmo de Siddiqi et al. [86] y $\theta_1 = 11, \theta_2 = 0.25$ para el algoritmo de Arcelli et al. [8]. Para el retículo se utilizó $\tau = -40$ y $\theta_1 = 7, \theta_2 = 0.25$.

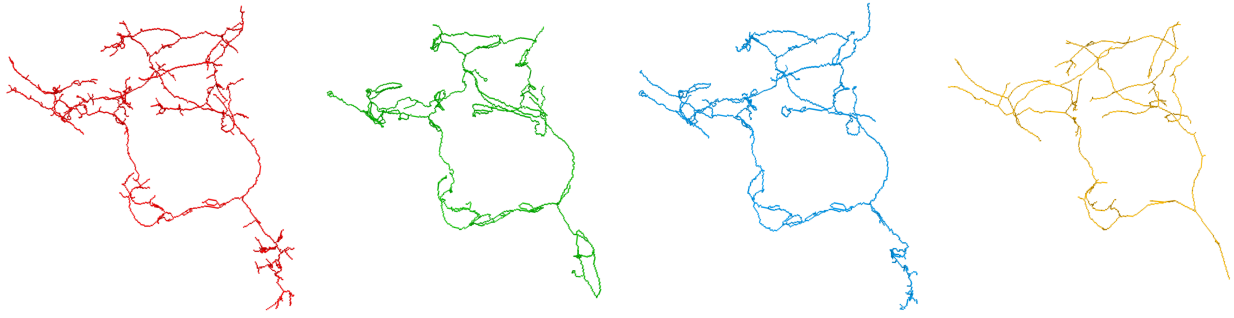


Figura 5.27: Comparación visual de los *skeletons* calculados utilizando cada algoritmo para el mismo volumen, correspondiente a la neurona de la Figura 5.26.

| | Palágyi y Kuba | Siddiqi et al. | Arcelli et al. | Au et al. |
|-------------------------------|----------------|----------------|----------------|-----------|
| Número de nodos | 314 | 91 | 158 | 103 |
| Número de aristas | 350 | 130 | 196 | 106 |
| Largo total | 3796,8 | 3370,5 | 3785,2 | 2960,5 |
| Índice de reconstructibilidad | 0,79 | 0,72 | 0,81 | 0,64 |

Tabla 5.3: Métricas escalares para los *skeletons* de la Figura 5.27 (neurona).

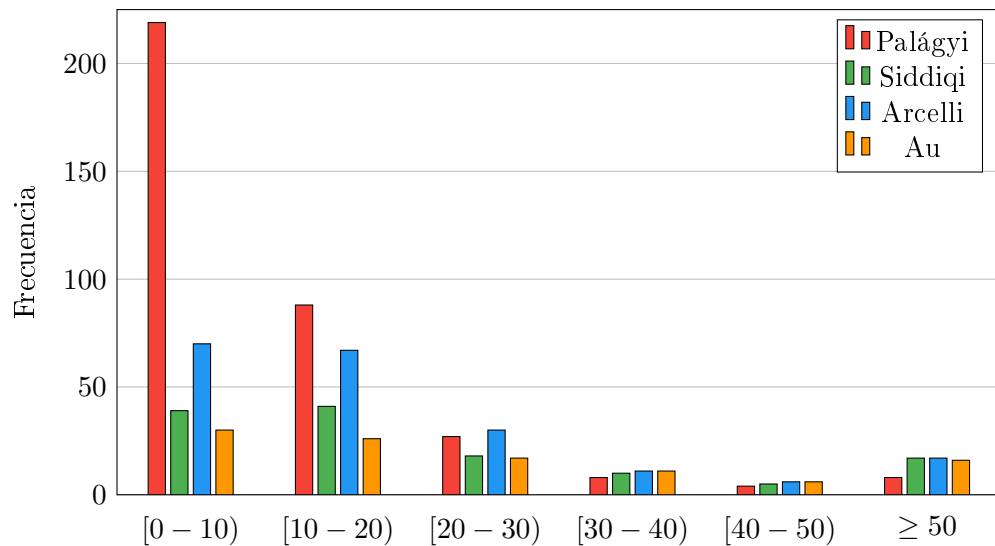


Figura 5.28: Histograma de largos para los *skeletons* de la Figura 5.27 (neurona).

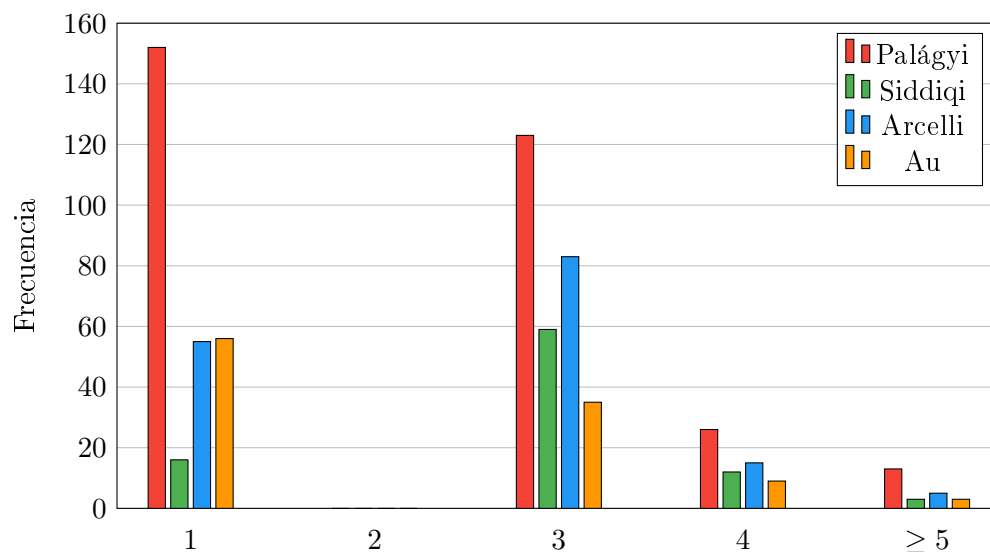


Figura 5.29: Histograma de grados para los *skeletons* de la Figura 5.27 (neurona).

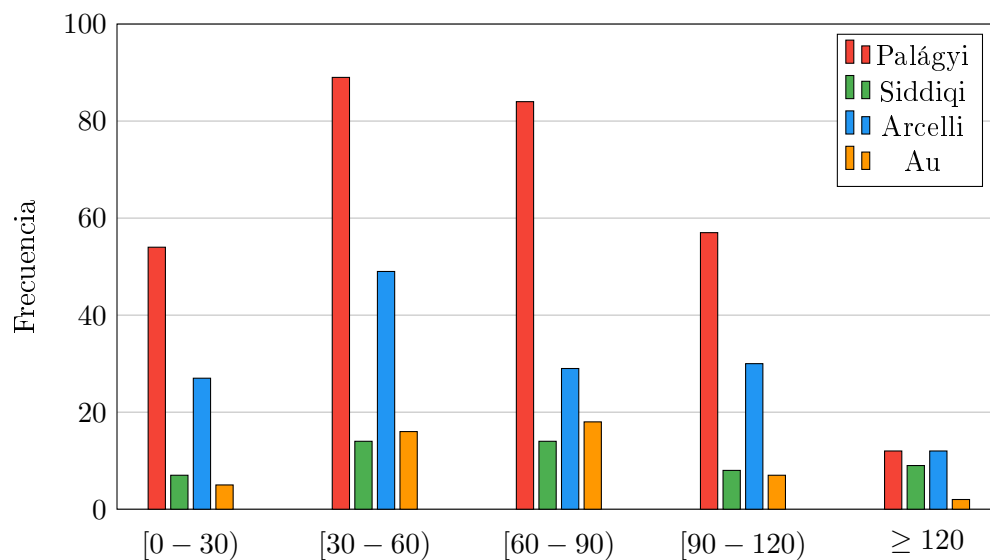


Figura 5.30: Histograma de ángulos de bifurcación para los *skeletons* de la Figura 5.27 (neurona).

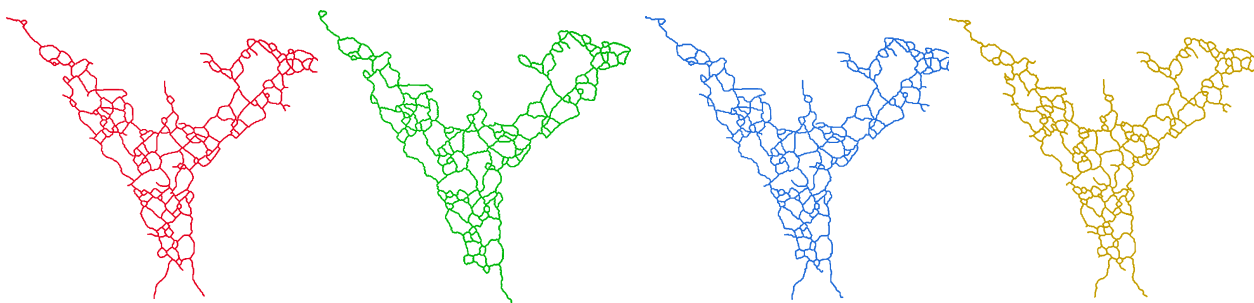


Figura 5.31: Comparación visual de los *skeletons* calculados utilizando cada algoritmo para la misma imagen, correspondiente al retículo endoplasmático de la Figura 5.26.

| | Palágyi y Kuba | Siddiqi et al. | Arcelli et al. | Au et al. |
|-------------------------------|----------------|----------------|----------------|-----------|
| Número de nodos | 263 | 220 | 256 | 283 |
| Número de aristas | 380 | 337 | 373 | 398 |
| Largo total | 6028,5 | 6205,7 | 6422,7 | 6472,1 |
| Número de ciclos inducidos | 120 | 120 | 120 | 119 |
| Índice de reconstructibilidad | 0,89 | 0,76 | 0,93 | 0,82 |

Tabla 5.4: Métricas escalares para los *skeletons* de la Figura 5.31 (retículo endoplasmático).

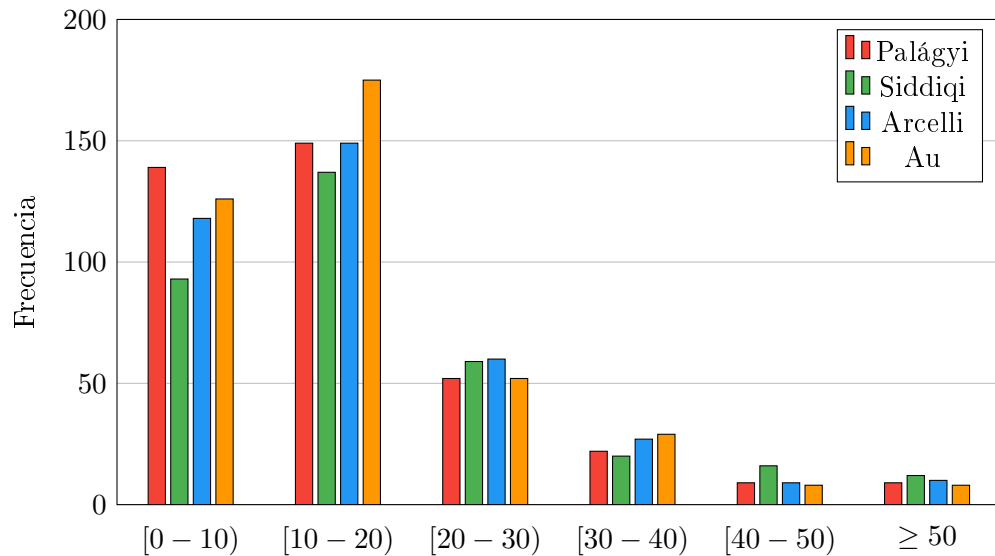


Figura 5.32: Histograma de largos para los *skeletons* de la Figura 5.31 (retículo endoplasmático).

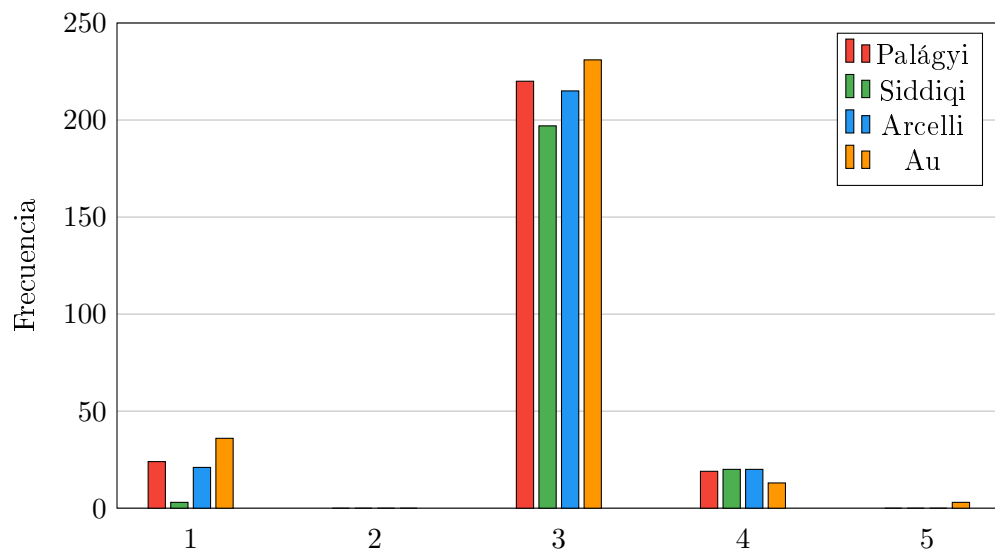


Figura 5.33: Histograma de grados para los *skeletons* de la Figura 5.31 (retículo endoplasmático).

5.4.3. Discusión

La cuantificación de la neurona revela que el algoritmo de Palágyi y Kuba [63] es altamente sensible al ruido. A diferencia de los datos generados y los modelos estándar, el borde de la neurona real presenta perturbaciones para cuya remoción este algoritmo no dispone de criterio alguno. Producto de esto, la métrica del número de nodos del *skeleton* calculado llega a triplicarse en relación a los demás algoritmos. Por otro lado, la parametrización del algoritmo de Siddiqi et al. [86] en definitiva significa un compromiso entre la preservación de ramas y la resistencia al ruido en el cálculo del *skeleton*. El valor para el parámetro τ utilizado en este caso consigue eliminar las ramas poco significativas, como evidencia el histograma de largos, pero el número de nodos y el histograma de grados revela que esto se logra a costa de una simplificación excesiva, donde la mayoría de los nodos de grado 1 han sido removidos. Finalmente, el histograma de grados muestra que los algoritmos de Arcelli et al. [8] y Au et al. [9] producen un resultado que refleja mejor la topología real de la neurona, pero en otras métricas difieren significativamente. El algoritmo de Arcelli et al. [8] parece ser más apropiado para calcular el largo total, mientras que el algoritmo de Au et al. [9] permite calcular los ángulos de manera más fidedigna.

En el caso de los retículos destaca una diferencia contraintuitiva. El algoritmo de Siddiqi et al. [86] nuevamente no preserva tantas aristas como los demás algoritmos, debido a la eliminación excesiva de nodos de grado 1. Sin embargo, el largo total de su *skeleton* supera al de Palágyi y Kuba [63]. Esto se explica por el hecho de que el *skeleton* calculado por el algoritmo de Siddiqi et al. [86] tiende a reproducir mejor la curvatura de las figuras. Esto es evidente al revisar las Figuras 5.2 y 5.3. Fuera de este detalle, el desempeño de los algoritmos es similar.

Capítulo 6

Conclusiones

Esta tesis consistió en implementar tres algoritmos de cálculo del *skeleton* orientados a volúmenes de vóxeles: el algoritmo de adelgazamiento topológico de Palágyi y Kuba [63], el algoritmo basado en la divergencia de Siddiqi et al. [86] y el algoritmo basado en la distancia de Arcelli et al. [8]. Luego, se compararon los resultados de estos algoritmos de acuerdo a métricas relevantes en el área de Biología, incorporando a la comparación la implementación previa de un cuarto algoritmo [9], basado en la contracción de mallas de superficie.

El análisis de las métricas en figuras simuladas y reales reveló que el algoritmo de cálculo del *skeleton* utilizado puede tener una incidencia significativa en mediciones relevantes para el estudio morfológico cuantitativo de neuronas y retículos endoplasmáticos. Con base en los resultados obtenidos, es posible concluir respecto de cada métrica analizada:

- Número de nodos: La distorsión de los datos introduce un error que no permite a ningún algoritmo conservar esta métrica. Los resultados en neuronas simuladas indican que el algoritmo de adelgazamiento topológico (Palágyi y Kuba [63]) permitiría obtener valores más cercanos al ideal, pero esto fue desmentido al calcular las métricas para el caso real, donde obtuvo el peor desempeño. El algoritmo de contracción de mallas (Au et al. [9]) presenta un desempeño superior en el caso real, debido a su elevada resistencia al ruido del contorno de la figura. Por otro lado, para retículos endoplasmáticos las diferencias entre algoritmos son menos significativas en esta métrica, salvo porque el algoritmo basado en la divergencia (Siddiqi et al. [86]) no es recomendable debido a su sensibilidad al valor del parámetro τ .
- Número de aristas: Por motivos similares al caso del número de nodos, ningún algoritmo logra un buen desempeño en esta métrica para figuras simuladas. Para el caso real de neuronas, nuevamente la resistencia al ruido del algoritmo de contracción de mallas (Au et al. [9]) le confiere un desempeño superior. En el caso de retículos endoplasmáticos, los algoritmos produjeron un resultado similar, salvo por el algoritmo basado en la divergencia (Siddiqi et al. [86]). Este

algoritmo produce una poda excesiva de ramas relevantes, por lo que tampoco es recomendable para esta métrica.

- Largo total: Los resultados en neuronas simuladas sugieren que el *skeleton* calculado por el algoritmo basado en la distancia (Arcelli et al. [8]) no difiere significativamente del valor ideal en cuanto a esta métrica. Esto se condice con el hecho de que este algoritmo conserva mejor la curvatura y los puntos de término que el algoritmo de contracción de mallas (Au et al. [9]). Los resultados fueron poco concluyentes para retículos endoplasmáticos generados y reales: todos los algoritmos produjeron resultados similares.
- Número de ciclos inducidos: Debido a la conservación topológica, los algoritmos implementados en esta tesis producen resultados equivalentes. En el caso de retículos generados, el número de ciclos difiere del real debido a pequeños túneles (de 1 píxel de ancho) que se producen al generar los datos. Sin embargo, el algoritmo de contracción de mallas (Au et al. [9]) no es recomendable para esta métrica, debido a que el proceso de construcción y reparación de la malla puede originar o remover ciclos arbitrariamente.
- Histograma de largos: En general, ningún algoritmo logra sobresalir para esta métrica en ninguna de las pruebas efectuadas. Esto sugiere la necesidad de un algoritmo *ad-hoc* para las aplicaciones donde esta métrica sea relevante. En las aplicaciones que motivan esta tesis, tal algoritmo debería ser capaz de separar las superposiciones entre ramas para el caso de las neuronas (posiblemente basándose en el radio de los segmentos que se entrecruzan) y permitir la eliminación de aristas internas cortas en el caso de los retículos.
- Histograma de grados: El algoritmo de contracción de mallas (Au et al. [9]) tiende a producir más nodos falsos de grado 1 tanto en neuronas como en retículos endoplasmáticos, nuevamente debido a imprecisiones originadas del proceso de construcción de la malla. Sin embargo, este algoritmo de todas formas es el más recomendable para esta métrica en el caso de neuronas, puesto que, como se ha señalado anteriormente, produce menos nodos falsos. En el caso de retículos endoplasmáticos, el algoritmo basado en la distancia (Arcelli et al. [8]) produce un resultado más estable (prácticamente solo nodos de grado 3 y 4 en todas las simulaciones y en el caso real).
- Histograma de ángulos: Si bien en las neuronas generadas cada algoritmo logró una distribución relativamente similar a la ideal, en las figuras reales se producen muchas ramificaciones falsas, cuyos ángulos de origen debiesen ser omitidos del histograma. En este sentido, el algoritmo de

contracción de mallas (Au et al. [9]) resulta más apropiado, debido a que produce un *skeleton* con menos ramas espurias.

Trabajo futuro

Los algoritmos comparados en esta tesis son de uso general. Es decir, no contemplan las restricciones geométricas particulares del problema de calcular el *skeleton* de figuras biológicas pertenecientes a los dominios abordados en esta tesis. Es más, en la literatura no se encontró ningún algoritmo que calcule un *skeleton* (o tal vez otro descriptor más apropiado) para el problema específico de cuantificar estructuras biológicas. El trabajo futuro contempla en primer lugar proponer un algoritmo de estas características.

Por otro lado, los profesionales del área de Biología no disponen actualmente de herramientas gratuitas que permitan corregir manualmente un *skeleton* tridimensional. Para el trabajo futuro se propone resolver esta carencia mediante el desarrollo de una interfaz integrada a programas de procesamiento de imágenes conocidos en el área, como ImageJ.

Por último, los algoritmos fueron implementados en el lenguaje MATLAB, con el objetivo de facilitar las operaciones matriciales. Sin embargo, al tratarse de un software comercial, la accesibilidad de la comunidad científica a MATLAB es limitada. Parte del trabajo futuro contempla la transcripción de los algoritmos al lenguaje C y la publicación de su código documentado.

Bibliografía

- [1] M. ABRÀMOFF, P. MAGALHÃES, AND S. RAM, *Image processing with ImageJ*, Biophotonics international, 11 (2004), pp. 36–42.
- [2] M. AHMED AND R. WARD, *A rotation invariant rule-based thinning algorithm for character recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (2002), pp. 1672–1678.
- [3] N. AHUJA AND J. CHUANG, *Shape representation using a generalized potential field model*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 19 (1997), pp. 169–176.
- [4] L. ALCAYAGA, *Generación de skeletons a partir de mallas de superficie*, Memoria para optar al título de Ingeniero Civil en Computación, Universidad de Chile, (2012).
- [5] C. ARCELLI AND G. S. DI BAJA, *A width-independent fast thinning algorithm*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, (1985), pp. 463–474.
- [6] C. ARCELLI AND G. S. DI BAJA, *Finding local maxima in a pseudo-euclidian distance transform*, Computer Vision, Graphics, and Image Processing, 43 (1988), pp. 361–367.
- [7] C. ARCELLI, G. S. DI BAJA, AND L. SERINO, *From 3D discrete surface skeletons to curve skeletons*, in International Conference Image Analysis and Recognition, Springer, 2008, pp. 507–516.
- [8] C. ARCELLI, G. S. DI BAJA, AND L. SERINO, *Distance-driven skeletonization in voxel images*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 33 (2011), pp. 709–720.
- [9] O. K. AU, C. TAI, H. CHU, D. COHEN-OR, AND T. LEE, *Skeleton extraction by mesh contraction*, ACM Trans. Graph., 27 (2008), pp. 44:1–44:10.
- [10] X. BAI AND L. J. LATECKI, *Path similarity skeleton graph matching*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 30 (2008), pp. 1282–1292.

- [11] I. BARAN AND J. POPOVIĆ, *Automatic rigging and animation of 3D characters*, in ACM Transactions on Graphics (TOG), vol. 26, ACM, 2007, p. 72.
- [12] G. BERTRAND AND G. MALANDAIN, *A new characterization of three-dimensional simple points*, Pattern Recognition Letters, 15 (1994), pp. 169–175.
- [13] L. BILLECI, C. MAGLIARO, G. PIOGGIA, AND A. AHLUWALIA, *Neuronmorphological analysis tool: open-source software for quantitative morphometrics*, Frontiers in neuroinformatics, 7 (2013), p. 2.
- [14] H. BLUM, *A transformation for extracting descriptors of shape*, in Models for the Perception of Speech and Visual Forms, MIT Press, 1967, pp. 362–380.
- [15] G. BORGEFORS, *On digital distance transforms in three dimensions*, Computer vision and image understanding, 64 (1996), pp. 368–376.
- [16] G. BORGEFORS, I. NYSTRÖM, AND G. S. DI BAJA, *Computing skeletons in three dimensions*, Pattern Recognition, 32 (1999), pp. 1225–1236.
- [17] G. BORGEFORS, I. RAGNEMALM, AND G. S. DI BAJA, *The euclidean distance transform: finding the local maxima and reconstructing the shape*, in Procs. of the 7th Scand. Conf. on image analysis, vol. 2, 1991, pp. 974–981.
- [18] P. BOURKE, *VOL file format specifications: 2001*. <http://paulbourke.net/dataformats/vol>. accessed: 2017-08-29.
- [19] A. BUCKSCH, R. LINDENBERGH, AND M. MENENTI, *Skeltre*, The Visual Computer, 26 (2010), pp. 1283–1300.
- [20] V. CHATZIS AND I. PITAS, *Interpolation of 3D binary images based on morphological skeletonization*, IEEE transactions on medical imaging, 19 (2000), pp. 699–710.
- [21] S.-Y. CHEN, J. D. CARROLL, AND J. C. MESSENGER, *Quantitative analysis of reconstructed 3D coronary arterial tree and intracoronary devices*, IEEE transactions on medical imaging, 21 (2002), pp. 724–740.
- [22] N. D. CORNEA, D. SILVER, AND P. MIN, *Curve-skeleton properties, applications, and algorithms*, IEEE Transactions on Visualization & Computer Graphics, 47 (2007), pp. 530–548.
- [23] N. D. CORNEA, D. SILVER, X. YUAN, AND R. BALASUBRAMANIAN, *Computing hierarchical curve-skeletons of 3d objects*, The Visual Computer, 21 (2005), pp. 945–955.

- [24] H. CUNTZ, F. FORSTNER, A. BORST, AND M. HÄUSSER, *One rule to grow them all: a general theory of neuronal branching and its practical application*, PLoS computational biology, 6 (2010), p. e1000877.
- [25] T. K. DEY AND W. ZHAO, *Approximating the medial axis from the voronoi diagram with a convergence guarantee*, Algorithmica, 38 (2004), pp. 179–200.
- [26] G. S. DI BAJA, *Well-shaped, stable, and reversible skeletons from the (3, 4)-distance transform*, Journal of visual communication and image representation, 5 (1994), pp. 107–115.
- [27] G. S. DI BAJA AND E. THIEL, *Skeletonization algorithm running on path-based distance maps*, Image and vision Computing, 14 (1996), pp. 47–57.
- [28] R. O. DUDA AND J. H. MUNSON, *Graphical-data-processing research study and experimental investigation*, tech. rep., DTIC Document, 1967.
- [29] F. F. FOL-LEYMARIE, *Three-dimensional Shape Representation via Shock Flows*, PhD thesis, Brown University, 2003.
- [30] H. FRIMMEL, J. NÄPPI, AND H. YOSHIDA, *Centerline-based colon segmentation for ct colonography*, Medical Physics, 32 (2005), pp. 2665–2672.
- [31] N. GAGVANI AND D. SILVER, *Animating volumetric models*, Graphical Models, 63 (2001), pp. 443–458.
- [32] T. GILLETTE, K. BROWN, AND G. A. ASCOLI, *The diadem metric: comparing multiple reconstructions of the same neuron*, Neuroinformatics, 9 (2011), pp. 233–245.
- [33] Z. S. GO, J. SHAH, AND H. P. ET AL., *Extraction of shape skeletons from grayscale images*, Computer Vision and Image Understanding, 66 (1997), pp. 133–146.
- [34] W. GONG AND G. BERTRAND, *A simple parallel 3d thinning algorithm*, in Pattern Recognition, 1990. Proceedings., 10th International Conference on, vol. 1, IEEE, 1990, pp. 188–190.
- [35] H. GREENSPAN, M. LAIFENFELD, S. EINAV, AND O. BARNEA, *Evaluation of center-line extraction algorithms in quantitative coronary angiography*, Medical Imaging, IEEE Transactions on, 20 (2001), pp. 928–941.
- [36] R. GUY, *Github.com: Matlab priority queue*. <https://github.com/guyrt/matlabpriorityqueue>. accessed: 2017-08-01, 2011.

- [37] M. S. HASSOUNA AND A. F. ET AL., *Variational curve skeletons using gradient vector flow*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 31 (2009), pp. 2257–2274.
- [38] T. HE, L. HONG, D. CHEN, AND Z. LIANG, *Reliable path for virtual endoscopy: Ensuring complete examination of human organs*, IEEE transactions on visualization and computer graphics, 7 (2001), pp. 333–342.
- [39] C. J. HILITCH, *Linear skeletons from square cupboards*, in Machine Intelligence 4, Edinburgh University Press, 1969, p. 403.
- [40] S. Y. HO, C. Y. CHAO, H. L. HUANG, T. W. CHIU, P. CHAROENKWAN, AND H. E, *Neurphologyj: an automatic neuronal morphology quantification method and its application in pharmacological discovery*, BMC bioinformatics, 12 (2011), p. 230.
- [41] J. HOPCROFT AND R. TARJAN, *Algorithm 447: efficient algorithms for graph manipulation*, Communications of the ACM, 16 (1973), pp. 372–378.
- [42] H. HUANG, S. WU, D. COHEN-OR, M. GONG, H. ZHANG, G. LI, AND B. CHEN, *L1-medial skeleton of point cloud.*, ACM Trans. Graph., 32 (2013), p. 65.
- [43] L. HUANG AND A. BIJAOUI, *Astronomical image data compression by morphological skeleton transformation*, Experimental Astronomy, 1 (1990), pp. 311–327.
- [44] R. JAIN, R. KASTURI, AND B. SCHUNCK, *Machine vision*, vol. 5, McGraw-Hill New York, 1995.
- [45] A. JALBA, J. KUSTRA, AND A. TELEA, *Surface and curve skeletonization of large 3d models on the gpu*, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35 (2013), pp. 1495–1508.
- [46] F. JANOOS, K. MOSALIGANTI, X. XU, R. MACHIRAJU, K. HUANG, AND S. WONG, *Robust 3d reconstruction and identification of dendritic spines from optical microscopy imaging*, Medical image analysis, 13 (2009), pp. 167–179.
- [47] X. JIANG AND W.-Y. YAU, *Fingerprint minutiae matching based on the local and global structures*, in Pattern recognition, 2000. Proceedings. 15th international conference on, vol. 2, IEEE, 2000, pp. 1038–1041.
- [48] R. KIMMEL, D. SHAKED, N. KIRYATI, AND A. BRUCKSTEIN, *Skeletonization via distance maps and level sets*, Computer vision and image understanding, 62 (1995), pp. 382–391.

- [49] T. Y. KONG, *A digital fundamental group*, Computers & Graphics, 13 (1989), pp. 159–166.
- [50] T. Y. KONG AND A. ROSENFELD, *Digital topology: Introduction and survey*, Computer Vision, Graphics, and Image Processing, 48 (1989), pp. 357–393.
- [51] L. LAM, S. LEE, AND C. Y. SUEN, *Thinning methodologies-a comprehensive survey*, IEEE Transactions on pattern analysis and machine intelligence, 14 (1992), pp. 869–885.
- [52] J.-M. LIEN, G. KURILLO, AND R. BAJCSY, *Skeleton-based data compression for multi-camera tele-immersion system*, Advances in Visual Computing, (2007), pp. 714–723.
- [53] A. LIEUTIER, *Any open bounded subset of \mathbb{R}^n has the same homotopy type as its medial axis*, Computer-Aided Design, 36 (2004), pp. 1029–1046.
- [54] H. LIU, Z. WU, D. F. HSU, B. S. PETERSON, AND D. XU, *On the generation and pruning of skeletons using generalized voronoi diagrams*, Pattern Recognition Letters, 33 (2012), pp. 2113–2119.
- [55] S. LOBREGT, P. W. VERBEEK, AND F. C. A. GROEN, *Three-dimensional skeletonization: principle and algorithm*, IEEE Transactions on Pattern Analysis & Machine Intelligence, (1980), pp. 75–77.
- [56] M. H. LONGAIR, D. A. BAKER, AND J. D. ARMSTRONG, *Simple neurite tracer: open source software for reconstruction, visualization and analysis of neuronal processes*, Bioinformatics, 27 (2011), pp. 2453–2454.
- [57] C. R. MAURER, R. QI, AND V. RAGHAVAN, *A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 25 (2003), pp. 265–270.
- [58] P. MIN, *Binvox code-3d mesh voxelizer*, Princeton University: Princeton University, (2011).
- [59] D. G. MORGENTHALER, *Three-dimensional Simple Points: Serial Erosion, Parallel Thinning, and Skeletonization*, Technical report: Computer Science Center, University of Maryland, 1981.
- [60] D. R. MYATT, T. HADLINGTON, G. A. ASCOLI, AND S. J. NASUTO, *Neuromantic—from semi-manual to semi-automatic reconstruction of neuron morphology*, Frontiers in neuroinformatics, 6 (2012).

- [61] R. OGNIWICZ AND M. ILG, *Voronoi skeletons: Theory and applications*, in Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on, IEEE, 1992, pp. 63–69.
- [62] R. OGNIWICZ AND O. KÜBLER, *Hierarchic voronoi skeletons*, Pattern recognition, 28 (1995), pp. 343–359.
- [63] K. PALÁGYI AND A. KUBA, *A parallel 3d 12-subiteration thinning algorithm*, Graphical Models and Image Processing, 61 (1999), pp. 199–221.
- [64] G. PANI, W. H. VOS, N. SAMARI, L. SAINT-GEORGES, S. BAATOUT, P. OOSTVELDT, AND M. A. BENOTMANE, *Morphoneuronet: an automated method for dense neurite network analysis*, Cytometry Part A, 85 (2014), pp. 188–199.
- [65] D. PERCHET, C. FETITA, AND F. J. PRETEUX, *Advanced navigation tools for virtual bronchoscopy*, in Electronic Imaging 2004, International Society for Optics and Photonics, 2004, pp. 147–158.
- [66] L. POTHUAUD, A. LAIB, P. LEVITZ, C. L. BENHAMOU, AND S. MAJUMDAR, *Three-dimensional-line skeleton graph analysis of high-resolution magnetic resonance images: A validation study from 34- μ m-resolution microcomputed tomography*, Journal of Bone and Mineral Research, 17 (2002), pp. 1883–1895.
- [67] C. PUDNEY, *Distance-ordered homotopic thinning: a skeletonization algorithm for 3d digital images*, Computer Vision and Image Understanding, 72 (1998), pp. 404–413.
- [68] M. PUHKA, M. JOENSUU, H. VIHINEN, I. BELEVICH, AND E. JOKITALO, *Progressive sheet-to-tubule transformation is a general mechanism for endoplasmic reticulum partitioning in dividing mammalian cells*, Molecular biology of the cell, 23 (2012), pp. 2424–2432.
- [69] M. PUHKA, H. VIHINEN, M. JOENSUU, AND E. JOKITALO, *Endoplasmic reticulum remains continuous and undergoes sheet-to-tubule transformation during cell division in mammalian cells*, The Journal of cell biology, 179 (2007), pp. 895–909.
- [70] G. REIG, M. CERDA, N. SEPÚLVEDA, D. FLORES, V. CASTAÑEDA, M. TADA, S. HÄRTEL, AND M. L. CONCHA, *Extra-embryonic tissue spreading directs early embryo morphogenesis in killifish*, Nature Communications, 8 (2017).

- [71] E. REMY AND E. THIEL, *Look-up tables for medial axis on squared euclidean distance transform*, in International Conference on Discrete Geometry for Computer Imagery, Springer, 2003, pp. 224–235.
- [72] D. RENIERS, J. VAN WIJK, AND A. TELEA, *Computing multiscale curve and surface skeletons of genus 0 shapes using a global importance measure*, IEEE Transactions on Visualization and Computer Graphics, 14 (2008).
- [73] I. Y. ROJAS, *Optimización y paralelización de un algoritmo de generación de skeletons a partir de mallas geométricas aplicado a estructuras biológicas*, Memoria para optar al título de Ingeniero Civil en Computación, Universidad de Chile, (2014).
- [74] A. ROSENFELD AND J. L. PFALTZ, *Sequential operations in digital picture processing*, Journal of the ACM (JACM), 13 (1966), pp. 471–494.
- [75] L. SABORET, M. ATTENE, AND P. ALLIEZ, *Laurent’s hand, the aim@ shape shape repository*, 2007.
- [76] P. K. SAHA, G. BORGEFORS, AND G. SANNITI DI BAJA, *A survey on skeletonization algorithms and their applications*, Pattern Recognition Letters, 76 (2016), pp. 3–12.
- [77] P. K. SAHA, B. B. CHAUDHURI, B. BIDYUT, AND D. D. MAJUMDER, *A new shape preserving parallel thinning algorithm for 3d digital images*, Pattern Recognition, 30 (1997), pp. 1939–1955.
- [78] P. K. SAHA, B. B. CHAUDHURI, B. CHANDA, AND D. D. MAJUMDER, *Topology preservation in 3d digital space*, Pattern Recognition, 27 (1994), pp. 295–300.
- [79] M. SCHAAP, C. METZ, T. VAN WALSUM, A. G. VAN DER GIESSEN, A. C. WEUSTINK, N. R. MOLLET, C. BAUER, H. BOGUNOVIĆ, C. CASTRO, AND X. D. ET AL., *Standardized evaluation methodology and reference database for evaluating coronary artery centerline extraction algorithms*, Medical image analysis, 13 (2009), pp. 701–714.
- [80] S. K. SCHMITZ, J. J. HJORTH, R. M. JOEMAI, R. WIJNTJES, S. EIJGENRAAM, P. DE BRUIJN, C. GEORGIU, A. P. DE JONG, A. VAN OOYEN, M. VERHAGE, ET AL., *Automated analysis of neuronal morphology, synapse number and synaptic recruitment*, Journal of neuroscience methods, 195 (2011), pp. 185–193.
- [81] D. SELLE, B. PREIM, A. SCHENK, AND H.-O. PEITGEN, *Analysis of vasculature for liver surgical planning*, IEEE transactions on medical imaging, 21 (2002), pp. 1344–1357.

- [82] L. SERINO, C. ARCELLI, AND G. SANNITI DI BAJA, *Decomposing 3d objects in simple parts characterized by rectilinear spines*, International Journal of Pattern Recognition and Artificial Intelligence, 28 (2014), p. 1460010.
- [83] E. C. SHERBROOKE, N. PATRIKALAKIS, AND E. BRISSON, *An algorithm for the medial axis transform of 3d polyhedral solids*, Visualization and Computer Graphics, IEEE Transactions on, 2 (1996), pp. 44–61.
- [84] P. SHILANE, P. MIN, M. KAZHDAN, AND T. FUNKHOUSER, *The princeton shape benchmark*, in Shape modeling applications, 2004. Proceedings, IEEE, 2004, pp. 167–178.
- [85] K. SHORT, M. HODSON, AND I. SMYTH, *Spatial mapping and quantification of developmental branching morphogenesis*, Development, 140 (2013), pp. 471–478.
- [86] K. SIDDIQI, S. BOUIX, A. TANNENBAUM, AND S. ZUCKER, *Hamilton-jacobi skeletons*, International Journal of Computer Vision, 48 (2002), pp. 215–231.
- [87] K. SIDDIQI AND S. PIZER, *Medial representations: mathematics, algorithms and applications*, Springer Science & Business Media, Volume 37, 2008.
- [88] A. SOBIECKI, A. JALBA, AND A. TELEA, *Comparison of curve and surface skeletonization methods for voxel shapes*, Pattern Recognition Letters, 47 (2014), pp. 147–156.
- [89] A. SOBIECKI, H. YASAN, C. HALUK, A. JALBA, C. ANDREI, AND A. TELEA, *Qualitative comparison of contraction-based curve skeletonization methods*, in Mathematical Morphology and Its Applications to Signal and Image Processing, Springer, 2013, pp. 425–439.
- [90] E. SORANTIN, C. HALMAI, B. ERDOHELYI, K. PALÁGYI, L. G. NYÚL, K. OLLÉ, B. GEIGER, F. LINDBICHLER, G. FRIEDRICH, AND K. KIESLER, *Spiral-ct-based assessment of tracheal stenoses using 3D-skeletonization*, IEEE transactions on Medical Imaging, 21 (2002), pp. 263–273.
- [91] R. STEFANELLI AND A. ROSENFELD, *Some parallel thinning algorithms for digital pictures*, Journal of the ACM (JACM), 18 (1971), pp. 255–264.
- [92] H. SUNDAR, D. SILVER, N. GAGVANI, AND S. DICKINSON, *Skeleton based shape matching and retrieval*, in Shape Modeling International, 2003, IEEE, 2003, pp. 130–139.
- [93] S. SVENSSON, C. ARCELLI, AND G. S. DI BAJA, *Finding cavities and tunnels in 3d complex objects*, in Image Analysis and Processing, 2003. Proceedings. 12th International Conference on, IEEE, 2003, pp. 342–347.

- [94] S. SVENSSON AND G. S. DI BAJA, *Using distance transforms to decompose 3d discrete objects*, Image and Vision Computing, 20 (2002), pp. 529–540.
- [95] A. TAGLIASACCHI, I. ALHASHIM, M. OLSON, AND H. ZHANG, *Mean curvature skeletons*, in Computer Graphics Forum, vol. 31, Wiley Online Library, 2012, pp. 1735–1744.
- [96] A. TAGLIASACCHI, T. DELAME, M. SPAGNUOLO, N. AMENTA, AND A. TELEA, *3d skeletons: A state-of-the-art report*, in Computer Graphics Forum, vol. 35, Wiley Online Library, 2016, pp. 573–597.
- [97] A. TELEA AND A. VILANOVA, *A robust level-set algorithm for centerline extraction*, in Proceedings of the symposium on Data visualisation 2003, Eurographics Association, 2003, pp. 185–194.
- [98] L. THURFJELL, E. BENGTSSON, AND B. NORDIN, *A new three-dimensional connected components labeling algorithm with simultaneous object feature extraction capability*, CVGIP: Graphical Models and Image Processing, 54 (1992), pp. 357–364.
- [99] E. TRACY, *Magicavoxel: a lightweight 8-bit voxel editor and interactive path tracing renderer, enjoy :)*: 2017. <https://ephtracy.github.io/>. accessed: 2017-08-29.
- [100] Y. F. TSAO AND K. S. FU, *A parallel thinning algorithm for 3D pictures*, Computer graphics and image processing, 17 (1981), pp. 315–331.
- [101] S. VILLARROEL, M. CERDA, F. SANTIBAÑEZ, O. RAMÍREZ, K. PALMA, M. CONCHA, N. HITSCHFELD-KAHLER, AND S. HÄRTEL, *Defining metrics to compare skeleton estimation methods applied to biological structures*, in Defining Metrics to Compare Skeleton Estimation Methods applied to Biological Structures, 2012.
- [102] L. VINCENT AND P. SOILLE, *Watersheds in digital spaces: an efficient algorithm based on immersion simulations*, IEEE transactions on pattern analysis and machine intelligence, 13 (1991), pp. 583–598.
- [103] L. WADE AND R. E. PARENT, *Automated generation of control skeletons for use in animation*, The Visual Computer, 18 (2002), pp. 97–110.
- [104] M. WAN, Z. LIANG, Q. KE, L. HONG, I. BITTER, AND A. KAUFMAN, *Automatic centerline extraction for virtual colonoscopy*, Medical Imaging, IEEE Transactions on, 21 (2002), pp. 1450–1460.

- [105] M. WEST, N. ZUREK, A. HOENGER, AND G. K. VOELTZ, *A 3d analysis of yeast er structure reveals how er domains are organized by membrane curvature*, The Journal of cell biology, 193 (2011), pp. 333–346.
- [106] L. WESTRATE, J. LEE, W. PRINZ, AND G. VOELTZ, *Form follows function: the importance of endoplasmic reticulum shape*, Annual review of biochemistry, 84 (2015), pp. 791–811.
- [107] R. ZWIGGELAAR, S. M. ASTLEY, C. R. BOGGIS, AND C. J. TAYLOR, *Linear structures in mammographic images: detection and classification*, IEEE transactions on medical imaging, 23 (2004), pp. 1077–1086.

Apéndices

A . Comparación en modelos generados a partir de segmentos curvilíneos

En esta sección se describe una comparación del desempeño de los algoritmos implementados al calcular el *skeleton* de un volumen curvo. El objetivo de este análisis adicional es complementar la comparación del Capítulo 5, donde los modelos generados estaban constituidos exclusivamente por cilindros rectos. Se explora el efecto de cada algoritmo en cuanto a la conservación del largo de formas sencillas.

Los modelos escogidos se construyen a partir de una línea de 100 vóxeles de largo. Esta línea es deformada con radios de curvatura progresivamente mayores, hasta formar una semicircunferencia. Con esto, se obtienen 5 curvas de vóxeles, que posteriormente son dilatadas utilizando un *kernel* esférico de diámetro 5. Así se obtienen 5 modelos, representando las curvas originales el *skeleton* ideal para cada caso.

La Tabla 6.1 muestra los modelos, el largo del *skeleton* ideal y los largos calculados por cada algoritmo.

La creciente diferencia entre los largos de los *skeletons* y el largo ideal se explica por la eliminación de vóxeles 6- y 18- adyacentes. A medida que la curvatura aumenta, la curva de vóxeles original contiene más vóxeles 6-vecinos entre sí. Muchos de estos vóxeles son eliminados por los algoritmos al favorecer las conexiones por vértice para obtener la figura más delgada posible. Por lo tanto, cuanto mayor sea la curvatura de la figura, peor será la aproximación del largo utilizando el *skeleton* calculado por los algoritmos implementados para esta tesis.

Los resultados muestran que los algoritmos de Siddiqi et al. [86] y Arcelli et al. [8] producen un *skeleton* de largo más cercano al original. Esto se condice con los resultados excesivamente simplificados del algoritmo de Palágyi y Kuba [63] mostrados a lo largo de esta tesis.






| Modelo | Largo ideal | Palágyi y Kuba | Siddiqi et al. | Arcelli et al. |
|------------------------------------------------------------------------------------|-------------|----------------|----------------|----------------|
|  | 100,00 | 100,00 | 100,00 | 100,00 |
|  | 120,36 | 93,97 | 99,45 | 102,25 |
|  | 144,54 | 84,28 | 101,08 | 103,15 |
|  | 175,48 | 88,74 | 111,71 | 114,95 |
|  | 228,45 | 146,99 | 157,85 | 149,75 |

Tabla 6.1: Resultados para los largos de los *skeletons* calculados a partir de modelos generados a partir de segmentos curvilíneos con distinta curvatura.

B . Obtención de volúmenes estándar

Los volúmenes de la Figura 6.1 fueron obtenidos voxelizando mallas de superficie pertenecientes a la base de datos Princeton Shape Benchmark [84], cuyo formato original es OFF. La voxelización se hizo con el programa `binvox` mediante el comando:

```
binvox -d 128 128 128 <ruta malla original><ruta archivo de salida>
```

Ejecutar este comando produce un archivo de vóxeles en un formato especial para `binvox`. El parámetro `-d 128 128 128` indica las dimensiones del archivo de vóxeles, es decir, la resolución del muestreo de la malla. Estos valores se escogieron por ser los utilizados para mostrar los resultados en el artículo de Arcelli et al. [8], correspondiente al tercer algoritmo implementado.

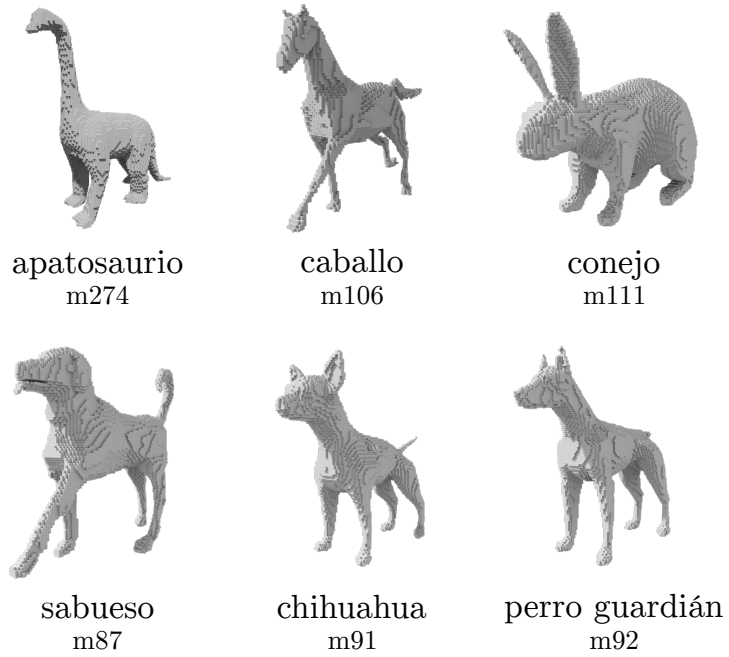


Figura 6.1: Volúmenes de prueba de la Princeton Shape Benchmark Database.

Adicionalmente, se utilizaron los volúmenes de la Figura 6.2. Estos volúmenes provienen de la base de datos Aim@Shape [75], que actualmente no se encuentra disponible en línea. Los archivos fueron proporcionados por Gabriella Sanniti di Baja, coautora del algoritmo del Capítulo 4, en formato VOL. En los apéndices se incluye la especificación de este formato y su lectura en MATLAB.

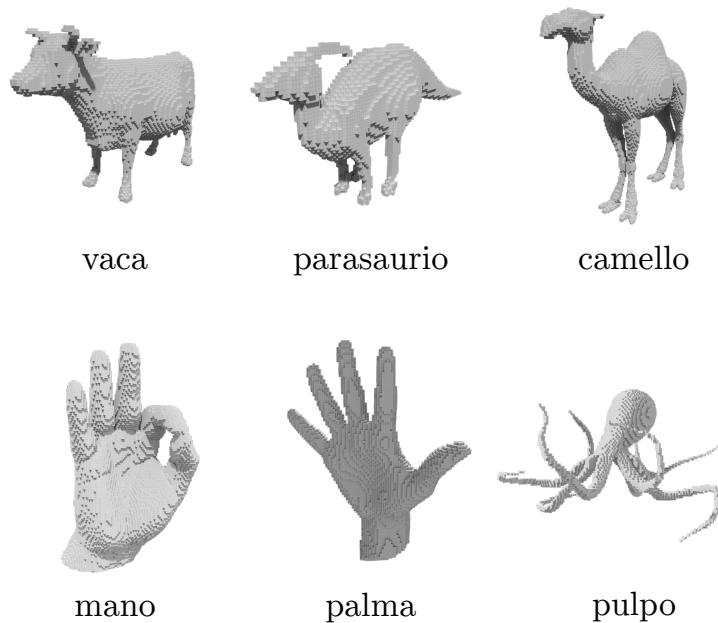


Figura 6.2: Volúmenes de prueba de la base de datos Aim@Shape.

C . Formatos de archivos de vóxeles y programas para su visualización

C .1. MagicaVoxel y el formato .xraw

El programa de edición y animación de vóxeles MagicaVoxel [99] se utilizó para todas las visualizaciones de volúmenes de esta tesis. El formato .xraw es propio de este programa, y su especificación puede encontrarse en GitHub [99]. Para la lectura en una matriz de MATLAB para este formato de archivos, y viceversa, se implementaron las funciones `xraw2mat.m` y `mat2xraw.m`. La Figura 6.3 muestra la interfaz de este programa.

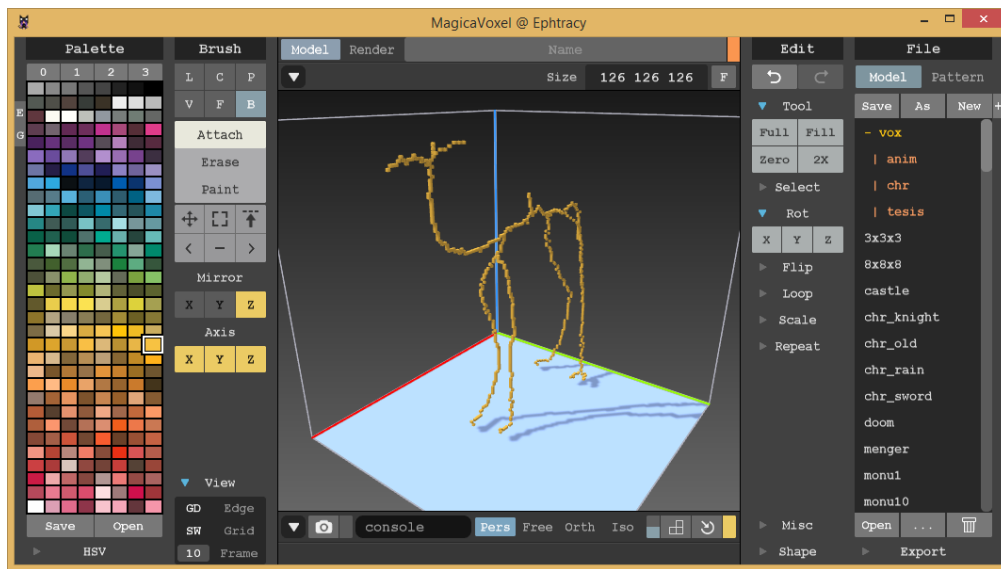


Figura 6.3: Interfaz del programa MagicaVoxel.

Existe una limitante para el tamaño del volumen que MagicaVoxel permite visualizar. Como máximo, el volumen puede ser de $128 \times 128 \times 128$ vóxeles, es decir, un tamaño insuficiente para los modelos que motivan esta tesis. Una versión alternativa llamada MagicaVoxel Viewer [99] permite visualizar modelos más grandes, pero no permite editarlos. Esta versión se utilizó para visualizar las neuronas y retículos endoplasmáticos de esta tesis.

C .2. Binvox y el formato .binvox

El paquete binvox fue utilizado para voxelizar los volúmenes de la Figura 6.1 a partir de mallas geométricas, y en general para visualizar volúmenes de vóxeles rápidamente. El formato .binvox fue diseñado de manera *ad-hoc* por el autor de binvox, Patrick Min, como el formato de archivos para binvox [58]. Para la lectura en una matriz de MATLAB para este formato de archivos, y viceversa, se implementaron las funciones `binvox2mat.m` y `mat2binvox.m`.

El paquete binvox incluye el visualizador viewbox, que se muestra en la Figura 6.4. Este visualizador es rápido y simple de utilizar, pero tiene problemas para mostrar modelos demasiado grandes.

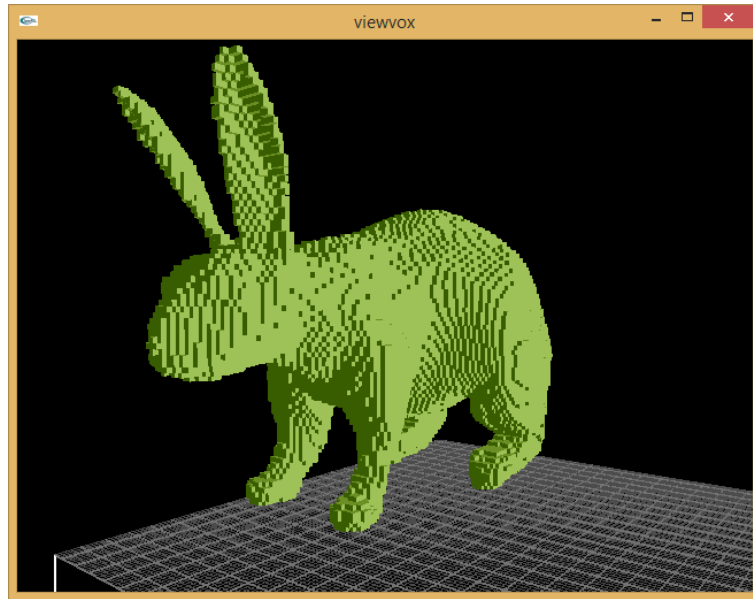


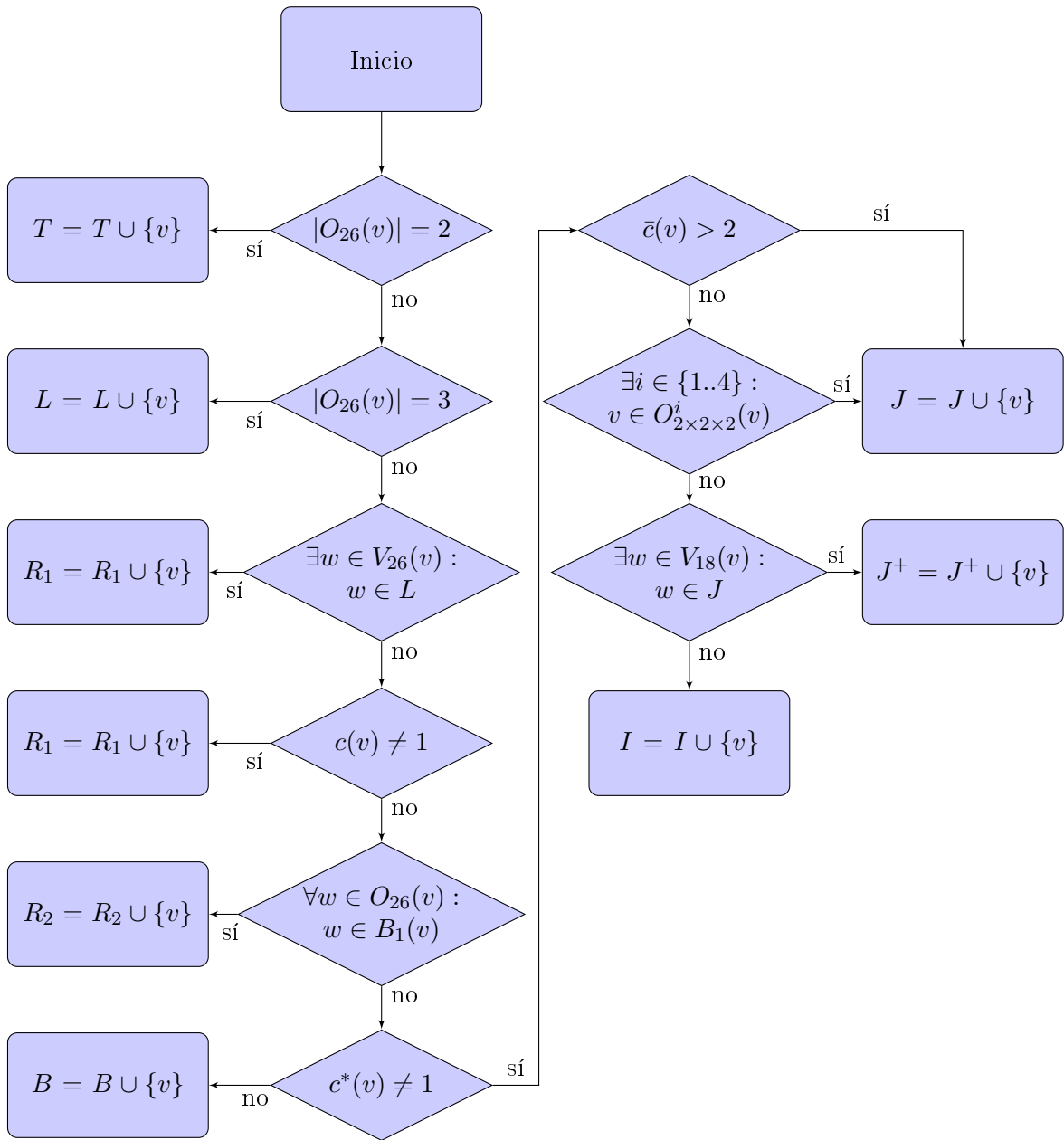
Figura 6.4: Interfaz del programa viewvox, del paquete binvox.

C .3. El formato VOL

No se encontró el origen de este formato ni un programa que pudiera leerlo, pero sí su especificación [18]. Los modelos provistos por Gabriella Sanniti di Baja, ilustrados en la Figura 6.2, estaban en este formato. Por lo tanto, para su lectura en MATLAB se programó la función `vol2mat.m`.

D . Clasificación de vóxeles para el algoritmo de Arcelli et al.

El siguiente diagrama especifica secuencialmente la clasificación de vóxeles utilizada en el algoritmo de Arcelli et al.



En este diagrama:

- $O_{26}(v)$ es el conjunto de vóxeles de objeto en $V_{26}(v)$.
- $c(v)$ y $\bar{c}(v)$ son los números utilizados para la condición de simplicidad de v .
- $O^i(v)$ son las regiones de $2 \times 2 \times 2$ vóxeles de objeto que contienen a v .

Para el algoritmo del Capítulo 4, los conjunto de vóxeles que más interesan son conjunto de vóxeles de borde B y el conjunto de vóxeles internos I .