Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Representación y análisis de sólidos voxelizados mediante árboles de bordes

T E S I S

que para optar por el grado de

Doctor en Ciencias
(Computación)

presenta

Luis Artemio Martínez Vázquez

Tutor Principal

Dr. Ernesto Bribiesca Correa
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, UNAM

Comité Tutor

Dr. Boris Escalante Ramírez
Facultad de Ingeniería, UNAM

Dr. Edgar Garduño Ángeles
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, UNAM

México, D.F. Septiembre 2015

Ut In Omnibus Glorificetur Deus

para mi querida esposa Bertha,
compañera de viaje...

Es menester mostrar mi profunda gratitud

Al Dr. Ernesto Bribiesca
Al Dr. Adolfo Guzmán
Al Dr. Boris Escalante
Al Dr. Edgar Garduño
Al Dr. Hermilo Sánchez
Al Instituto de Astronomía, UNAM
A la DGAPA, UNAM
A la Universidad Nacional Autónoma de México
Al pueblo que con su trabajo cotidiano sostiene esta Universidad

Resumen

Debido a la importancia creciente del análisis de forma y el reconocimiento de patrones tridimensionales en áreas como medicina, visión por computadora, robótica, entre otras; ha surgido la necesidad de contar con descriptores sencillos que permitan la representación, manejo y el análisis de objetos tridimensionales. En el ámbito de los sólidos voxelizados, las técnicas de esqueletización han dominado la búsqueda de descriptores desde hace algún tiempo. No obstante el desarrollo que han alcanzado, aún persisten algunas deficiencias que, o son intrínsecas a los esqueletos o bien no han sido resueltas del todo.

Los códigos de cadenas permiten representar objetos mediante el uso de su borde o frontera y a partir de ésta, generar descriptores que capturan las características de los mismos. En la mayoría de los casos, son secuencias de caracteres obtenidos mediante la aplicación de reglas que codifican los bordes del objeto. En general, se busca que los códigos de cadenas tengan ciertas características que ayuden al análisis de los objetos tales como ser invariantes a transformaciones de escala, rotaciones, entre otras.

Teniendo como base un código de cadenas que fue utilizado para generar descriptores de curvas y estructuras arborescentes, denominado 5OT, cuyo alfabeto $\{0, 1, 2, 3, 4\}$ describe cambios de dirección ortogonales en el espacio tridimensional; en este trabajo se introducen dos familias de árboles tridimensionales que permiten representar sólidos voxelizados mediante descriptores unidimensionales 5OT. La primera familia está constituida por los árboles envolventes que describen la superficie de sólidos voxelizados. La segunda familia está formada por árboles de bordes que son una representación más compacta que los árboles envolventes y buscan ser una alternativa a los esqueletos de sólidos voxelizados. Entre las características más sobresalientes de los árboles envolventes y de bordes se encuentran la invariancia a rotaciones, traslaciones y transformaciones de escala.

Adicionalmente, en este trabajo se explora la factibilidad de usar algunas métricas para comparar los árboles envolventes y de bordes como descriptores de sólidos voxelizados. Particularmente se consideró de interés sondear algunos algoritmos de análisis léxico de cadenas como la distancia de Levenshtein que proporciona un valor a las diferencias globales entre cadenas. Aprovechando que el árbol como gráfica es intrísencamente descrito por la cadena, también se estudió la capacidad de detectar similitudes entre sólidos mediante la teoría espectral de gráficas así como la distancia de edición entre árboles. Asímismo, se introducen los modelos de adyacencia celular y de puntos para proporcionar un marco de referencia que ayude a establecer las diferencias básicas entre los distintos tipos de códigos de cadenas existentes.

Contenido

Capítulo 1

Planteamiento y objetivos de la tesis

1.1. Ámbito de la tesis

En las últimas décadas ha habido un incremento notorio en el uso de imágenes tridimensionales (3D) en muchas disciplinas científicas y técnicas que van desde los sistemas de diseño asistido por computadora (CAD), imaginología médica, visualización científica, visión por computadora, realidad virtual, entre otras. Es de tal magnitud el volumen creciente de información 3D que ha llevado al surgimiento de nuevos métodos para representar las formas de los objetos, así como algoritmos y métodos para analizarlos. Incluso comienzan a vislumbrarse nuevos modos de usar la información 3D acumulada durante estos años especialmente en áreas de diseño industrial que han generado nuevas tendencias en el diseño de manufacturas como la Tecnología de Agrupamiento (*Group Technology* en inglés) en el cual los componentes que tienen características semejantes se agrupan para facilitar su clasificación, determinar los requerimientos específicos de diseño y fabricación [JKR09, RCCT06].

A lo largo de este trabajo, se entenderá como imagen tridimensional la representación de un objeto sólido tridimensional mediante una división en elementos cúbicos unitarios denominados *voxeles* en el espacio $\mathbb{Z}^3$. Dado un sólido; a cada voxel se le puede asignar un valor real que se denomina densidad [TY09]. En nuestro caso, únicamente se utilizarán objetos con densidad uno. Por consiguiente, un voxel que no forme parte del sólido tendrá un valor de cero. Es importante señalar que cada objeto se considerará aislado del mundo real y que fue el resultado de un proceso de digitalización.

Puesto que la información 3D es más compleja de analizar y comparar que la información 2D, existe un interés creciente en desarrollar procedimientos para la creación de descriptores sencillos de calcular, análogos a los propuestos a lo largo del tiempo para 2D, que faciliten el análisis y la comparación de objetos tridimensionales. En general se busca que estos descriptores tengan una serie de propiedades deseables similares al caso 2D; como por ejemplo, ser invariantes ante transformaciones rígidas, compactos en su representación, menor complejidad de cómputo y tiempo, y capacidad para capturar las características principales de la forma del sólido. Un ejemplo de aplicación que busca obtener ventaja de lo anterior son los catálogos de compañías que ofrecen piezas mecánicas. Si un cliente requiere una pieza con determinadas características basta esbozar sus rasgos más importantes y hacer una búsqueda en bases de datos 3D para encontrar la pieza solicitada o bien las más parecidas disponibles en un catálogo. El proceso de búsqueda inicia transformando el boceto original para abstraer las características de la pieza en un modelo 3D de voxeles. Mediante técnicas de morfología matemática tanto las piezas de los catálogos como la del cliente son adelgazadas a un esqueleto que captura las características más importantes del objeto original y es mediante la comparación de esqueletos que se lleva a cabo la búsqueda en los catálogos. La expresión del boceto original en términos de construcciones más simples reduce la cantidad de información y por consiguiente facilita su manejo para tareas de búsqueda y procesamiento [FMK03, Go04, IJL05, JKR09].

En la búsqueda de descriptores para representar objetos 3D se pueden distinguir dos tendencias principales de investigación y desarrollo de algoritmos: a) el uso de códigos de cadenas y b) la esqueletización [KR04, CSM07, ASS11]. Ambas tendencias ya han sido exploradas exhaustivamente en el caso de imágenes 2D.

Los códigos de cadenas por lo general tratan de representar una imagen con base en su borde o frontera y a partir de ésta generar un descriptor que captura las características de la imagen [GW07], un ejemplo clásico de código bidimensional es el desarrollado por H. Freeman [HF61].

La esqueletización permite mediante ciertos procedimientos, obtener una representación de menor dimensión para un objeto [KR04]. Los algoritmos para obtener esta representación también son conocidos como de adelgazamiento (*thinning* en inglés) o erosión y al resultado obtenido generalmente se le denomina esqueleto. Los esqueletos también pueden ser utilizados en aplicaciones tales como búsqueda de patrones y manejo de bases de datos de objetos 3D. Generalmente la esqueletización es un proceso iterativo que

requiere de la identificación, iteración tras iteración de ciertos voxeles en el objeto de manera que puedan ser aplicadas las operaciones morfológicas para la eliminación de los mismos. La representación medial de un objeto es un modelo muy conocido que ha sido utilizado ampliamente para obtener esqueletos de objetos 2D y 3D [DH73].

Aunque los esqueletos preservan la topología de los objectos originales y pueden ofrecer información cualitativa, éstos no necesariamente pueden ser regenerados a partir de sus esqueletos lo que causa una pérdida en la información [SNS02]. En particular este es el caso de los esqueletos formados de curvas que generalmente son representaciones que pierden toda la información geométrica del objeto [CSM07, ASS11]. Por consiguiente, obtener una representación para describir la geometría de un objeto y que adicionalmente permita recuperarlo en su forma original es una meta de investigación deseable.

En el universo de áreas relacionadas con el estudio de imágenes 3D y de objetos voxelizados en particular, la representación mediante el uso de código de cadenas ha sido poco explorado, contrariamente a lo que ha ocurrido con el caso bidimensional que ha sido ampliamente investigado. En 2008, E. Bribiesca [EB08] propuso un método para representar objetos 3D con formas arborescentes usando un código de cadenas que había desarrollado previamente [EB00], denominado 5OT en el artículo [SCB08] (notación que se usará de aquí en adelante). Este código permite la comparación de árboles 3D a través de la inspección únicamente del código que representa cada árbol. Entre otras caractertísticas, el código 5OT es invariante bajo rotaciones, traslaciones y transformaciones especulares además de preservar sus características topológicas. Cabe señalar que el código 5OT también ha sido utilizado para definir una métrica de disimilitud para comparar curvas 3D [BA06] y es un código que ofrece mayor compresión que el código de Freeman [SCB08]. En un artículo publicado en 2012 que incluye algunos de los resultados obtenidos en el presente proyecto de investigación [BGM12], se extendió el uso del código 5OT en sólidos voxelizados y se propuso una representación mediante *árboles envolventes*.

Recientemente se publicaron dos nuevas propuestas para representar sólidos voxelizados. La primera propuesta publicada por Sánchez-Cruz *et al.* [SC14] está basada en la extensión del código de Freeman al espacio tridimensional y la segunda publicada por Lemus *et al.* [LBG14] en la representación de las caras de los sólidos.

1.2. Objetivos de la tesis y contribuciones

Con las ideas de esqueletización de objetos y la representación de árboles en mente, la presente tesis se centra en la representación de objetos 3D mediante el uso del código 5OT como una opción distinta a los algoritmos de esqueletización. Se introducen los *árboles envolventes* y los *árboles de bordes* como alternativas de representación a través del uso del código 5OT. Los árboles envolventes (AEs) tienen como característica principal la de recorrer todos los vértices de la superficie del sólido para obtener una cadena 5OT. Los árboles de bordes (AdBs) son un refinamiento de los árboles envolventes que representan el recorrido a través de un subconjunto de vértices de la superficie que se denomina el borde. Los árboles de bordes se presentan en dos variantes: los árboles de bordes planos (ABPs) y los árboles de bordes generalizados (ABGs). Los objetivos principales de esta tesis han sido:

1. Revisión del código 5OT para generar descriptores de sólidos voxelizados mediante AEs, los cuales recorran los vértices que forman la superficie de los sólidos y preserven sus características geométricas.

2. Determinar las condiciones bajo las cuales un voxel puede ser considerado como parte de un borde y proponer alguna característica que pueda ser fácilmente calculable para detectarlo, análoga a los puntos simples utilizados en algunos algoritmos de esqueletización [LD10]. Con la ventaja que a diferencia de los algoritmos iterativos, no es necesario iterar el proceso de eliminación de voxeles. Al igual que en el caso de los AEs, el árbol obtenido se representará mediante el código de cadenas y debe preservar las características geométricas del objeto inicial. A diferencia de los procedimientos mencionados para la obtención de esqueletos los árboles no se obtienen a través de un proceso de erosión. El árbol debe tener la capacidad de representar objetos con cavidades y túneles que intersecten la superficie, ya que de otra manera no hay forma de acceder a la superficie interna desde la superficie externa del objeto.

3. Explorar la posibilidad de utilizar los baricentros de los voxeles como alternativa para representar un sólido en lugar de los vértices de la superficie como ocurre con los AEs.

4. Investigar aspectos relacionados con la cuantificación de las diferencias entre los objetos originales a partir de árboles envolventes o de bordes. Particularmente se considera de interés sondear algunos algoritmos de

análisis léxico de las cadenas como el reportado en [BA06] y la clásica distancia de Levenshtein [VL66, BB07]. Aprovechando que el árbol como gráfica es intrísencamente descrito por la cadena, resulta viable explorar la similitud mediante la teoría espectral de gráficas así como la distancia de edición entre árboles mediante algún algoritmo conocido como el de Zhang-Shasha [ZS89] y RTED [PA11].

Las principales contribuciones de esta tesis han dado origen a las siguientes publicaciones: un artículo arbitrado [BGM12], un capítulo de libro [BGM10], un artículo de congreso arbitrado [MBG13] y un artículo que actualmente se encuentra en proceso de arbitraje [MBG14]. Sin ánimo de ser exhaustivos, en la Figura 1.1 se muestran las aportaciones presentadas en este documento. Los cuadros sombreados tratan de enmarcar las distintas aportaciones presentadas en esta tesis y cómo se enlazan las dos temas principales que se desarrollan a lo largo del presente trabajo, los códigos de cadenas y la representación de sólidos voxelizados 3D. La confluencia de ambos temas se da alrededor del cálculo de la similitud entre sólidos 3D. A continuación se describen brevemente las aportaciones de esta tesis:

1. Se hace una exposición de los modelos reticulares de puntos y de células. Con base en estos modelos se establece un marco de referencia para agrupar los códigos de cadenas 2D y 3D. En especial, se describe detalladamente el modelo reticular de células (Apéndice A) como antecedente del código 5OT que será el marco de referencia para definir los AEs.

2. Se proponen los AEs como una representación de sólidos voxelizados en el contexto del modelo reticular de células[1] [BGM12].

3. Se propone una extensión de los AEs a $\mathbb{Z}^3$ utilizando los baricentros de los voxeles en lugar de sus vértices en el marco del modelo reticular de puntos.

4. Se propone la definición de los AdBs como una extensión de los AEs. Se dan unas condiciones básicas bajo las cuales un vértice de la superficie de un sólido puede ser considerado o no parte del borde [MBG13].

5. Se propone un algoritmo (Apéndice B) susceptible de ser programado que permite calcular los AdBs y los AEs, así como sus respectivos códigos de cadenas [MBG14].

[1] También se usa ampliamente el término *celdas*.

6. Se proponen como métricas para evaluar la similitud entre sólidos representados por medio de códigos de cadenas 5OT, la distancia de Levenshtein [VL66, BB07], la distancia de edición en árboles [PA11] y la distancia espectral entre gráficas [CRS10].

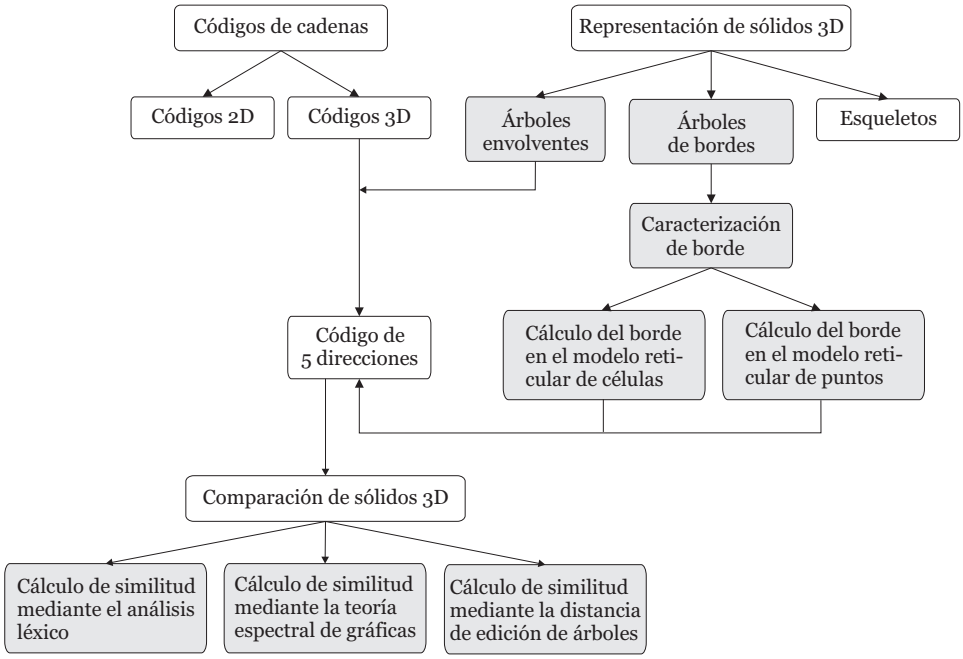


Figura 1.1: Diagrama de los temas desarrollados y cómo se relacionan las dos temas principales tratados a lo largo del presente trabajo. Los cuadros sombreados indican los subtemas en los cuales se centran las aportaciones de esta tesis.

1.3. Organización de la tesis

El texto está dividido en siete capítulos. En el capítulo 2 se introducen las ideas generales sobre los modelos reticulares de puntos y de células. A partir de éstos modelos se plantean las definiciones básicas sobre códigos de cadenas 2D y 3D; en especial, se proporciona un modelo de referencia para el uso del código 5OT. En el capítulo 3 se presenta el código de cinco direcciones 5OT, sus características principales y su uso en la representación de curvas

3D. En el capítulo 4 se utiliza el código 5OT para representar árboles como antecedente para introducir los árboles envolventes (AEs), también se dan las propiedades principales de los AEs y algunos ejemplos. En el capítulo 5 se presentan los árboles de bordes en sus dos variantes como una extensión al concepto de AEs. En el sexto capítulo se exploran algunas métricas para evaluar la similitud entre AEs y AdBs. Finalmente en el último capítulo se plantean las conclusiones y el trabajo futuro. En el apéndice A se hace una exposición detallada del concepto de complejo celular CW (*Closure-finiteness with the Weak topology*) introducido por J. H. C. Whitehead [JM84] y la manera en que tanto los pixeles como los voxeles pueden considerarse espacios topológicos que se van construyendo como una agregación sucesiva de células de dimensiones 0, 1 y 2, como base formal para los conceptos de conexidad y su uso en el procesamiento de imágenes [JM84, Ma82, Ko89].

Capítulo 2

Antecedentes

En este capítulo se presentan las definiciones básicas y las principales ideas relacionadas con los códigos de cadenas. Como punto de partida se introducen códigos de cadenas bidimensionales que se utilizan como ejemplos para presentar aspectos relacionados con los códigos de cadenas que posteriormente se aplicarán al caso 3D.

2.1. Introducción

En el ámbito del procesamiento digital de imágenes, la representación de objetos o regiones obtenidas previamente mediante la segmentación de una imagen constituye un área de estudio de gran interés debido a que los descriptores pueden ser procesados eficientemente. Los esquemas de representación pueden ser clasificados en dos grandes grupos: los descriptores que utilizan la frontera de las regiones y los descriptores que usan las características internas de una región para representarla. Los códigos de cadenas y las aproximaciones poligonales son ejemplos de descriptores del primer grupo, y los esqueletos del segundo grupo [My98]. Los códigos de cadenas pueden ser bidimensionales (2D) o tridimensionales (3D). Este trabajo se centra en el estudio del código de cadenas 3D 5OT que se presenta en el siguiente capítulo y su aplicación a la representación de sólidos tridimensionales.

Los códigos de cadenas han sido ampliamente utilizados porque pueden ser fáciles de calcular y permiten una considerable reducción en la representación de formas, además de que son utilizados como formato estándar de entrada por algoritmos de análisis de formas o reconocimiento de patrones. Entre algunos ejemplos se pueden mencionar sistemas de identificación biométrica [PHB12], el reconocimiento de placas de identificación de

automóviles [JZ12], la cuantificación y clasificación de la tortuosidad en venas de la retina [TOU13] o el cálculo del número de Euler en imágenes binarias [SSP13].

2.2. Definiciones básicas

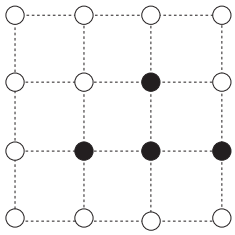
En lo que sigue se utilizarán las definiciones dadas por R. Klette y A. Rosenfeld [KR04], J. Toriwaki y H. Yoshida [TY09], y G. Lohmann [Lo98]. Primeramente se introducirán los conceptos para el caso 2D y posteriormente para 3D.

Definición 2.1. Una *imagen* n-dimensional I, es una función $f : I \subset \mathbb{R}^n \to \mathbb{R}$ tal que cada punto $p \in I$ representa un punto de la imagen, al valor $f(p)$ se denomina la *densidad de la imagen* en p. En el caso de una imagen 2D cada punto se denomina *pixel* y *voxel* para los puntos 3D. Una imagen es *binaria* si la densidad de cada elemento únicamente puede tomar uno de dos valores.

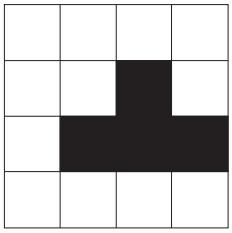
En general, una imagen se digitaliza mediante un proceso que incluye dos pasos principales: el muestreo y la cuantificación. El muestreo puede hacerse de dos maneras, una consiste en muestrear la función $f(p)$ mediante un arreglo ordenado de tal forma que cada elemento del arreglo representa un punto de la imagen. En la otra, la imagen se divide en pequeñas células donde todas tienen la misma forma, tamaño y el valor de la imagen dentro de cada célula es el mismo. Las dos formas de muestreo, que pueden pensarse como equivalentes si los puntos de muestreo de la primera se asocian con los baricentros de las células de la segunda darán origen a los dos modelos reticulares que se comentan posteriormente, y también a dos implementaciones de los códigos de cadenas. En el caso de la cuantificación, a cada elemento de la imagen (o de muestreo) se le asigna una densidad dentro de un conjunto finito de valores posibles.

A lo largo del presente trabajo, únicamente se utilizarán imágenes binarias cuyos valores son $\{0, 1\}$. Los elementos cuya densidad sea 1 representarán al *objeto* de la imagen y se colorearán de negro o bien se indicará cuales son los pixeles o voxeles que forman al objeto, mientras que los elementos cuya densidad sea 0 representarán el *fondo* de la imagen y tendrán color blanco. Generalmente el fondo no aparecerá en las figuras a menos que haya riesgo de confusión. La Figura 2.1 presenta el ejemplo de una imagen binaria de 4×4 pixeles y las dos formas de muestreo. Aunque las definiciones pueden

extenderse al caso n-dimensional, en lo que sigue únicamente se tratarán imagenes bidimensionales y tridimensionales.



(a) Muestreo basado en un arreglo ordenado de puntos.



(b) Mustreo basado en células.

Figura 2.1: Se muestran las dos formas de muestreo en una misma imagen binaria de 4×4 pixeles.

En el caso 2D, se asumirá que las células están conformadas de la siguiente manera: cada punto de $\mathbb{Z}^2$ es el vértice de un cuadrado cuyos lados son de longitud uno y paralelos a los ejes coordenados. El baricentro de cada cuadrado estará desplazado $(0.5, 0.5)$ con respecto a los vértices correspondientes. Por consiguiente, el conjunto de baricentros estará definido como $\overline{\mathbb{Z}^2} = (0.5, 0.5) + \mathbb{Z}^2 = \{(x + 0.5, y + 0.5) : x, y \in \mathbb{Z}\}$. Las coordenadas del pixel en una imagen corresponderán a las coordenadas de su baricentro. En 3D, el correspondiente conjunto de baricentros será $\overline{\mathbb{Z}^3} = (0.5, 0.5, 0.5) + \mathbb{Z}^3 = \{(x + 0.5, y + 0.5, z + 0.5) : x, y, z \in \mathbb{Z}\}$. Las coordenadas de un voxel serán las coordenadas de su baricentro.

Conexidad

En lo anteriormente expuesto, únicamente se ha indicado cómo los elementos en lo individual conforman el objeto en una imagen. Las siguientes definiciones aclaran las relaciones de adyacencia entre ellos.

Definición 2.2. Sean $p, q \in \overline{\mathbb{Z}^2}$ dos puntos distintos con coordenadas (p_x, p_y) y (q_x, q_y), respectivamente.

- Se dice que p y q son *4-adyacentes* si $|p_x - q_x| + |p_y - q_y| = 1$, y
- *8-adyacentes* si $1 \leq |p_x - q_x| + |p_y - q_y| \leq 2$ donde máx$(|p_x - q_x|, |p_y - q_y|) = 1$.

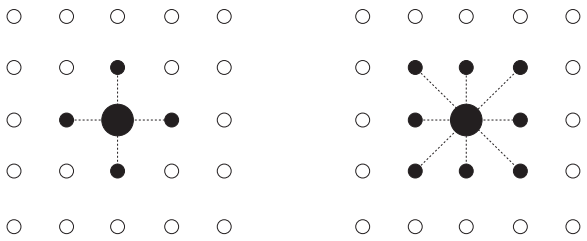


Figura 2.2: Puntos 4 y 8 adyacentes al vértice central.

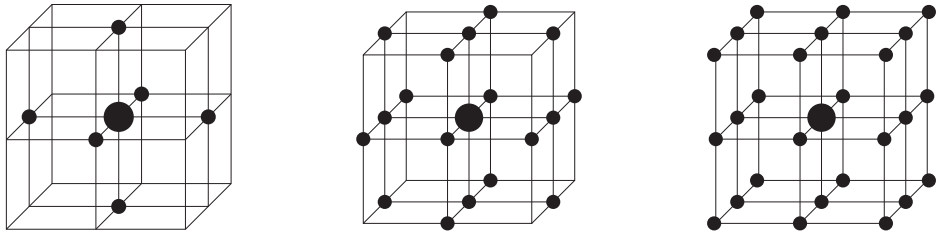


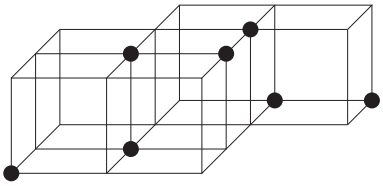
Figura 2.3: Puntos 6, 18 y 26 adyacentes al vértice central.

Definición 2.3. Sean $p, q \in \overline{\mathbb{Z}^3}$ dos puntos distintos con coordenadas (p_x, p_y, p_z) y (q_x, q_y, q_z), respectivamente.

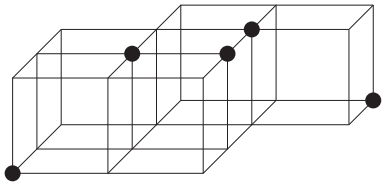
- Se dice que p y q son *6-adyacentes* si $|p_x - q_x| + |p_y - q_y| + |p_z - q_z| = 1$,
- *18-adyacentes* si $1 \leq |p_x - q_x| + |p_y - q_y| + |p_z - q_z| \leq 2$ donde máx$(|p_x - q_x|, |p_y - q_y|, |p_z - q_z|) = 1$, y
- *26-adyacentes* si máx$(|p_x - q_x|, |p_y - q_y|, |p_z - q_z|) = 1$.

Definición 2.4. Dado un punto $p \in \overline{\mathbb{Z}^m}$, $m = 2, 3$; la $k - vecindad$ de p es la unión de p con todos los puntos que sean k-adyacentes a p y se denotará por $N_k(p)$. Una $k - trayectoria$ es una secuencia de puntos $(p_0, p_1, \ldots, p_{n-1})$ en $\overline{\mathbb{Z}^m}$, $m = 2, 3$ tales que tomados de dos en dos son k-adyacentes; esto es, (p_i, p_{i+1}) son k-adyacentes donde $i = 0, \ldots, n-2$. Un conjunto $X \subset \overline{\mathbb{Z}^m}$, $m = 2, 3$ es $k - conexo$ o una $k - componente$, si para cualesquiera dos puntos en X existe una k-trayectoria entre estos dos puntos.

Las Figuras 2.2 y 2.3 muestran los puntos que forman las vecindades en 2D y 3D, respectivamente. La Figura 2.4 presenta dos ejemplos de trayectorias en puntos 18 y 26-adyacentes.



(a) Ejemplo de una 18-trayectoria.



(b) Ejemplo de una 26-trayectoria.

Figura 2.4: Ejemplos de trayectorias para puntos 18 y 26-adyacentes.

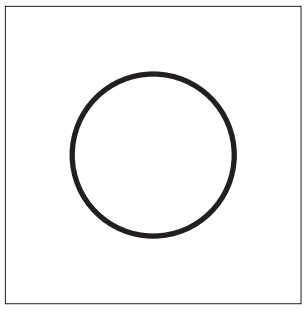
Nótese que las definiciones previas definidas originalmente para baricentros pueden extenderse también a los vértices. Además cabe señalar que los conceptos sobre adyacencia y conexidad ya mencionados pueden aplicarse tanto a los pixeles que conforman un objeto en la imagen como a los del fondo. Sin embargo, esto no significa que pueda utilizarse el mismo tipo de conexidad para el objeto y el fondo como se muestra a continuación.

En la Figura 2.5 se muestra un ejemplo de la razón de este impedimento. El teorema de Jordan establece que toda curva plana simple cerrada divide el plano en dos regiones, una interior acotada y una exterior no acotada [TH07] como se muestra en la Figura 2.5(a). Ahora, considérese la 8-trayectoria cerrada de la Figura 2.5(b). Si se utiliza la 8-adyacencia tanto para la curva como para el fondo, puede notarse que no se cumple la versión discreta del teorema de Jordan ya que el fondo contenido en el interior de la curva y el fondo exterior de la misma constituyen una sola componente. Una manera sencilla de evitar lo anterior es adoptar diferentes conexidades para la curva y el fondo. En este caso como el fondo interior no es 4-conexo con el exterior de la curva, es suficiente asignar una 8-adyacencia para la curva y una 4-adyacencia para el fondo.

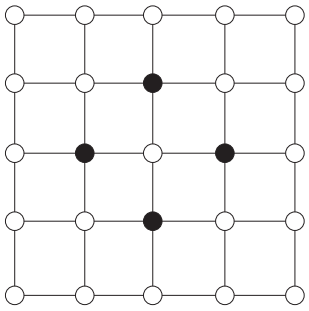
Nótese que la paradoja del teorema de Jordan también ocurre en el caso tridimensional. Las combinaciones de adyacencias que permiten se cumpla el teorema de Jordan son 6 para el fondo y 26 para los objetos, y viceversa. 6 para fondo y 18 para los objetos, y viceversa [Lo98].

Complejos celulares

Un planteamiento alternativo para definir adyacencias que permite sortear las dificultades planteadas por las paradojas es definir las imágenes como *complejos celulares* [Ko89]. Los complejos celulares tienen como base los complejos CW creados por J. H. C. Whitehead en 1949 para el estudio de



(a) Ejemplo de una curva continua cerrada.



(b) Ejemplo de una 8-trayectoria cerrada.

Figura 2.5: La paradoja del teorema de Jordan.

homologías [JM84]; con los cuales, a partir de unidades básicas denominadas *células* que corresponden a puntos, aristas y caras, es posible dar una definición constructiva para el pixel y el voxel (véase el apéndice A).

Definición 2.5. En tres dimensiones considérese el espacio real $\mathbb{R}^3$ dividido en cubos dados por planos paralelos a los ejes coordenados donde cada plano está ubicado en una coordenada entera del eje coordenado correspondiente. Una *0-célula* es un punto de $\mathbb{Z}^3$, una *1-célula* es una línea que conecta dos 0-células 6-adyacentes, una *2-célula* es un cuadrado unitario delimitado por cuatro 1-células, y una *3-célula* es un cubo unitario definido por seis 2-células adjuntas. En el apéndice A, se presentan los elementos básicos de los complejos CW y cómo en el contexto de $\mathbb{R}^3$, las 3-células o voxeles corresponden a cubos sólidos y en $\mathbb{R}^2$ las 2-células o cuadros rellenos a pixeles.

Definición 2.6. Un *complejo celular n-dimensional* es un conjunto C de *m-células*, $m \leq n$, que incluye una relación transitiva, irreflexiva y antisimétrica $B \subset C \times C$ denominada relación facial[1] en la que $\dim(c) < \dim(c')$ para toda $(c, c') \in B$. Donde $\dim(c)$ denota la dimensión de c, $\dim(c) = m$ si c es una m-célula.

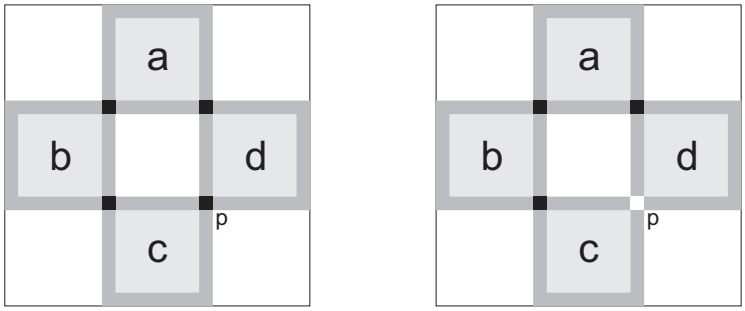
La relación facial asegura que cada célula m-dimensional tiene como frontera un conjunto de células de dimensiones menores que m. Por ejemplo, una 2-célula no puede ser adyacente a otra 2-célula. Una 2-célula estaría acotada por 1-células que corresponderían a las aristas, que a su vez tendrían como frontera un subconjunto de 0-células que serían sus vértices; esto es, dos

[1] en inglés *bounding relation* o *facial relation* [Ko89].

células son adyacentes si una es frontera de otra. La relación facial induce una definición distinta de adyacencia que evita la paradoja del teorema de Jordan.

Definición 2.7. Dos 2-células c_1 y c_2 son 0-*adyacentes* si $c_1 \neq c_2$ y $c_1 \cap c_2$ contiene una 0-célula. Dos 3-células c_1 y c_2 son n-*adyacentes* si $c_1 \neq c_2$ y $c_1 \cap c_2$ contiene una n-célula, $n = \{0, 1, 2\}$.

Definición 2.8. Una secuencia de elementos $(c_0, \ldots, c_{k-1})$ de un complejo celular C es una *trayectoria* si cada par de elementos $(c_i, c_{i+1}), i = 0, \ldots, k-2$ es adyacente. Un subconjunto $D \subset C$ es *conexo* si para cualesquiera dos elementos c_1, c_2 existe una trayectoria de c_1 a c_2.



(a) Ejemplo de una trayectoria cerrada

(b) Ejemplo de una trayectoria abierta.

Figura 2.6: Conexidad en complejos celulares.

Considérense el complejo $C = \{a, b, c, d\}$ de la Figura 2.6 que consta de cuatro 2-células y 4 0-células. C divide al plano digitalizado en dos regiones: una interna acotada por C y el exterior. Estas regiones están separadas ya que no existe al menos una 0-célula que las una (véase la Figura 2.6(a)). De acuerdo con la definición 2.8, toda trayectoria cerrada en C debe pasar por p. Nótese que si p se elimina de C, el conjunto $C - \{p\}$ se convierte en una trayectoria abierta porque p se vuelve parte del fondo, c y d se desconectan porque la 0-célula que las unía se eliminó. Lo cual coincide con lo esperado del teorema de Jordan, ya que cuando p se vuelve parte del fondo, desconecta C pero ahora conecta las regiones interior y exterior originales, evitándose así la paradoja del teorema de Jordan (Figura 2.6(b)).

Modelos reticulares

Con base en las definiciones anteriores se pueden definir dos modelos reticulares alrededor de los cuales se pueden organizar los conceptos básicos en geometría digital: el modelo de puntos y el modelo celular.

- En el *modelo de puntos*[2], un *pixel* es un baricentro y un conjunto de pixels $\mathbb{G}$ en 2D puede ser el conjunto $\overline{\mathbb{Z}^2}$ o un subconjunto $\mathbb{G}_{m,n}$ de tamaño $m \times n$ de $\overline{\mathbb{Z}^2}$, donde $\mathbb{G}_{m,n} = \{(i,j) \in \overline{\mathbb{Z}^2} : 1 \leq i \leq m \text{ y } 1 \leq j \leq n\}$. Para el caso 3D, la retícula puede ser $\overline{\mathbb{Z}^3}$ o un subconjunto de tamaño $l \times m \times n$.

- En el *modelo celular*[3], en 2D un *pixel* es una 2-célula y en 3D un voxel es una 3-célula.

Aunque los dos modelos son básicamente equivalentes, cada uno ofrece una plataforma teórica distinta para explorar conceptos y propiedades en geometría digital [BCK07]. Como puede verse en la sección anterior el mérito del modelo celular es que evita la paradoja del teorema de Jordan. En la Figura 2.7 se muestran las representaciones de un objeto en ambos modelos reticulares.

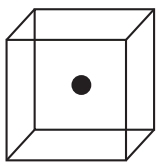
2.3. Códigos de cadenas 2D

En esta sección se presentan dos ejemplos de códigos de cadenas 2D, cada uno desarrollado en modelos reticulares distintos como preámbulo para presentar el código de cadenas 5OT.

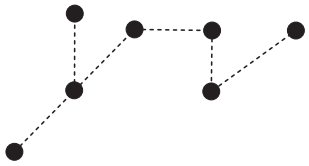
El código de Freeman

La primera propuesta para representar curvas bidimensionales mediante códigos de cadenas fue hecha por H. Freeman en 1961 [HF61]. El código de cadenas de Freeman (CCF) sigue el borde de una figura y va asignando direcciones en sentido contrario a las manecillas del reloj conforme pasa de un pixel a otro. Se requieren cuatro u ocho direcciones, dependiendo de la conectividad utilizada, para representar cambios en los ángulos en múltiplos de 90 o 45°, respectivamente. Las figuras 2.8(a)-(b) muestran el código de cada dirección, el contorno de un objeto y su código de Freeman. El CCF

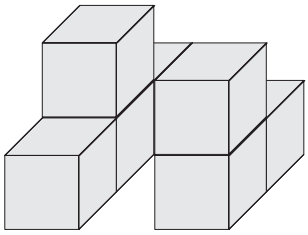
[2] grid point model [KR04]
[3] grid cell model [KR04]



(a) Baricentro de un voxel.



(b) Modelo de puntos.

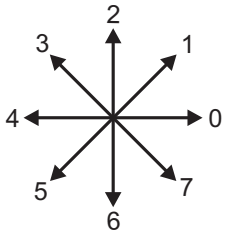


(c) Modelo de células.

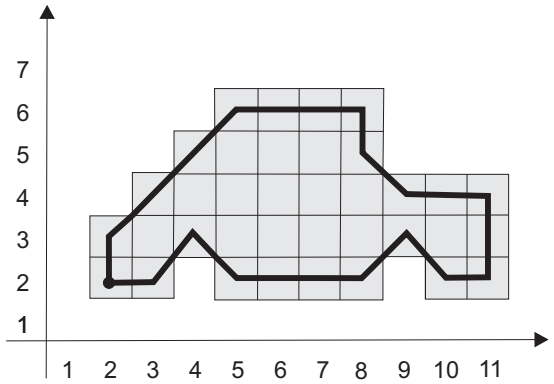
Figura 2.7: Representación de un objeto en los dos modelos reticulares.

puede verse como una secuencia de segmentos rectilíneos con direcciones y longitudes específicas. Aunque no fue explícitamente definido así, el CCF es un ejemplo de código desarrollado en el modelo reticular de puntos.

Un inconveniente del CCF es que las cadenas no son invariantes bajo rotación debido al hecho que emplean direcciones absolutas. Esta deficiencia dio pie a la investigación en este campo para obtener mejores representaciones que tuviesen la misma capacidad de descripción pero que además tuvieran otras propiedades tales como ciertos invariantes a rotación, traslación, obtener la imagen especular, etcétera. Algunos ejemplos de otros esquemas de códigos de cadenas son el que presentó Papert en 1973 [SP73] que es uno de los códigos más simples pues solo contiene dos símbolos $\{0, 1\}$. Algunos desarrollos posteriores han sido el código de cadenas de diferencias direccionales [LZ05], el código compactado de cadenas [LWWZ07] y un código basado en propiedades espaciales y de orientación de patrones [PHB12]. Una característica del CCF es que los segmentos de líneas rectas cambian de longitud para direcciones múltiplos de 45° a $\sqrt{2}l$, donde l es la longitud de los lados paralelos a los ejes coordenados; lo que dificulta el cálculo de elementos secundarios que podrían obtenerse del código de cadenas tales como el perímetro, debido a que los lados pueden tener longitudes distintas.



(a) Direcciones del código de Freeman de 8 direcciones.



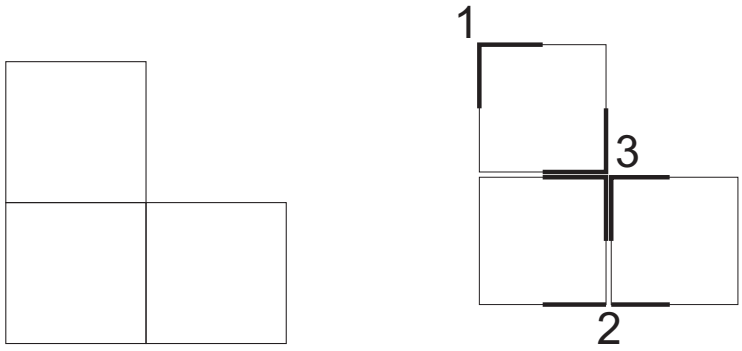
(b) Ejemplo de aplicación del código de Freeman, inicia en $(2, 2)$: 2111000670066435444354.

Figura 2.8: Código de Freeman.

El código VCC

El código *vertex chain code* (VCC) fue introducido en [BG80] por E. Bribiesca y A. Guzmán para generar números de forma; posteriormente, E. Bribiesca [EB99] le dio una interpretación en términos del número de vértices adyacentes en una figura formada por 2-células. El VCC es un ejemplo de código que, aunque no fue explícitamente definido así, hace uso del modelo de retícula celular. Un elemento de la cadena indica el número de vértices de 2-células que coinciden en un punto conforme se va recorriendo el contorno de la forma. En la Figura 2.9(a) se muestra una forma y en la Figura 2.9(b) se indican los tres casos de cadenas que pueden ocurrir al recorrer la forma. Se requiere que las 2-células que integran el objeto sean adyacentes mediante 1-células, esto es, que los pixeles sean adyacentes mediante sus caras. Cabe hacer notar que el VCC puede extenderse a células triangulares y hexagonales.

Por definición, donde las direcciones dependen intrínsecamente de la forma y del objeto y no de una referencia externa, el VCC es invariante a traslaciones, rotaciones y es posible establecer un procedimiento para hacerlo invariante al punto de inicio.



(a) Forma original.

(b) Tipos de cadenas cuando se utilizan 2-células.

Figura 2.9: Código VCC.

2.4. Códigos de cadenas 3D

El primer código para representar curvas en 3D lo propuso H. Freeman en 1974 como una extensión a su código original [HF74]. Este código genera

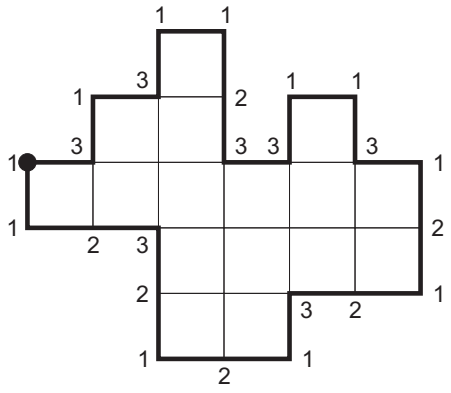


Figura 2.10: Ejemplo de aplicación del código VCC, inicia en el vértice marcado y el recorrido es en el sentido contrario a las manecillas del reloj: 112321213212131133211313.

códigos de cadenas de 26 direcciones.

Los códigos de cadenas 3D pueden ser una alternativa a los esqueletos como formas de representación. Aunque los esqueletos son una forma de representación en sí mismos, los esqueletos pueden ser considerados como un paso intermedio a los cuales se les pueden aplicar los códigos de cadenas para obtener una representación más compacta.

La esqueletización es un esquema de representación muy utilizado que permite obtener una representación de un objeto digital en una dimensión menor [KR04]. Los algoritmos para obtener esta representación también son conocidos como de adelgazamiento o erosión y al resultado obtenido generalmente se le denomina *esqueleto*. Los esqueletos pueden ser muy útiles en aplicaciones que incluyen búsquedas en bases de datos de objetos 3D [Go04] ya que en lugar de analizar directamente los objetos, se comparan los esqueletos que a su vez pueden estar almacenados como un índice del objeto debido a su menor tamaño. Existen dos tipos de procedimientos para adelgazar objetos 3D: ser reducidos directamente a un esqueleto, o pueden ser reducidos inicialmente a una superficie esquelética y posteriormente, ésta se transforma nuevamente en un esqueleto [GS06].

La esqueletización es un proceso iterativo que se lleva a cabo en el modelo reticular de puntos para 3D. Generalmente requiere de la identificación, iteración tras iteración de ciertas superficies frontera del objeto de manera que puedan ser aplicadas las operaciones morfológicas para la eliminación

de voxeles. Además de compactar la información del objeto 3D, los procedimientos de esqueletización deben preservar propiedades topológicas del objeto. El algoritmo clásico del fuego que se propaga en la pradera y consume un objeto 3D que genera una representación medial [DH73] es un ejemplo de proceso de esqueletización. Otros ejemplos de algoritmos de esqueletización muy conocidos son los propuestos por Ma y Sonka [MS96], el de Palágyi y Kuba [PK99]. N. Cornea *et al.* [CSM07] y C. Arcelli *et al.* [ASS11] hacen un recuento exhaustivo de los algoritmos de esqueletización publicados en la literatura. Cabe señalar que los esqueletos para representar objetos 3D tienen una limitación importante y no en todos los casos es posible reconstruir el objeto original a partir de ellos [SNS02]. Los esqueletos también son elementos susceptibles de ser representados mediante el código de cadenas 5OT en ambos modelos reticulares como se muestra en la sección 4.7.

En 2008, E. Bribiesca [EB08] propuso un método para representar objetos arborescentes tridimensionales mediante un código de cinco direcciones. Posteriormente, en el artículo [BGM12] extendimos el uso de este método al definir los árboles envolventes (AEs) en el modelo reticular de células para representar sólidos usando el mismo código de cadenas. Cabe mencionar que en 2014 se publicaron dos propuestas para representar sólidos voxelizados. La primera propuesta publicada por Sánchez-Cruz *et al.* [SC14] está basada en la extensión del código de Freeman al espacio tridimensional y la segunda publicada por Lemus *et al.* [LBG14] en la que se obtiene un ciclo hamiltoniano en la gráfica de adyancencias de la superficie del sólido mediante un código de cadenas de nueve valores que representan cambios de dirección al recorrer el sólido.

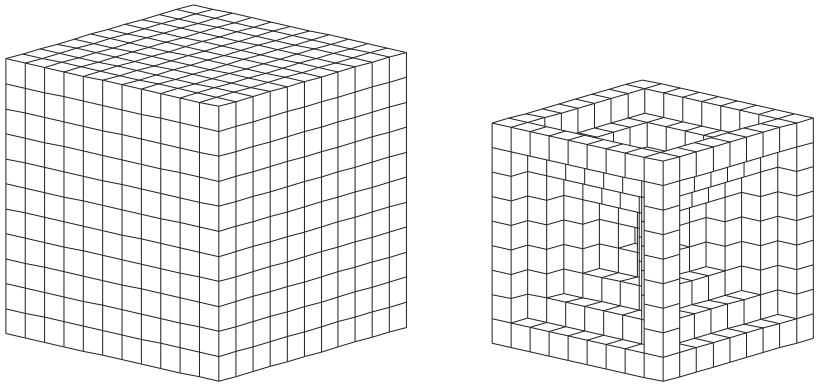
Dado un sólido 3D, un AE es un árbol en el cual cada vértice corresponde a un nodo del árbol que a su vez puede ser transformado a un código de cadena. Los AEs son árboles con grado máximo 6 embebidos en $\mathbb{Z}^3$. Un código de este tipo facilita la comparación de árboles en 3D a través de la inspección únicamente de la representación de cada árbol, ya que el código descriptor es invariante bajo rotaciones, traslaciones y transformaciones especulares además de preservar sus características topológicas. Aunque los AEs ofrecen una reducción de la información necesaria para representar un objeto 3D, la característica fundamental de los AEs de pasar por todos los vértices que forman la superficie del sólido puede llevar a una sobre-representación del objeto ya que existen sólidos cuya superficie es susceptible de ser descrita sin utilizar todos los vértices.

Partiendo de estas ideas, es claro que puede extenderse el concepto de los AEs al utilizar únicamente los vértices que formen el borde del objeto para

simplificar el árbol y por consiguiente, reducir aún más la longitud del código de cadena que representa al objeto original. Con lo anterior se obtendrían árboles, denominados *árboles de bordes* (AdBs), para representar sólidos que pueden ser más ventajosos que los AEs originales.

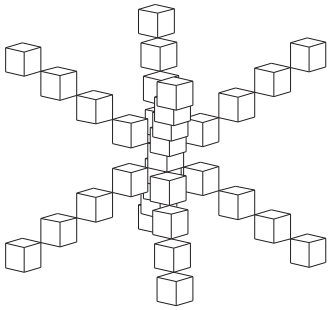
En la figura 2.11 se muestran una superficie esqueletal (b) y el esqueleto obtenido (c) con el algoritmo de fuego en la pradera para un sólido de $11 \times 11 \times 11$ voxeles. Las figuras 2.11(d) y (e) corresponden a un árbol envolvente y a un árbol de bordes respectivamente. Obsérvese que tanto la superficie esqueletal como el esqueleto podrían se representados con AEs.

De lo expuesto anteriormente se desprenden las líneas de trabajo que serán expuestas en los siguientes capítulos: la descripción del código 5OT, su uso en la representación de árboles 3D con la ayuda de pares de paréntesis "( )", y posteriormente la representación de sólidos con AEs y AdBs; para finalizar con la comparación de la similitud entre objetos representados por medio de códigos de cadenas 5OT.

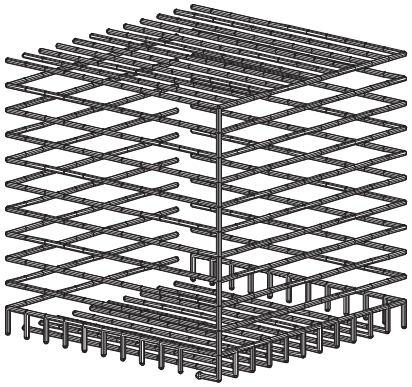


(a) Sólido original

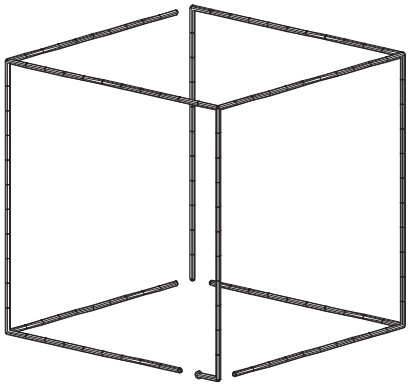
(b) Superficie esqueletal



(c) Esqueleto



(d) Árbol envolvente



(e) Árbol de bordes planos

Figura 2.11: Ejemplos de esqueletos y árboles calculados para un sólido.

Capítulo 3

El código de cadenas 5OT

En este capítulo se presenta el código de cadenas 5OT; se plantean los aspectos preliminares, su definición y un ejemplo del cálculo del código de cadenas para una curva 3D. Asímismo, se listan sus propiedades más importantes.

3.1. Introducción

En 1974 H. Freeman introdujo un método para representar curvas digitales tridimensionales [HF74] basado en un código de 26 direcciones que tuvo su origen en el código original 2D publicado años antes [HF61]. Tiempo después, con base en una notación de A. Guzmán para representar objetos 3D compuestos de segmentos ortogonales [AG87], E. Bribiesca desarrolló un código de cadenas (denominado 5OT por H. Sánchez-Cruz [SCB08], notación que se utiliza en este trabajo) para representar curvas en el espacio [EB00]. Posteriormente, se agregaron los paréntesis "()" a su alfabeto para poder representar árboles tridimensionales [EB08]. Adicionalmente, este código también ha sido utilizado para definir una métrica de disimilitud[1] para comparar curvas 3D [BA06] y se ha mostrado que es un código con una mayor capacidad de compresión que el código de Freeman [SCB08].

A continuación se hacen algunas precisiones preliminares que anteceden a la definición del código 5OT.

[1] dissimilarity en inglés

3.2. Conceptos preliminares

Antes de proceder a la presentación del código 5OT es necesario mencionar algunos conceptos generales de la teoría de gráficas estándar [BM08].

a) Una *gráfica* $G = (V, E)$ es un par ordenado que consiste de un conjunto $V = \{v_1, v_2, \ldots, v_n\}$ de vértices y un conjunto $E = \{e_1, e_2, \ldots, e_m\}$ de aristas. Si una arista e une un par de vértices $\{u, v\}$, éstos serán los *extremos* de e, y se dirá que u y v son *adyacentes*. $|V|$ y $|E|$ denotan el número de vértices y aristas, respectivamente. $|V|$ es el *orden* y $|E|$ el *tamaño* de la gráfica. Si los extremos de una arista e son el mismo vértice, dicha arista será un *lazo*. Si dos o más aristas tienen los mismos extremos, entonces se dirá que son aristas *paralelas*. Una gráfica es *simple* si no tiene lazos ni aristas paralelas. El *grado* de un vértice v denotado por $d(v)$ es el número de aristas que inciden en v.

b) Sea $G = (V, E)$ una gráfica simple, una *trayectoria* en G es una secuencia de aristas $e_1, e_2, \ldots, e_n$ tales que

   1. e_i y e_{i+1} tienen un extremo común;
   2. si e_i no es un lazo, ni es la primera o la última arista, entonces e_i comparte uno de sus extremos con e_{i-1} y el otro con e_{i+1}.

   También es posible describir una trayectoria mediante una sucesión de vértices y aristas: $v_0 \overset{e_1}{—} v_1 \overset{e_2}{—} \cdots v_{n-1} \overset{e_n}{—} v_n$, donde v_0 y v_n son los vértices inicial y final, respectivamente. La longitud de la trayectoria es n. Una trayectoria es *simple* si ningún vértice aparece más de una vez. Una trayectoria simple con tres o más vértices es un *ciclo* si los vértices inicial y final son el mismo.

c) Una gráfica es *conexa* si existe una trayectoria entre cualesquiera dos vértices de la gráfica. En lo que resta del texto únicamente se considerarán gráficas no dirigidas.

d) Un *árbol* es una gráfica conexa sin ciclos. Una *hoja* es un vértice de grado uno.

Una premisa importante que se adoptará a lo largo del presente trabajo será que cada objeto está aislado del mundo real y que fue el resultado de un proceso de digitalización. En lo que respecta a los espacio $\mathbb{R}^3$ y $\mathbb{Z}^3$ se harán las siguientes consideraciones:

e) En lo que sigue se asumirá el modelo reticular de células.

f) Las curvas discretas 3D estarán compuestas de 1-células; esto es, de segmentos rectilíneos de longitud uno inmersos en el espacio $\mathbb{Z}^3$, de tal manera que a partir una curva inicial continua en 3D se obtendrá una representación discreta 3D que podrá ser descrita mediante el código de cadenas 5OT. La Figura 3.1 muestra este esquema de representación, la curva (a) es la curva continua original y (b) es la versión discreta de (a) que se asume inmersa en el modelo reticular de células en $\mathbb{Z}^3$ como se indica en el acercamiento a uno de sus extremos.

g) Los árboles discretos 3D estarán formados de 1-células de manera semejante a las curvas, como se muestra en la Figura 3.2(b). De hecho, una curva puede considerarse un árbol discreto.

h) Los sólidos voxelizados estarán constituidos por conjuntos de 3-células 2-adyacentes; es decir, son sólidos cuyos voxeles tienen caras comunes. Por consiguiente; los lados, caras y el volumen de cada voxel será unitario.

i) Dado un objeto digitalizado $\mathcal{O}$ (una curva, un árbol o un sólido), se le puede asociar de manera natural una gráfica de adyacencia $G_{\mathcal{O}}$ en la que cada nodo de la gráfica corresponde a un vértice (0-célula) del objeto y cada arista está asociada a una 1-célula que forma parte del objeto. La Figura 3.2(a) es un ejemplo de gráfica que describe una curva discreta en 3D. La trayectoria $v_0 \overset{e_1}{—} v_1 \overset{e_2}{—} v_2 \overset{e_3}{—} v_3 \overset{e_4}{—} v_4 \overset{e_5}{—} v_5 \overset{e_6}{—} v_6 \overset{e_7}{—} v_7 \overset{e_8}{—} v_8$ representa la curva formada por los vértices y las aristas con líneas más gruesas (en la Figura se omitieron las etiquetas de las aristas). La Figura 3.2(b) es un ejemplo de gráfica generada por un árbol embebido en el modelo reticular de células. La Figura 3.2(c) muestra la gráfica del sólido formado por los 5 voxeles sombreados.

3.3. El código 5OT

Dados un objeto digitalizado $\mathcal{O}$, y su gráfica de adyacencia $G_{\mathcal{O}}$ inmersa en $\mathbb{Z}^3$ como se indicó en el inciso i) de la sección anterior, el código 5OT permite describir los cambios de dirección conforme se va recorriendo $G_{\mathcal{O}}$. El código 5OT utiliza el producto vectorial de las direcciones determinadas por las aristas (por *dirección* se entenderá un vector de longitud uno), las cuales se van descubriendo al tiempo que se recorren los vértices que integran el objeto. Dos aristas determinan un cambio de dirección ortogonal y dos cambios de

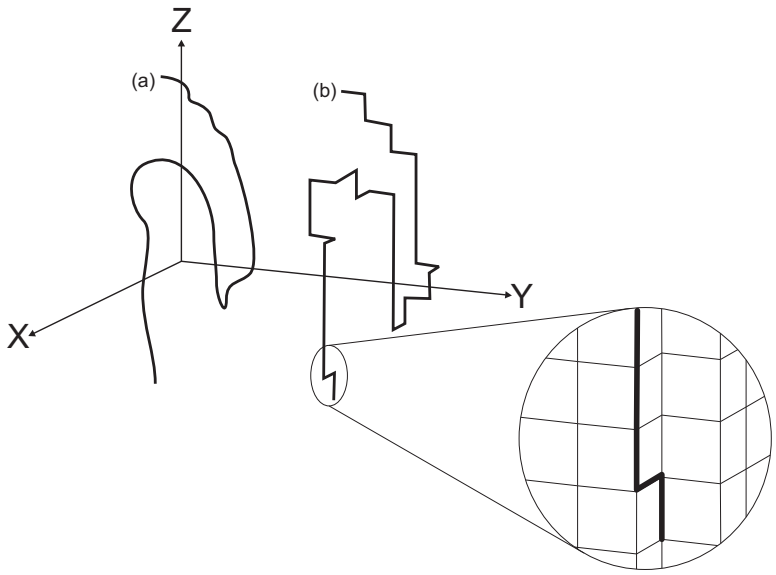
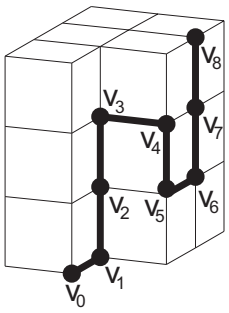
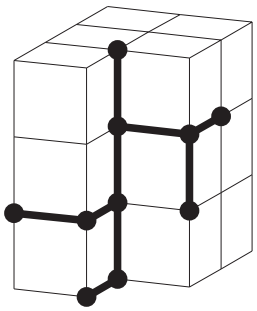


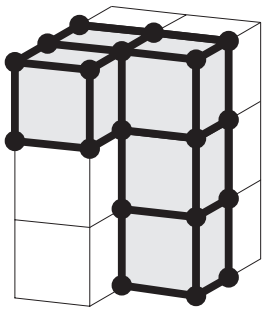
Figura 3.1: Ejemplo de una curva 3D: (a) curva 3D continua, (b) representación discreta de la curva mostrada en (a). Se muestra un fragmento de la curva (b) embebido en el modelo reticular de células en $\mathbb{Z}^3$.



(a) Ejemplo de la gráfica de una curva 3D.



(b) Ejemplo de la gráfica de un árbol 3D.



(c) Ejemplo de la gráfica de un sólido.

Figura 3.2: Ejemplos de gráficas formada por tres objetos: una curva, un árbol y un sólido. Los puntos son las 0-células que corresponden a los vértices de las gráficas y las 1-células a las aristas. En (a) se omitieron las etiquetas de las aristas.

dirección definen un elemento en la cadena. La Figura 3.2(a) ejemplifica una manera de obtener los cambios de dirección. Los vértices v_0, v_1 y v_2 conforman las dos aristas contiguas que definen los vectores $\boldsymbol{b} = v_1 - v_0$ y $\boldsymbol{c} = v_2 - v_1$. Los valores del código 5OT estarán determinados por la aplicación sucesiva del producto $\boldsymbol{b} \times \boldsymbol{c}$.

Definición 3.1. Una *cadena* a es una secuencia ordenada de n elementos tomados del conjunto $\{0, 1, 2, 3, 4\}$ que se denota por $a = a_1 a_2 \dots a_n = \{a_i : 1 \leq i \leq n\}$. Cada elemento es un valor asignado a un vértice e indica el cambio de dirección ortogonal definido por dos aristas contiguas. Si las dos aristas de un ángulo dado tienen direcciones $\boldsymbol{b}$ y $\boldsymbol{c}$ (véase la Figura 3.3), y la siguiente arista en el vértice por evaluar tiene dirección $\boldsymbol{d}$, el elemento de la cadena en esa posición estará determinado por la siguiente función:

$$\text{elemento de la cadena}(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}) = \begin{cases} 0, \text{ si } \boldsymbol{d} = \boldsymbol{c}; \\ 1, \text{ si } \boldsymbol{d} = \boldsymbol{b} \times \boldsymbol{c}; \\ 2, \text{ si } \boldsymbol{d} = \boldsymbol{b}; \\ 3, \text{ si } \boldsymbol{d} = -(\boldsymbol{b} \times \boldsymbol{c}); \\ 4, \text{ si } \boldsymbol{d} = -\boldsymbol{b}; \end{cases} \tag{3.1}$$

donde $\times$ denota el producto cruz en $\mathbb{R}^3$. Las direcciones $\boldsymbol{b}$ y $\boldsymbol{c}$ constituyen la *manija*. En la Figura 3.3 se muestran las cinco direcciones correspondientes al código 5OT. Nótese que cuando se obtiene un valor "0", se debe utilizar la misma manija para calcular el siguiente valor de la cadena hasta que se obtenga un valor distinto de "0", ya que la función 3.1 utiliza direcciones $\boldsymbol{b}$ y $\boldsymbol{c}$ que previamente forman ángulos ortogonales no nulos. El vértice inicial de la manija se indica con la esfera.

3.4. Ejemplo de uso del código 5OT

En la Figura 3.4 se muestra un ejemplo de la aplicación del código 5OT para representar una curva discreta en 3D (Los voxeles se muestran sólo como referencia). El vértice de inicio está indicado por la esfera y en cada paso, la manija se indica en negro. En (a) se observa que la curva continúa a través de una dirección "0". Puesto que el valor es "0", se utiliza la misma manija para calcular el siguiente valor de la cadena que en este caso es un "1" (b). Para calcular el tercer cambio de dirección, la manija debe avanzarse sobre la curva de modo que la función (3.1) pueda aplicarse nuevamente, en este caso el cambio de dirección corresponde a un "4" (c). En las Figuras

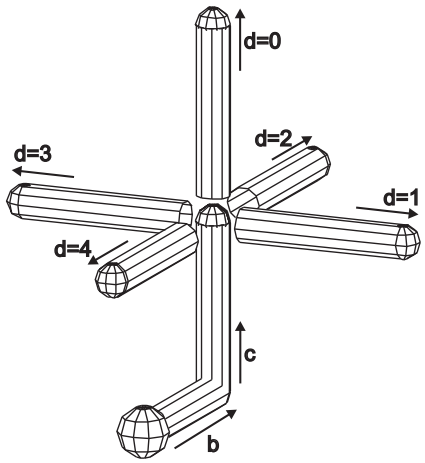


Figura 3.3: Dadas dos direcciones que forman un ángulo recto ***b*** y ***c*** en la manija, se muestran los cinco valores que se pueden asignar a una cadena dependiendo de la dirección ***d***. La esfera indica el vértice inicial de la manija.

(d)-(h) se presentan las cadenas restantes. Al final, curva queda representada por la cadena 01414031.

3.5. Propiedades del código de cadenas 5OT

Algunas de las propiedades más importantes del código 5OT son las siguientes [EB00]:

1. *Invariante bajo rotaciones.* El código de cadenas es invariante bajo rotaciones debido a que utiliza cambios relativos de direcciones que dependen de la curva misma y no de un marco de referencia externo como los ejes cartesianos. La manija inicial que es la que determina el inicio del recorrido sobre la curva y la manera en que se asignan las direcciones de acuerdo con la función (3.1).

2. *Invariantes bajo traslaciones y escalamientos.* Debido a que en una traslación las manijas inicial y subsiguientes se desplazan en la misma medida que la curva misma, la cadena que representa la curva no se afecta. Si el escalamiento es aplicado a todo el objeto, las longitudes de todos los segmentos de la curva (incluyendo la manija) cambiarán en la misma proporción, teniéndose como consecuencia que el proceso

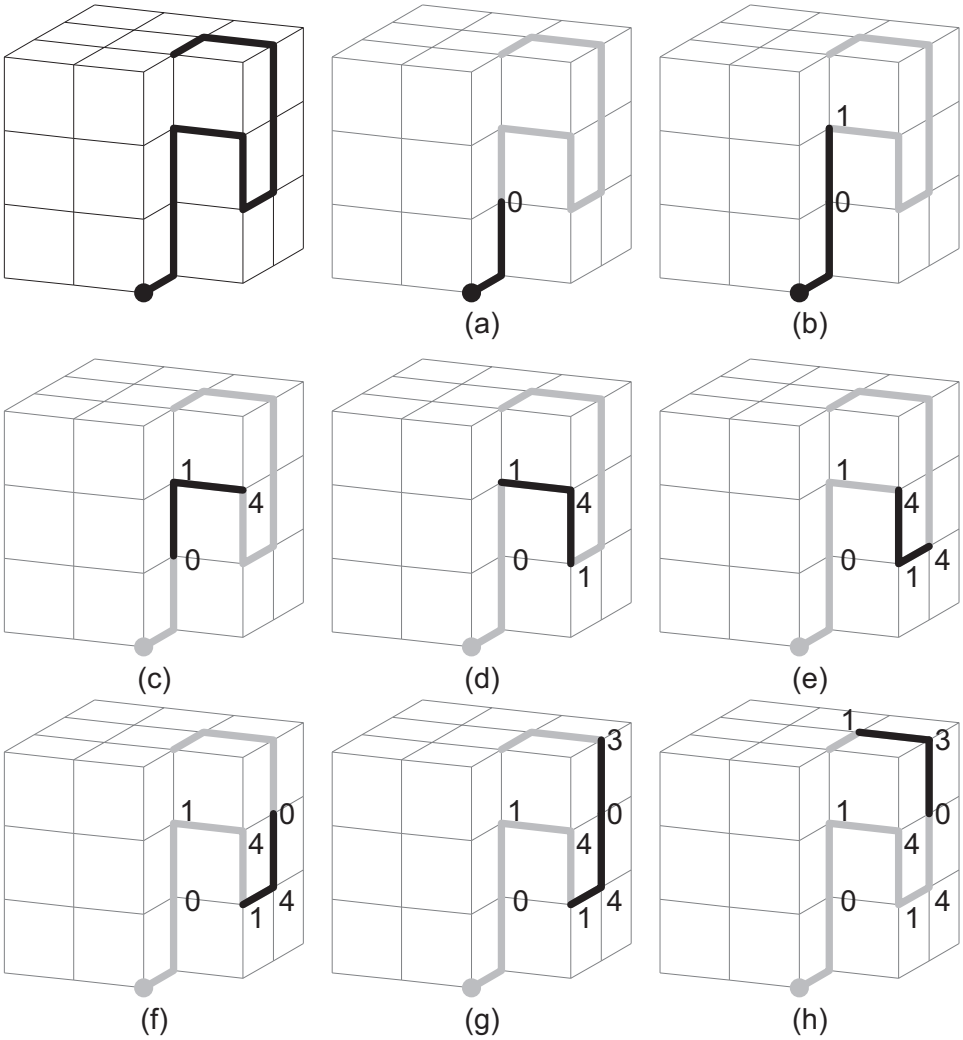


Figura 3.4: Procedimiento para el cálculo de la cadena 5OT de una curva 3D.

de cálculo de la cadena para la curva en relación a la curva de tamaño original no cambia. Por consiguiente, es posible afirmar que el código 5OT es invariante a translaciones y escalamientos.

3. *Facilidad en el cálculo de la imagen especular.* La imagen especular de una curva se puede obtener fácilmente intercambiando las direcciones 1 por 3 y viceversa. Esta modificación corresponde a un cambio de signo en la dirección en la función (3.1) como puede observarse en la Figura 3.3.

4. *La inversa de la cadena.* Dada una cadena a, la inversa de a es otra cadena formada al recorrer a en sentido inverso. Si a es una cadena que representa una curva $\mathcal{C}$, la cadena inversa de a a su vez representa a $\mathcal{C}$ recorrida en sentido inverso. Es importante señalar que cuando la cadena tiene elementos "0" se debe considerar un corrimiento de algunas secciones de la cadena [EB08].

5. *Invariante al punto de inicio en curvas abiertas.* Dada una curva digitalizada $\mathcal{C}$, sean a su cadena y a' su inversa. Si ambas cadenas se consideran como números en base 5, la cadena que represente al número menor puede adoptarse como la que represente a $\mathcal{C}$. Por ejemplo, si $\mathcal{C}$ recorrida en un sentido es representada por el código $a = 2334123$ y en sentido inverso por $a' = 3214332$, entonces se puede adoptar el primer código como la representación de $\mathcal{C}$ porque corresponde al entero menor entre los dos posibles.

6. *Invariante al punto de inicio en curvas cerradas.* En el caso de las curvas cerradas, es posible recorrer la curva e ir escogiendo cada vértice como punto de inicio para calcular el código de cadenas. Dado el conjunto de códigos obtenidos, a cada código se le puede asociar su entero en base 5. Escogiendo aquel código que tenga el número entero de magnitud menor como en el inciso anterior, se obtiene una representación única para la curva.

7. *Comparación de curvas.* A través de sus cadenas es posible establecer elementos de comparación entre curvas; por ejemplo, el número de segmentos que constituyen la curva (longitud) o si tienen algún grado de similitud mediante la búsqueda de secuencias comunes en métodos de comparación de cadenas mencionados en el capítulo 6.

Capítulo 4

Árboles envolventes (AEs)

En este capítulo se extiende el código 5OT para ser usado en la representación de árboles y sólidos voxelizados. El procedimiento para obtener una representación 5OT de árboles es la base a partir de la cual se obtiene la representación de sólidos. En la primera parte del capítulo se desarrollan los árboles envolventes en el modelo reticular de células. Posteriormente se aplica a ejemplos en el modelo reticular de puntos.

4.1. Introducción

En el ámbito de los objetos 3D se han buscado alternativas que favorezcan la abstracción de las características de los sólidos y faciliten tanto su representación como el análisis de los mismos. En particular la esqueletización ha atraído la atención porque genera objetos de menor dimensión que mantienen las propiedades topológicas de los objetos originales. Sin embargo, entre sus desventajas más importantes se encuentran la dificultad para recuperar el objeto original subyacente [SNS02]. Cabe señalar que si bien las superfices esqueletales y los esqueletos unidimensionales son representaciones que capturan ciertas características básicas del objeto subyacente de una manera compacta, generalmente los algoritmos propuestos en la literatura utilizan sus propias definiciones y parámetros que generalmente dependen fuertemente de las aplicaciones para las cuales fueron diseñados haciendo difícil su uso y la evaluación de su desempeño en otros ámbitos [CSM07][ASS11].

La Figura 2.11 muestra el ejemplo de la superficie esqueletal y el esqueleto calculados para un cubo de $11 \times 11 \times 11$ ($= 1331$) voxeles con el algoritmo que genera la representación medial [DH73] que comúnmente se llama fuego en la pradera (*prairie fire* en inglés). Se puede observar que tanto la superficie

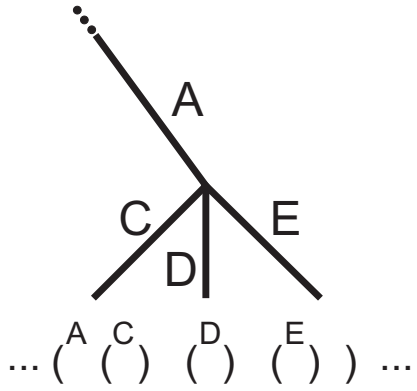


Figura 4.1: Ejemplo de representación de las subramas de una rama utilizando el anidamiento de paréntesis.

esqueletal como el esqueleto todavía preservan en alguna medida la forma del objeto inicial, sin embargo en objetos más complejos la geometría de los mismos se pierde completamente. Los árboles envolventes (AEs) se plantean como una alternativa basada en el código 5OT para la representación de objetos voxelizados que preserva la forma y sus características geométricas [BGM12].

Dado un sólido voxelizado en el contexto del modelo reticular de células, un AE recorre todos los vértices de la superficie del sólido. El AE se representa utilizando el código de cadenas 5OT presentado en el capítulo anterior. Los AEs describen a los sólidos por medio de árboles de grado máximo 6, debido a que los sólidos a los cuales se les aplicarán los AEs deberán ser 2-adyacentes; esto es, dos voxeles adyacentes deben tener una cara común. Nótese que los AEs también pueden ser utilizados en el modelo reticular de puntos utilizando los baricentros de los voxeles como se mostrará más adelante en el capítulo.

Entre las propiedades más importantes de los AEs se encuentran las siguientes: preservan la topología y la geometría de los objetos 3D, son invariantes a rotaciones y traslaciones. Adicionalmente a partir de ellos es posible obtener fácilmente la imagen especular. Esta representación permite reducir la cantidad de información para describir un sólido puesto que únicamente se almacena la cadena obtenida.

4.2. Representación de árboles 3D

Para representar árboles 3D, es necesario extender el código 5OT agregando los paréntesis "()" a los dígitos $\{0, 1, 2, 3, 4\}$ previamente definidos. Los paréntesis anidados se usarán para describir las ramificaciones de los árboles. La Figura 4.1 muestra la idea del anidamiento de los paréntesis. Dada una rama A de la que se derivan las subramas C, D y E, se utiliza un par de paréntesis para separar cada rama.

Con la finalidad de simplificar la presentación del algoritmo que obtiene el descriptor de un árbol [EB08], el procedimiento se dividió en dos fases. La primera fase se describe en el Algoritmo 1 que determina la manija inicial y tiene como entrada principal la gráfica del árbol 3D. Entrega como salida un árbol cuyas aristas están etiquetadas con la función (3.1). La segunda fase está descrita en el Algoritmo 2, recibe como entrada el árbol con aristas etiquetadas de la fase anterior y tiene como salida la cadena del árbol.

En la Figura 4.2 se muestra un ejemplo de árbol 3D y el procedimiento de cálculo de su código de cadenas 5OT. Como extensión al código 5OT mostrado en el capítulo anterior, se incluye el uso de paréntesis anidados para representar las ramificaciones del árbol. En las Figuras 4.2(b) y (c) se exhiben el inicio del procedimiento y el uso de los paréntesis "()" en las cadenas, respectivamente.

Se escogió como vértice inicial la hoja marcada con el punto más grueso. La Figura 4.2(c) muestra el origen seleccionado con una esfera. Los vértices que forman la manija son v_0, v_1 y v_2. Al inicio las variables del Algoritmo 1 tienen los siguientes valores:

$V_{\mathcal{A}} = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$, $\mathbf{Q} = \emptyset$, $V_T = E_T = \emptyset$.

Después de la selección de la manija inicial, los valores son los siguientes: $V_{\mathcal{A}} = \{v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$, $\mathbf{Q} = \{(v_0, v_1, v_2)\}$, $V_T = \{v_0, v_1, v_2\}$ y $E_T = \{v_0{-}v_1, v_1{-}v_2\}$.

En la siguiente iteración, la única dirección posible en la Figura 4.2(b) es "0". Por consiguiente, la etiqueta de la arista $\{v_3{-}v_2$ tendrá etiqueta "0", $V_{\mathcal{A}} = \{v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$, $\mathbf{Q} = \{(v_1, v_2, v_3)\}$, $V_T = \{v_0, v_1, v_2, v_3\}$ y $E_T = \{v_0{-}v_1, v_1{-}v_2, v_2 \overset{0}{-} v_3\}$.

En el vértice v_6 se encuentra una bifurcación que ofrece dos caminos correspondientes a los elementos "1" y "3". En el recorrido del árbol se deben descubrir los nodos disponibles primero a través de la cadena "1" y entonces continuar con el nodo de la cadena "3". Nótese que si algún nodo ya fue

Algoritmo 1 Fase 1 para calcular el código 5OT de un árbol 3D.

Entrada: El conjunto de vértices $V_{\mathcal{A}}$ y aristas $E_{\mathcal{A}}$ de un árbol digitalizado $\mathcal{A} \subset \mathbb{Z}^3$. Una cola sencilla $\mathbf{Q}$ de triadas ordenadas (u, v, w) donde u, v, w son los vértices a partir de los cuales se formará la manija para descubrir los vértices que siguen a u.

Salida: Una gráfica $\mathbf{T}_{\mathcal{A}} = (V_T, E_T)$ que representa al árbol $\mathcal{A}$.

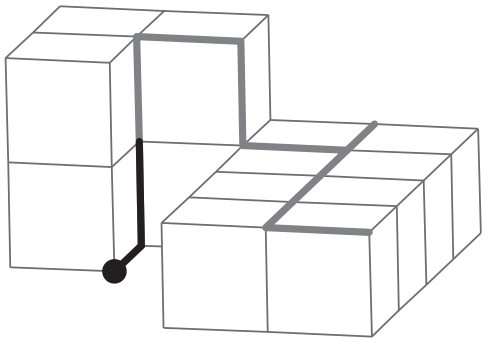
1: Seleccionar un vértice inicial $v_0 \in V_{\mathcal{A}}$ que sea una hoja en $\mathcal{A}$.
2: Seleccionar dos vértices $v_1 \in V_{\mathcal{A}} \cap N_6(v_0)$ y $v_2 \in V_{\mathcal{A}} \cap N_6(v_1)$ tales que $v_1 - v_0$ y $v_2 - v_1$ sean aristas ortogonales en $\mathcal{A}$.
// La arista $v_2 - v_1 \in E_{\mathcal{A}}$ se representa con el tipo de letra normal y la diferencia de vectores se indicará con negritas: $\boldsymbol{v_2 - v_1}$
3: $V_T \leftarrow \{v_0, v_1, v_2\}, E_T \leftarrow \{v_0 - v_1, v_1 - v_2\}$ // Se agregan a $\mathbf{T}$ los vértices v_0, v_1 y v_2 con sus respectivas aristas
4: $V_{\mathcal{A}} \leftarrow V_{\mathcal{A}} - \{v_0, v_1, v_2\}$
5: $\mathbf{Q} \Leftarrow (v_0, v_1, v_2)$ // $\mathbf{Q} \Leftarrow$ indica la operación de agregar un elemento al final de $\mathbf{Q}$
6: **while** $\mathbf{Q} \neq \emptyset$ **do**
7: $(u, v, w) \Leftarrow \mathbf{Q}$ // $\Leftarrow \mathbf{Q}$ indica la operación de obtener el primer elemento de $\mathbf{Q}$
8: $U \leftarrow V_{\mathcal{A}} \cap N_6(w)$ // U es un conjunto auxiliar temporal
9: $V_{\mathcal{A}} \leftarrow V_{\mathcal{A}} - U$
10: $\boldsymbol{b} \leftarrow \boldsymbol{v} - \boldsymbol{u}$
11: $\boldsymbol{c} \leftarrow \boldsymbol{w} - \boldsymbol{v}$
12: **while** $U \neq \emptyset$ **do**
13: $m \leftarrow \min\big(\{\text{elemento de la cadena}(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{\alpha} - \boldsymbol{w}) : \alpha \in \mathbf{U} \text{ y } \alpha - w \in E_{\mathcal{A}}\}\big)$
14: $V_T \leftarrow V_T \cup \{\alpha_m\}$ // α_m es el vértice que arroja el valor mínimo en la línea 13
15: $E_T \leftarrow E_T \cup \{w \underline{m} \alpha_m\}$ // $\{w \underline{m} \alpha_m\}$ es la arista cuya etiqueta m es el elemento de la cadena obtenido en la línea 13
16: $\mathbf{Q} \Leftarrow (v, w, \alpha_m)$
17: $U \leftarrow U - \{\alpha_m\}$
18: **end while**
19: **end while**

Algoritmo 2 Fase 2 para calcular el código 5OT de un árbol 3D.

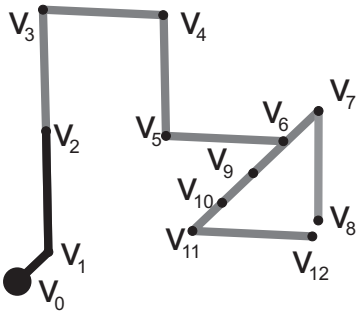
Entrada: El árbol $\mathbf{T} = (V_T, E_T)$ proveniente de la primera fase. Se asume que cada vértice $u \in V_T$ tiene una variable a_u que guarda las cadenas de los vértices sucesores a u en $\mathbf{T}$, en el caso de las hojas de $\mathbf{T}$ esta cadena es vacía.

Salida: Una cadena a que representa al árbol $\mathcal{A}$ que corresponde a un vértice de la última arista de **LI**.

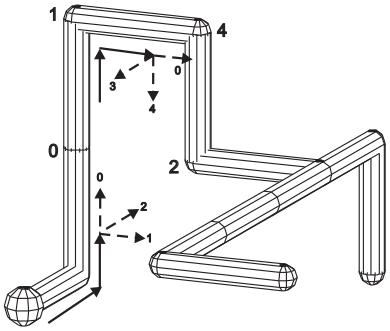
1: $\mathbf{L} \leftarrow$ la lista de aristas de $\mathbf{T}$ recorridas mediante BFS a partir del vértice inicial
2: $\mathbf{LI} \leftarrow$ la lista L en el orden invertido
3: **while** $|\mathbf{LI}| > 2$ **do**
4: $\{u \overset{m}{-} v\} \Leftarrow \mathbf{LI}$ // $\Leftarrow \mathbf{LI}$ indica la operación de obtener el primer elemento de **LI**
5: $n \leftarrow$ sucesores(u) // sucesores(u) es el número de vértices sucesores a u en $\mathbf{T}$
6: **if** $n = 1$ **then**
7: $a_u \leftarrow m_v + a_v$ // El signo + indica la concatenación de cadenas de caracteres, m_v es la etiqueta de la arista $\{u \overset{m}{-} v\}$.
8: **else**
9: $a_u \leftarrow (m_{v_0}) + \cdots + (m_{v_4}) + a_v$ // m_{v_i} es la etiqueta de cada arista que une u con los vértices sucesores que pueden ir desde v_0 hasta v_4.
10: **end if**
11: **end while**



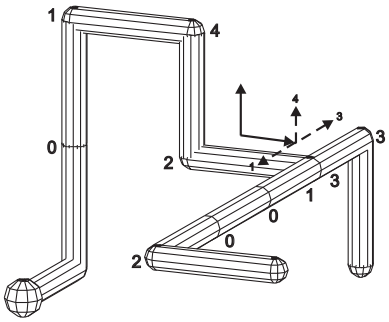
(a) Ejemplo de curva 3D, el inicio de la manija (en negro) se indica con una esfera. En gris se indica el resto del árbol que será descrito por la cadena 5OT.



(b) Numeración de los vértices utilizada en el ejemplo.



(c) Los primeros cuatro elementos del descriptor parcial son 0142.



(d) El descriptor completo del árbol: 0142(1002)(33).

Figura 4.2: Ejemplo del cálculo de los elementos de una cadena definidos por la función (3.1) para el caso de un árbol 3D.

ocupado por alguna iteración previa, el proceso no puede generar una cadena en esa dirección. Los paréntesis indican una rama a partir de una cadena. Las ramas deberán ser recorridas en orden creciente dependiendo de los elementos de la cadena.

Al finalizar la primera fase, los conjuntos serán los siguientes: $V_{\mathcal{A}} = \emptyset$, $\mathbf{Q} = \emptyset$, $V_T = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}\}$ y $E_T = \{v_0 - v_1, v_1 - v_2, v_2 \overset{0}{-} v_3, v_3 \overset{1}{-} v_4, v_4 \overset{4}{-} v_5, v_5 \overset{2}{-} v_6, v_6 \overset{3}{-} v_7, v_7 \overset{3}{-} v_8, v_6 \overset{1}{-} v_9, v_9 \overset{0}{-} v_{10}, v_{10} \overset{0}{-} v_{11}, v_{11} \overset{2}{-} v_{12}\}$.

La aplicación del algoritmo de la segunda fase inicia con una lista $\mathbf{L}$ de las aristas del árbol $\mathbf{T}$ recorridas mediante BFS. $\mathbf{L} = \{v_0 - v_1, v_1 - v_2, v_2 \overset{0}{-} v_3, v_3 \overset{1}{-} v_4, v_4 \overset{4}{-} v_5, v_5 \overset{2}{-} v_6, v_6 \overset{1}{-} v_9, v_6 \overset{3}{-} v_7, v_9 \overset{0}{-} v_{10}, v_7 \overset{3}{-} v_8, v_{10} \overset{0}{-} v_{11}, v_{11} \overset{2}{-} v_{12}\}$. La lista $\mathbf{LI}$ es la lista $\mathbf{L}$ ordenada en sentido inverso. Se extrae el primer elemento de $\mathbf{LI}$: $v_{11} \overset{2}{-} v_{12}$, puesto que v_{11} solo tiene un sucesor, entonces $a_{11} = 2$. La siguiente arista es $v_{10} \overset{0}{-} v_{11}$, por consiguiente, $a_{10} = 0 + a_{11} = 02$. Extrayendo la tercera arista se obtiene $a_7 = 3$. Aplicando repetidamente el procedimiento se obtiene $a_9 = 002$. v_6 tiene dos sucesores, por lo tanto, $a_6 = (1002)(33)$. Al final la cadena a que represente a $\mathbf{T}$ será $a = a_2 = 0142(1002)(33)$.

Cabe señalar que no es necesario ordenar los vértices del árbol tal como aparecen en la Figura 4.2, este ordenamiento únicamente se hizo para efectos del ejemplo. Para utilizar ambos algoritmos solo se requiere de un identificador único para cada vértice, que en principio pueden ser las coordenadas del mismo vértice.

En el Algoritmo 1 se ha dado por sentado que siempre es posible obtener las dos primeras aristas ortogonales que definen la manija inicial. Lo anterior se debe a que en caso de un árbol 3D que por sus características no permita la creación de la manija inicial, se puede hacer uso de una manija externa que se adhiera a una hoja del árbol para iniciar el cálculo de su cadena.

4.3. Árboles envolventes

Los sólidos voxelizados pueden representarse con una gráfica cuyas aristas unen los vértices que forman el sólido, del cual se extrae un árbol mediante el procedimiento presentado en la sección anterior. El AE se calcula a partir de un vértice inicial y dos vértices más que determinan las dos direcciones iniciales que forman la manija del descriptor. Una vez escogida la manija, las direcciones se van descubriendo mediante la función (3.1). El procedimiento para generar el árbol envolvente de un sólido se describe en el Algoritmo 3 [BGM12].

Algoritmo 3 Algoritmo para calcular el árbol envolvente de un sólido.

Entrada: La gráfica $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ de la superficie $\mathcal{S}$ del sólido voxelizado.
Salida: Una cadena a que describe un árbol envolvente de $\mathcal{S}$.

1: Seleccionar un vértice arbitrario $v_0 \in V_{\mathcal{S}}$ del sólido como el origen del árbol envolvente.
2: Seleccionar dos vértices $v_1 \in V_{\mathcal{S}} \cap N_6(v_0)$ y $v_2 \in V_{\mathcal{S}} \cap N_6(v_1)$ tales que $v_0 - v_1$ y $v_1 - v_2$ sean aristas ortogonales en $\mathcal{S}$.
3: Mediante el algoritmo 1, obtener la gráfica $\mathbf{T}_{\mathcal{S}}$ con las etiquetas correspondientes a $E_{\mathcal{S}}$.
4: A partir de $\mathbf{T}_{\mathcal{S}}$ y el algoritmo 2, calcular la cadena a.

4.4. Ejemplos de árboles envolventes

A continuación se muestran algunos ejemplos de AEs.

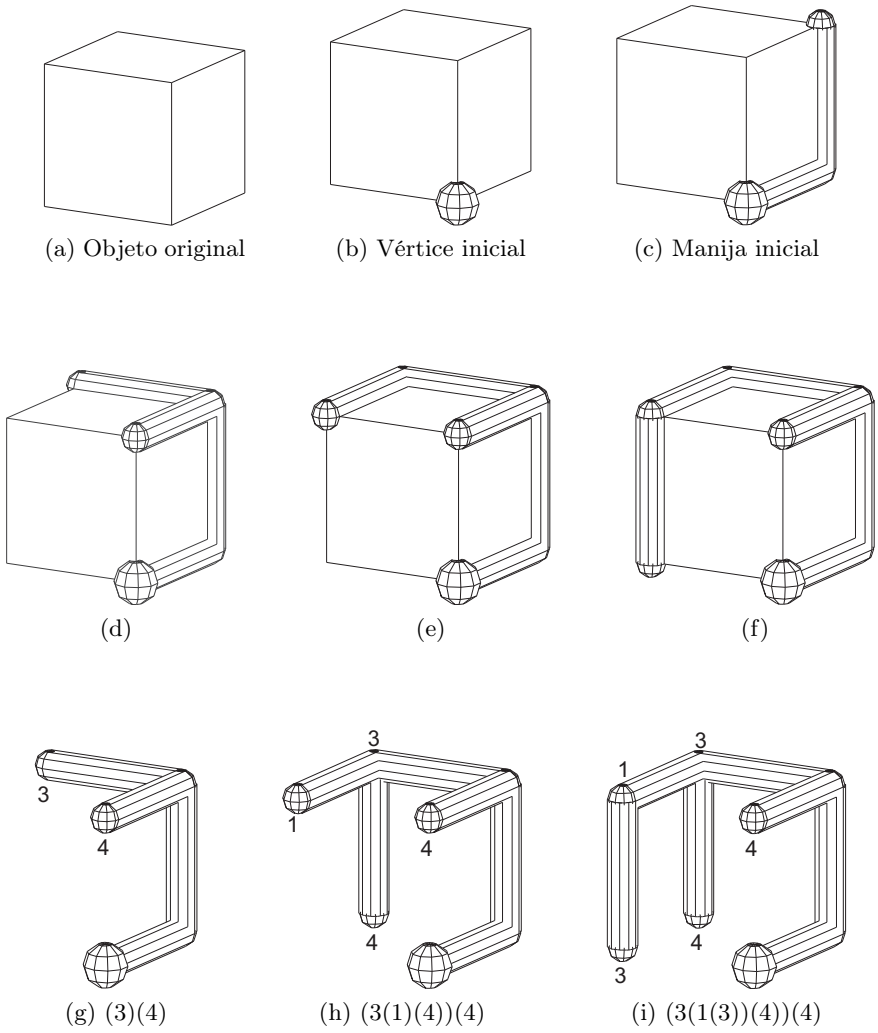
1. En la Figura 4.3 se muestra la secuencia en la que se calcula el AE de un voxel que es el sólido más sencillo. Una vez determinada la manija, los dos primeros vértices que se encuentran corresponden a las direcciones (3)(4). Posteriormente se van encontrando los demás vértices que forman la superficie del voxel hasta recorrerlos todos y obtener la cadena final (3(1(3))(4))(4).

2. En la Figura 4.4 se muestran un sólido y su correspondiente AE. La cadena del AE es

(0(0(0(3(0(0(0(0(1(0)))(1(0(0(3(2)))(3))))(1(0(0(0(3))(3)))))(1(0(0(0(0)(3))))))(1(0(0(0(0))))))(4(0(0(0(
0))))))(3(0(0(0(0(1(0(0)))(2))))))(4(0(0(0(0(3(0(0))))))))(3(0(0(0(0(1(0(0(0(4))(4)))))))))(4(0(0(0(0(3(
0(0(0)(4)))))))))(3(0(0(0(0(1(0(0(0(0(3))(3)(4))(3))(3))(3))(4))(4(1(0(0(0(0)))))))(4(1(0(0(0(0)))))))(4
(1(0(0(0(0)))))))(4(1(0(0(0(0)))))))(4(0(0(0(0(3(0(0(0))))(4))(4))(4))(4)))

3. En la Figura 4.5 se muestra una palanca voxelizada, su correspondiente AE es el siguiente:

(0(0(0(0(0(0(3(0(0(0(0(0(0(0(0(0(1(0(0(0(0(0(0(0(0(0)(3))(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(1(1(0
(1(2(0(2(2(2(2(2(0(2(2(0(0(0(0(0(0(0(0(4(2(2(0(4(2))(2(0(1(0(0(0(0(1(0(0(0(0(0(
0(1(0(0(0(0(2(0(0(0(0(0(0))))))))(2(0(0(0(0(0(0(0(1))(1)))))))))(2(0(0(0(0(0(0(0(1(0))(2)))))))))(2(0(0(
0(0(0(0(0(1(0(0))))(1(0(0(0))))))))))))(2(0(0(0(0(0(0(1(0(0(0(0)))))))(1(0(0(0(0(0(0)))))))(1(0(0(0(0(0(0
)))))))(1(0(0(0(0(0(0)))))))(1(0(0(0(0(0(0)))))))(1(0(0(0(0(0(0)))))))(3(0(0(0(
0(0(0)))))))(4(0(0(0(0(0(0)))))))(1(0(0(0(0)))))(4(0(0(0(0(0(0)))))))(1(0(0(0(0)))))(1(0(0(0)))))(1(0
(0(0)))))(1(0(0))))(1(0))))(1(1(0(2(2(2(2(2(0(2(2(0(0(0(0(0(0(0(0(0)))))))))))))))))))(1(1(0(2(2(2(2(0(2

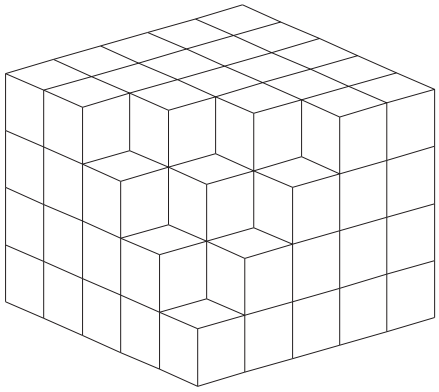


(a) Objeto original (b) Vértice inicial (c) Manija inicial

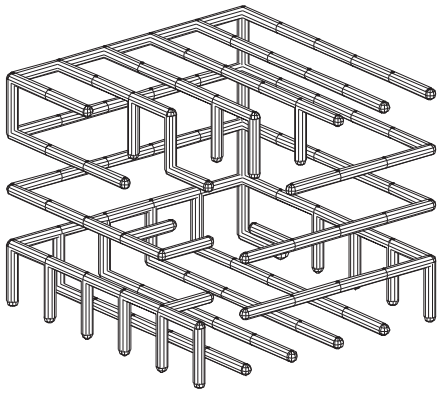
(d) (e) (f)

(g) (3)(4) (h) (3(1)(4))(4) (i) (3(1(3))(4))(4)

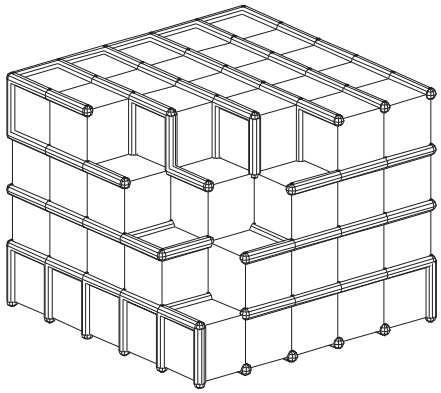
Figura 4.3: Secuencia para el cálculo del AE de un voxel.



(a) Sólido original.

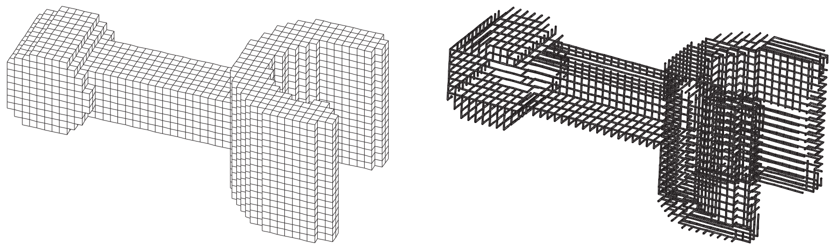


(b) AE calculado a partir de todos los vértices del sólido.



(c) Sólido con el AE sobrepuesto.

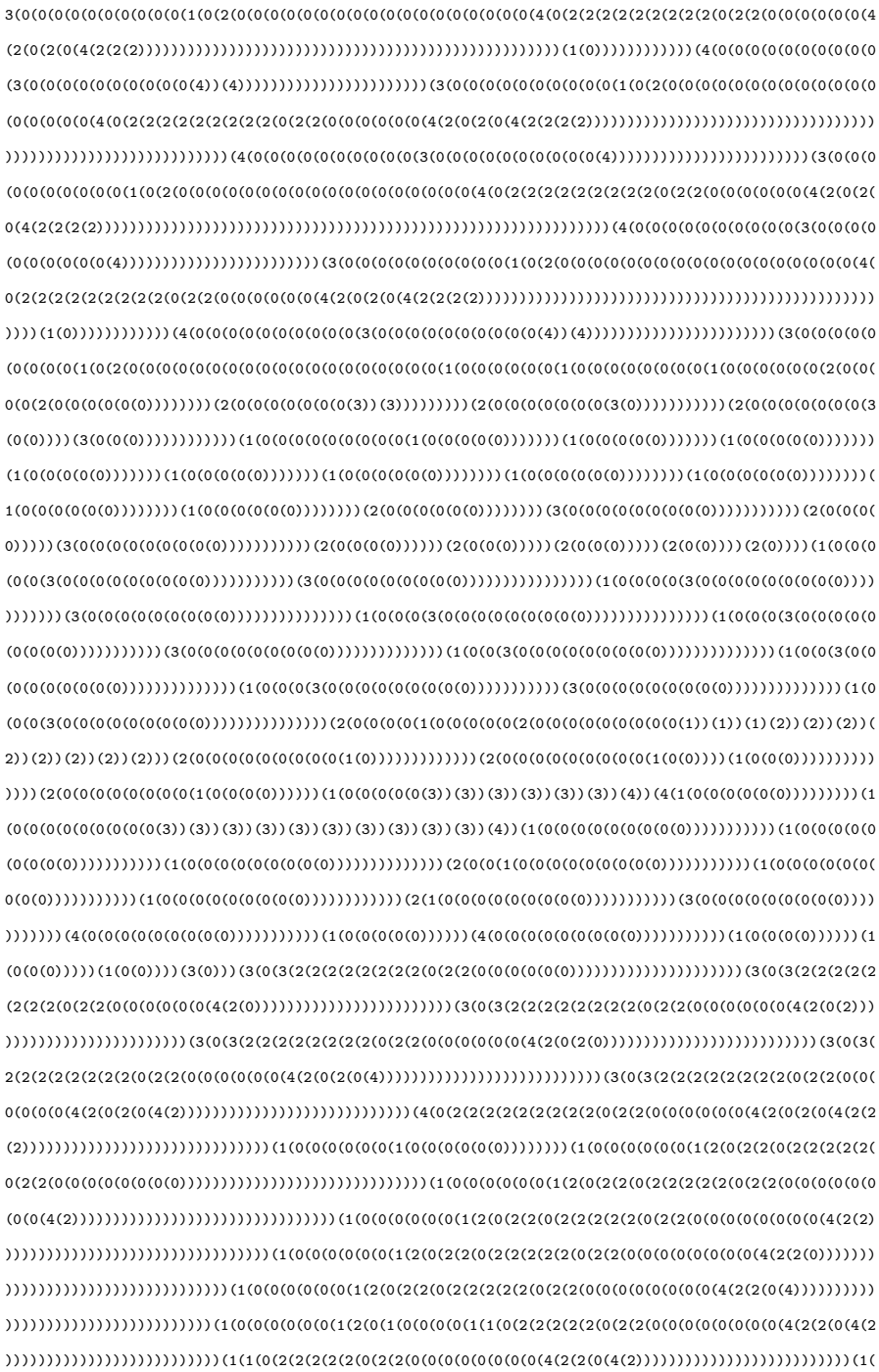
Figura 4.4: Ejemplo de un sólido y su correspondiente AE.



(a) Sólido original. (b) Árbol envolvente.

Figura 4.5: Ejemplo de una palanca voxelizada y su AE.

(2(0(0(0(0(0(0(0(0(4(2))))))))))))))))))))))(1(1(0(2(2(2(2(2(0(2(2(0(0(0(0(0(0(0(0(4(2(2)))))))))))))))))
))))))(1(1(0(2(2(2(2(2(0(2(2(0(0(0(0(0(0(0(0(4(2(2(0)))))))))))))))))))))))))(2(2(0(2(2(2(2(2(0(2(2(0(0(0(
0(0(0(0(0(4(2(2(0(4)))))))))))))))))))))))))))(1(0(0(0(0(1(0(0(0(0(0(4(0(0(0(0(0(0)))))))))(4(0(0(0(0(0(0)))
)))))))))))))))(3(0(0(0(0(2(0(0(0(0(4(0(0(0(0(0(0))))))))(4(0(0(0(0(0(0)))))))))))))))))(1)(3(0(0(0(0)))
))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(3(0)))(2(0(0(0(0(0(0(0(0(0(0
(0(0(0(0(0(0(0(3(0(0(0(0(2(0(0(0(0(4(0(0(0(0(0(0))(2(0(0(0(0(0(0(0(0(0
(0(0(0(0(0(0(0(0(0(0(3(0(0(0(0(2(0(0(0(0(4(0(0(0(0(0(0))))))))(4(0(0(0(0(0(0))))))))))))))))))))))))))))))
)))))(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(3(0(0(0(0(2(0(0(0(4(0(0(0(0(0(0))))))))))))))))))))))))))))))
))))))(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(3(0(0(0(0(2(0(0(0(4(0(0(0(0(0(0))))))))))))))))))))))))))))))
)))))))(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(3(0(0(0(0(2(0(0(0(0(4(0(0(0(0(0(0))))))))(4(0(0(0(0(0(0))))
))))))))))))))))))))))))))))))(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(3(0(0(0(0(2(0(0(0(0(4(0(0(0(0(0(0)
))))))))))))))))))))))))))))))))))(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(3(0(0(0(0(1(0(1(0(0(0(0(0(0(
0(2(0(0(0(0(2(0(0(0(0))))))(2(0(0(0(0(0(0(3))(3))(3)))))))(2(0(0(0(0(0(0(3(0))))))))))(2(0(0(0(0(0(0(3(0(
0))))(3(0(0(0))))))))))))(1(0(0(0(0(0(0(1(0(0(3(2))))))(1(0(0(3(2(0)))))))(1(0(0(3(2(0)))))))(1(0(0(3(2(0)
))))))(1(0(0(3(2(0)))))))(1(0(0(3(2(0)))))))(1(0(0(3(2(0)))))))(2(0(0(0(0(0(3(0(0(0(0))))))(3(0(0(0(0(0))
))))(4))(4))(4)))))(3(0(0(0(0(0(0(0))))))))(2(0(0(0(0)))))(3(0(0(0(0(0(0(0))))))))(2(0(0(0(0)))))(3(0(0(0(0(0
(0))))))))(2(0(0(0)))))(2(0(0(0)))))(2(0(0))))(2(0)))(2)))(1(0(0(0(0(0(3(0(0(0(0(0(0(0))))))))(3(0(0(0(0(0(
0)))))))))))))(2(0(0(0(0(4(0(0(0(0(0(0))))))))(4(0(0(0(0(0(0))))))))))))(1(0(1(2(2(2(2(2(2(2(2(0(2(2(0(0(
0(0(0(0))))))))))))))))))))(1(0(1(2(2(2(2(2(2(2(2(0(2(2(0(0(0(0(0(0(4(2(0))))))))))))))))))))))))))(1(0(1(
2(2(2(2(2(2(2(0(2(2(0(0(0(0(0(0(4(2(0(2)))))))))))))))))))))))))))(1(0(1(2(2(2(2(2(2(2(0(2(2(0(0(0(0(0(
0(4(2(0(2(0))))))))))))))))))))))))))))(4(0(2(2(2(2(2(2(2(2(2(0(2(2(0(0(0(0(0(0(4(2(0(2(0(4))))))))))))))))
))))))))))(3(0(0(0(0))))))))))))))))))))))))))))))(1(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1)
)(1))(1))(2))(2(0(3(0(0(0(0(0(0(0(0(0(0)))))))))))(3(0(0(0(0(0(0(0(0(0(0)))))))))))))(2(0(3(0(0(0(0(0(0(
0(0(0(0))))))))))))))(2(0(3(0(0(0(0(0(0(0(0(0(0)))))))))))))))(2(0(3(0(0(0(0(0(0(0(0(0(0))))))))))))))(2(0
(3(0(0(0(0(0(0(0(0(0(0))))))))))))))(2(0(3(0(0(0(0(0(0(0(0(0(0)))))))))))))(3(0(0(0(0(0(0(0(0(0(0(3(0(0(0)
)))))))))))))))(2(3(0(0(0(0(0(0(0(0(0(0)))))))))))))(1(0(0(0(0(0(0(0(0(0(0(1)(2(0(0(0(0(0(0))))))))(1))(
1))(1))(1))(1))(1))(1))(1))(1))(1))(2))(4(0(0(0(0(0(0(0(0(0(0)))))))))))))(3(0(0(0(0(0(0(0(0(0(1(0(2(0(0(0
(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(4(0(2(2(2(2(2(2(2(2(2(0(2(2(0(0(0(0(0(0(4(2(0(2(0(4(2))))))))))))))))))
))(4(0(0(0(0(0(0(0(0(0(0(3(0(0(0(0(0(0(0(0(0)))))))))))))))))))))(



```
1(0(2(2(2(2(2(0(2(2(0(0(0(0(0(0(0(0(4(2(2(0(4(2)))))))))))))))))))))))(1(1(0(2(2(2(2(2(0(2(2(0(0(0(0(0
(0(0(0(4(2(2(0(4(2))))))))))))))))))))))))(1(1(0(2(2(2(2(2(0(2(2(0(0(0(0(0(0(0(0(4(2(2(0(4(2)))))))))
))))))))))))))(2(2(0(2(2(2(2(2(0(2(2(0(0(0(0(0(0(0(0(4(2(2(0(4(2)))))))))))))))))))))))))(1(0(0(0(0)))
))(3(0(0(0(0))))))(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0))))))))))))))(1(1(0(0(0(0(0(0(4(0(0(
0(0))))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0
(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0))))))
)))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4
(0(0(0(0))))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0))))))))))))))(1(
1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0))
)))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(0(0(4(0(0(0(0)))))))))))))(1(1(0(0(0(0(
0(0(4(0(0(0(0))))))))))))))(3(2(0(0(0(0(0(0(0(0)(4))(4(0))))))))))))(3))(4))(4(0(1(0(0(0(0(0(0(0(0(0(0))))
))))))))(1(0(0(0(0(0(0(0(0(0(0))))))))))))(4(0(0(1(0(0(0(0(0(0(0(0(0(0)))))))))))))(1(0(0(0(0(0(0(0(0(0
)))))))))))))(4(0(0(1(0(0(0(0(0(0(0(0(0(0)))))))))))))))(4(0(0(1(0(0(0(0(0(0(0(0(0(0)))))))))))))))(4(0(
0(1(0(0(0(0(0(0(0(0(0(0)))))))))))))))(4(0(0(1(0(0(0(0(0(0(0(0(0(0)))))))))))))))(4(0(0(1(0(0(0(0(0(0(0
(0(0)))))))))))))))(4(0(0(1(0(0(0(0(0(0(0(0(0(0))))))))))))(1(0(0(0(0(0(0(0(0(0(0(1(0(0(0(0))))))))))))))
)))))(4(0(1(0(0(0(0(0(0(0(0(0(0))))))))))))(1(0(0(0(0(0(0(0(0(0(0(1(0(0(0(0(0(0)))))))))))))))))))(4(0(0
(0(0(0(0(0(0(0(0(3(0(0(0(0(0(0(0(0(0))))))))))(4))(4))(4))(4))(4))(4))(4))(4))(4))(4)))
```

4. En la Figura 4.6 se muestra un busto de Ludwig van Beethoven, su versión voxelizada y el AE. Por razones de espacio se omite la cadena.

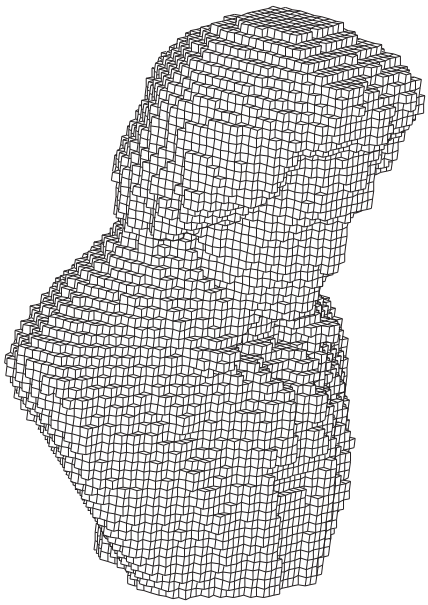
4.5. Árboles envolventes en el modelo reticular de puntos

El procedimiento descrito en las secciones 4.2 y 4.3 que se aplicó en el contexto del modelo reticular de células, únicamente hace uso de la lista de vértices que constituyen la superficie del sólido. Por consiguiente, el mismo procedimiento puede ser utilizado para construir un AE en el modelo reticular de puntos si se cambia el conjunto de vértices por un conjunto de baricentos que forman un sólido. La Figura 4.7 muestra un AE calculado para un sólido a través de sus baricentros. En las Figuras 4.7(b) y (d) se muestran los AEs correspondientes a un sólido de $2 \times 2 \times 2$ voxeles que es recorrido por sus baricentros y el que es recorrido por sus vértices, respectivamente.

En la Figura 4.8 se muestra el AE correspondiente al modelo de puntos del ejemplo 2 de la sección 4.4 (Figura 4.4), su código de cadenas es

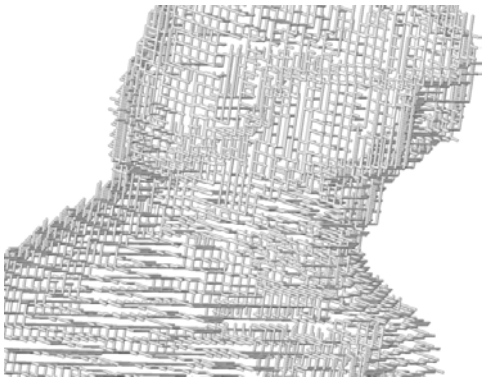
```
(0(0(3(0(0(0(1))(1(0)))(1(0(0))))(1(0(0(0)))))(4(0(0(0)))))(3(0(0(0(1(0))))))(4(0(0(0(3(0)))))))(3(0(0(0(
1(0(0(3(2)))(3))(3))(4))(4(1(0(0(0))))))(4(1(0(0(0))))))(4(1(0(0(0))))))(4(0(0(0(3(0(0)))(4))(4))(4)))
```

(a) Busto de L. Beethoven.



(b) Versión voxelizada del busto.

(c) AE del busto.



(d) Acercamiento sobre el AE del busto.

Figura 4.6: Ejemplo de un AE calculado para un busto de Ludwig van Beethoven.

Una observación muy evidente, es que el AE obtenido en el modelo de puntos siempre será más corto que el correspondiente al modelo de células, ya que recorre un número menor de baricentros que de vértices.

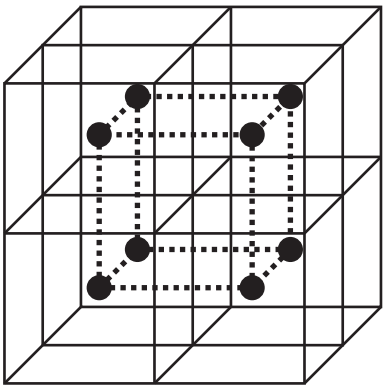
4.6. Propiedades de los árboles envolventes

Algunas de las propiedades de los AEs son las siguientes [BGM12]:

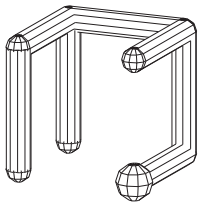
- *Los AEs sin invariantes bajo rotaciones.* Como se mencionó en el capítulo anterior los cambios de dirección dependen del sólido mismo. Por consiguiente el AE es invariante a rotaciones.
- *Facilidad en el cálculo de la imagen especular.* La imagen especular de un AE se puede obtener fácilmente intercambiando las direcciones 1 y 3, al igual que en el caso de las curvas 3D.
- Si a es la cadena de un AE y m el número de vértices de la superficie de un sólido, entonces $n = m - 3$, donde n es el número de elementos de la cadena.
- Sean un sólido S en el contexto del modelo reticular de células y n el número de caras expuestas de los voxeles que forman la superficie envolvente de S. Cada cara puede dar origen a 8 manijas, 4 en un sentido como se muestra en la Figura 4.9 y 4 en el sentido contrario. El número total de AEs es de $8n$ para cada sólido ya que toda manija siempre reside en una cara.
- Dado un sólido, la cadena calculada en el modelo reticular de puntos tendrá menos símbolos que la correspondiente al modelo reticular de células porque recorre un número menor de baricentros que de vértices.

4.7. Cadenas de esqueletos

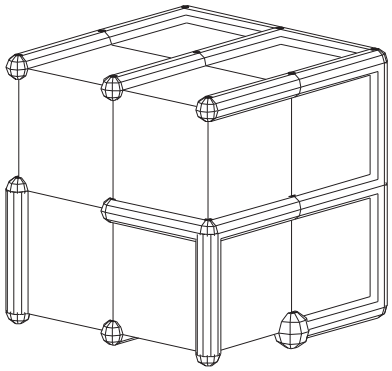
En general, los algoritmos de esqueletización producen objetos k-conexos, ($k = 0, 1, 2$); esto es, que los voxeles pueden tener 2-células (caras), 1-células (aristas) o 0-células (vértices) comunes, susceptibles de ser representados mediante el código 5OT en alguno de los dos modelos reticulares. En el caso del modelo de puntos, existen algoritmos de esqueletización como el publicado por Carso *et al.* para modelar los conductos pulmonares [CEM10] que genera esqueletos 2-conexos en el cual la representación 5OT puede ser muy adecuada porque la cadena es única si se tiene un procedimiento para



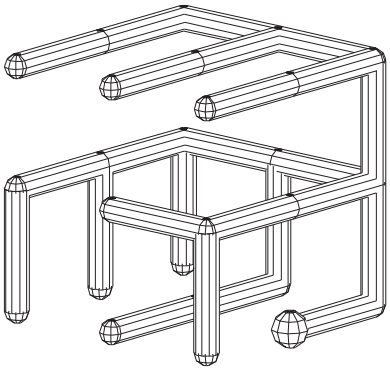
(a) Los baricentros de un sólido unidos por líneas punteadas.



(b) AE correspondiente a los baricentros.

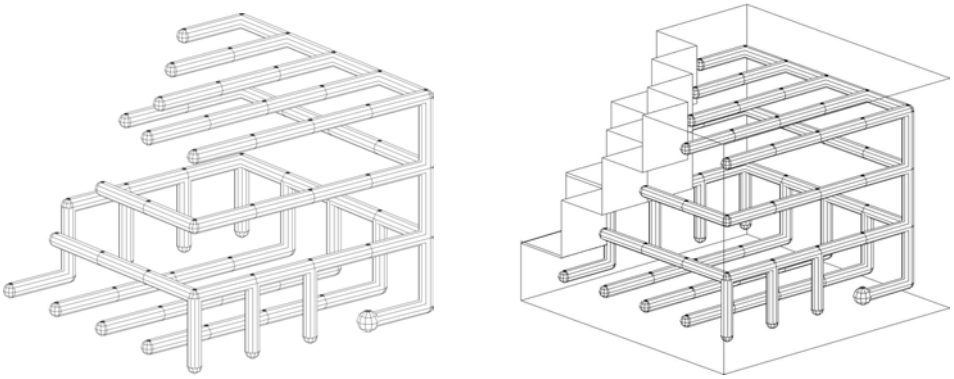


(c) AE correspondiente a un sólido de $2 \times 2 \times 2$.



(d) AE correspondiente a un sólido de $2 \times 2 \times 2$.

Figura 4.7: Comparación de AEs en ambos modelos reticulares.



(a) AE en el modelo de puntos del sólido de la Figura 4.4.

(b) AE colocado en el interior del sólido.

Figura 4.8: Ejemplo de AE en el modelo de puntos para el sólido de la figura 4.4.

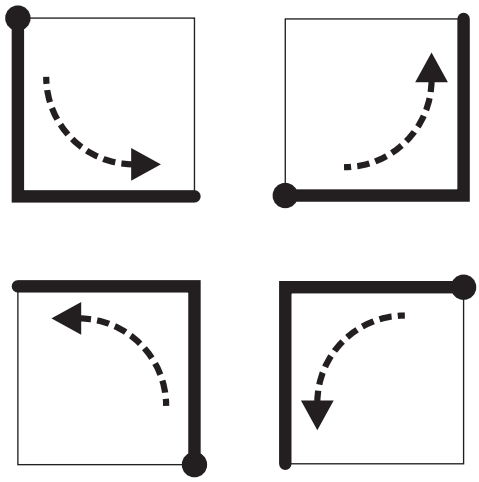
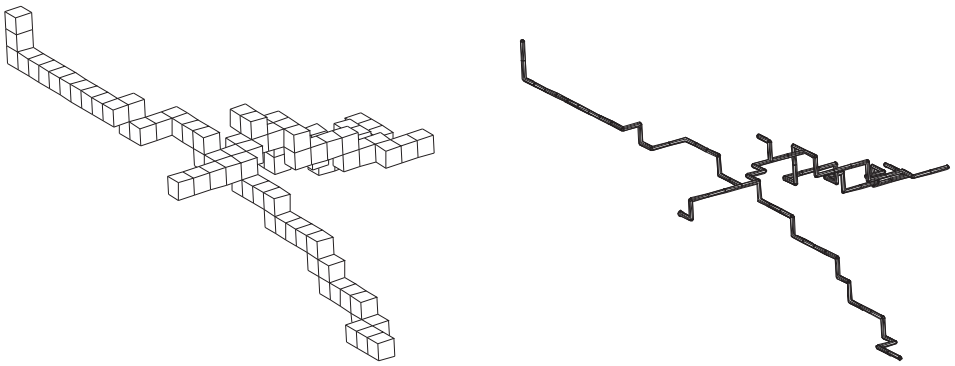


Figura 4.9: Ejemplos de las manijas generadas en una dirección sobre una cara expuesta de un voxel. El vértice inicial de cada manija está marcado con un punto.



(a) Ejemplo de esqueleto 2-conexo. (b) AE calculado.

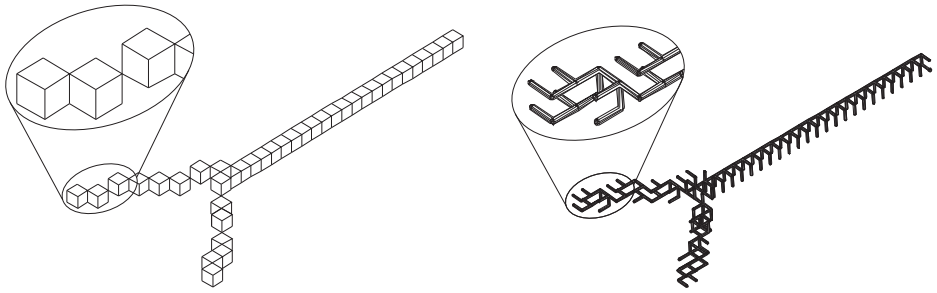
Figura 4.10: Ejemplo de AE para un esqueleto 2-conexo.

establecer el punto inicial a partir del cual se calcule la cadena 5OT. En la Figura 4.10 se muestra el ejemplo de un esqueleto y su correspondiente AE.

Aunque para el uso del código 5OT se ha impuesto la restricción que los voxeles tengan conexidad a través de sus caras (2-células), es importante señalar la posibilidad de utilizarlo en objetos que tengan conexidad por aristas (1-células) o por vértices (0-células), siempre que los voxeles mantengan su alineación paralela a los ejes coordenados. La Figura 4.11 muestra un esqueleto de la palanca presentada en la Figura 4.5 y un AE del esqueleto calculado en el modelo reticular de células. Como puede observarse en el acercamiento de la Figura 4.11(a) algunos voxeles tienen conexidades por aristas y vértices. No es posible calcular el AE directamente sobre el esqueleto debido a que el código 5OT no tiene direcciones para ángulos distintos de 90 grados, pero es factible calcular su AE en el modelo celular como se exhibe en la Figura 4.11(b).

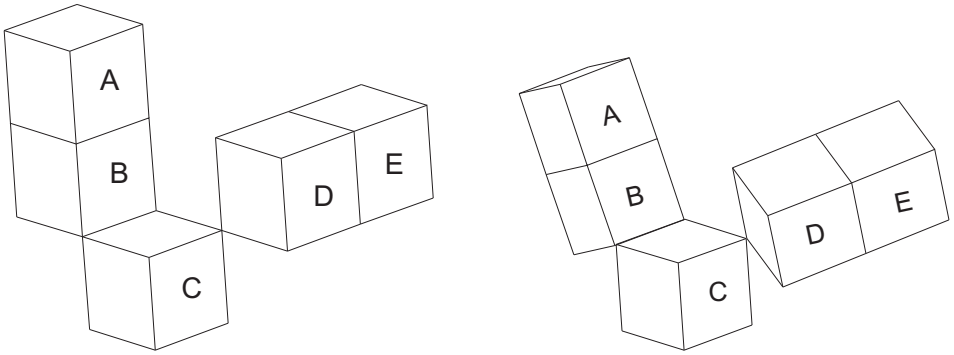
La razón de imponer la restricción de la conexidad por caras en sólidos que utilicen el código 5OT tiene su origen en la posibilidad de que dos voxeles conexos por aristas, no necesariamente mantienen sus caras paralelas a los ejes coordenados. La Figura 4.12(a) presenta un sólido alineado con los ejes coordenados y en (b) se ofrece un ejemplo en que, si bien C está alineado y tiene conexidad por aristas con B, los voxeles A y B no están alineados con los ejes coordenados. El problema análogo para la conexidad por vértices es señalado con los voxeles C, D y E.

De lo anteriormente expuesto, se puede concluir que es viable utilizar el código 5OT para representar esqueletos y medir la similitud entre ellos como



(a) Esqueleto de la palanca de la Figura 4.5. (b) AE del esqueleto.

Figura 4.11: Uso del código 5OT en esqueletos.



(a) Sólido original alineado con los ejes coordenados.

(b) El voxel C está alineado pero los voxeles A, B, D y E no.

Figura 4.12: Origen de la restricción de conectividad por caras en sólidos voxelizados.

se muestra en el capítulo 6.

Capítulo 5

Árboles de bordes (AdBs)

En este capítulo se introducen las nociones básicas acerca de los planos digitales para obtener los árboles de bordes y representarlos con el código 5OT.

5.1. Introducción

Debido a que los AEs visitan todos los vértices que forman la superficie del sólido subyacente, es natural preguntarse si es posible obtener una representación equivalente a los AEs que no requiera de todos los vértices y que preserve la geometría del sólido. Las figuras 2.11(d) y (e) presentan un ejemplo en el que el AE es mucho más complejo que el árbol que traza el borde del sólido.

Los árboles de bordes (AdBs) son una propuesta que bosqueja sólidos voxelizados mediante el recorrido de sus bordes y utiliza el código 5OT para representarlos como un refinamiento de los AEs. La idea básica es obtener un objeto análogo a un esqueleto representado con el código 5OT en el cual los vértices que se utilizan como borde han sido seleccionados adecuadamente. Se proponen dos tipos de AdBs, los árboles de bordes planos (ABPs) y los árboles de bordes generalizados (ABGs).

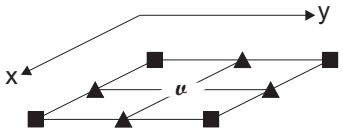
Para obtener los ABPs se omite el trazado por los vértices de las caras planas paralelas a los ejes coordenados de la superficie del sólido. En el caso de los ABGs, éstos se obtienen a partir de las intersecciones de los planos digitales obtenidos mediante un procedimiento de segmentación del sólido en planos digitales [Bu03]. Al igual que los AEs los ABPs se calculan en el modelo reticular de células. Los algoritmos utilizados para calcular los planos digitales que forman un objeto operan en el modelo reticular de

puntos, posteriormente se usan los vértices de los voxeles representados por los baricentros para conseguir los ABGs. Cabe señalar que el procedimiento descrito en la sección 4.3 aplicado a diferentes conjuntos de vértices permite calcular los AdBs. A continuación se describe cada tipo de AdB en particular.

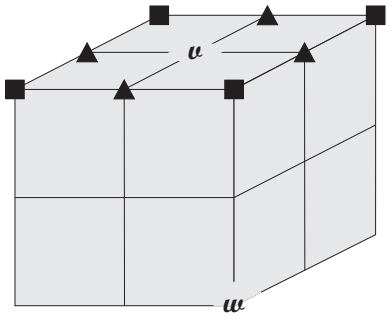
5.2. Árboles de bordes planos

Para calcular un ABP es necesario omitir unos vértices de la superficie del sólido con el procedimiento descrito en el capítulo anterior. Los vértices candidatos a ser excluidos son aquellos que tienen una *vecindad plana*. Dado un vértice v, la *vecindad plana-xy* de v, denotada por $N_{xy}(v)$, es el subconjunto de vértices de $N_{26}(v)$ embebido en un plano paralelo al plano $z = 0$. $N_{xz}(v)$ y $N_{yz}(v)$ son las vecindades planas paralelas a los planos $y = 0$ y $x = 0$, respectivamente (la Figura 5.1(a) es un ejemplo de una vecindad plana). La Figura 5.1(b) muestra una vecindad plana de v sobre un sólido de $2 \times 2 \times 2$. El vértice w no tiene vecindades planas.

Cabe notar que si un vértice tiene una vecindad plana entonces ésta pertenece a una cara de la superficie del sólido voxelizado como se muestra en la Figura 5.1(b). Los vértices que no tienen vecindades planas como w son los que constituyen el borde *plano* del sólido. Dado un sólido voxelizado interesa detectar únicamente los vértices que no tienen vecindades planas para poder aplicarles el procedimiento que traza el borde del objeto.



(a) Vecindad plana de u.



(b) Vecindad plana de w sobre un objeto de $2 \times 2 \times 2$ voxeles. El vértice w no tiene vecindades planas.

Figura 5.1: Ejemplo de una vecindad plana para un vértices v sobre la superficie de un sólido.

Previo a la aplicación del procedimiento descrito en la sección 4.3, es

necesario borrar de la lista de vértices de la superficie del sólido aquellos vértices que tengan vecindades planas $(N_{xy}(v), N_{xz}(v), N_{yz}(v))$.

Ejemplos de árboles de bordes planos

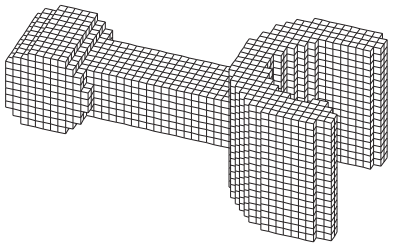
En la Figura 5.2 se muestran tres ejemplos de ABPs calculados para objetos manufacturados. La cadena de la Figura 5.2(b) es la que viene a continuación y tiene 4665 caracteres. En este caso la reducción de la cadena es de casi la mitad, ya que la cadena de la Figura 4.5 del capítulo anterior tiene 9642 caracteres.

```
(0(0(0(0(0(0(3(1(0(0(0(0(0(0(0(0(0(0(1(1(0(2(2(0(0(0(0(4(2(2(2(2(0(2(2(0(0(4(2))))))))))))))))))))))(1))(
1))(1))(1))(1))(1))(1))(1))(1))(1))(2(2(0(1(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(2(2
(0(0(0(0(1(0(0(0(0(0(0(0(0(0(3(3(2(2)))))(3(3(2(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(2(0(0(0(0(0(0(0
(0(0(0(0(0(1(0(1(2(0(1(0(0(0(0(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(2(2(2(2(2
(0(2(1(0(0(0(0)))))(2(0(0(0(0(0(0(0(0(3(0(0(0(0(1(1)))(1(1)))(1(1)))(1(1)))(1(1)))(4(2(2(0(3(0(0(0(0(1(1)
))(1(1)))(1(1)))(1(1)))(1(1)))(4(2(2(0(0(0(0(0(0(2(1(0(0(0(0(1(1(2(2(2))))))(1(1(2(2(2))))))(1(1(2(2(2)))
)))(1(1(2(2(2))))))(1(1(2(2(2))))))(2(2(2(2(2(2(0(1(0(0(0(0))))))))))))(3(0(0(0(0)))))))))))))))))))))
)))))(3(0(0(0(0)))))))))))))(3(0(0(0(0))))))(4(0(0(0(0)))))))(4))(4))(4))(4))(4))(4))(4))))))))(4(0(0(0(0
(3(0(3(1(0(0(0(0(0(0(0(0(0(0(0)))))))))))))(2(0(2(1(0(0(0(0(0(0(0(0(0(0(0(1(1(2))))(1(1(2))))(1(1(2))))(1(
1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(2(2(2(2(0(1(0(0(0
(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(2(2(0(0(0(0(0(0(0(0(1(0(0(0(0(0(0(0(0(0(
0(0(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(4(2(2(0(1(0(0(0(0
(0(0(0(0(0(0(0(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(3(3)))(4(2(2(
0(0(0(0(0(0(1(0(0(0(0(0(0(0(0(0(0(0(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)
))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1
(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(2(2(2(2(2(2(2(0(3(0(0(0(0(0(0(0(0(0(0(0(0)))))))))))
))))))))))))))))))))))))))))))))))))(3(0(0(0(0(0(0(0(0(0(0(0))))))))))))))(4(0(0(0(0(0(0(0(0(0(0(0)))))
))))))))))))))(4(0(0(0(0)))))))))))))))))))))))))(2(0(1(1))(2(2(0(0(1(3))(4(2(0(2)(3)(4(0(0(0(0(0(0(0(0(0
(0(0(0(0(0(0(0(0))))))))))))))))))))))))))))))))))(3(3(2))))(3(3(2))))(3(3(2))))(3(3(2))))(3(3(2))))(3(3(2)
)))(3(3(2(2(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(2(0(0(0(0(0(0(0(0(0(0(0(0(3(0(3(1(0(0(0(0(1(1(2(2(2(2
(2))))))))(1(1(2(2(2(2(2))))))))(1(1(2(2(2(2(2))))))))(1(1(2(2(2(2(2))))))))(1(1(2(2(2(2(2))))))))(2(2(2(
2(2(2(2(2(0(1(0(0(0(0(1))(1))(1))(1))(1))(2(2(0(0(0(0(0(0(1(0(0(0(0(3))(3))(3))(3))(3))(4(2(0(2(0(1(0(0(0
(0(3(3(2(2)))))(3(3(2(2)))))(3(3(2(2)))))(3(3(2(2)))))(3(3(2(2)))))(4(2(2(2(2(0(1(0(0(0(0(3))(3))(3))(3))
(3))(4(2(0(0(3(0(0(0(0(1))(1))(1))(1))(1))(4(2(0(1(0(0(0(0(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2
)))))(1(1(2(2)))))(2(2(2(2(2(0(2(1(0(0(0(0)))))(2(0(0(3(0(0(0(0)))))))))(3(0(0(0(0)))))))))))))))))))))
)))(3(0(0(0(0)))))))))))))))))))))))))))(4(0(0(0(0)))))))(4(1(0(0(0(0(0))))))))(4))(4))(4))(4))(4))(4(1(0(
0(0(0(0)))))))))))))))(4(0(0(0(0(1(0(1(2(1(0(0(0(0(0(0(0(0(0(0(0(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1
(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2))
)))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(1(1(2(2(2(2)))))))(2(2(2(2(2(2(2(0(2(1(0(0
```

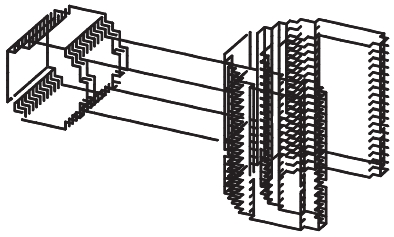
```
(0(0(0(0(0(0(0(0(0)))))))))))(2(0(0(0(0(0(0(3(0(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(
1))(1))(1))(4(2(0(1(0(0(0(0(0(0(0(0(0(0(0)))))))))))))(2(0(3(0(0(0(0(0(0(0(0(0(0(0(1(1(2(2)))))(1(1(2(2)))
))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2))))
)(1(1(2(2)))))(1(1(2(2)))))(4(2(2(2(2(0(3(0(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1
))(1))(4(2(0(0(1(0(0(0(0(0(0(0(0(0(0(0(3))(3))(3))(3))(3))(3))(3))(3))(3))(3))(3))(3))(4(2(0(2(1(0(0(0(0(
0(0(0(0(0(0(0(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1
(1(2))))(1(1(2))))(1(1(2))))(2(2(2(2(0(1(0(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1)
)(1))(2(2(0(0(1(0(0(0(0(0(0(0(0(0(0(0))))))))))))))))))))))(3(0(0(0(0(0(0(0(0(0(0(0))))))))))))))))))))
)))))))))))))))(3(0(0(0(0(0(0(0(0(0(0(0))))))))))))))))))))))(3(0(0(0(0(0(0(0(0(0(0(0))))))))))))(4(0(0(
0(0(0(0(0(0(0(0(0))))))))))))))))))))(3(0(0(0(0(0))))))(4(0(0(0(0(3(0(0(0(0(0))))))))))))))))))))))))))
))(2(0(2(2(0(0(4(2(0(2(1(0(0(0(0(0(1))(1(1)))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(2(2(2))))(4(0(0(0(0
(0(0(0(0(0(0(0(0(0(0(0(0(0)))))))))))))))))))))))))))))))))))(3(3(2(2)))))(4(2(2(2(2(0(1(1))(2(2(0(0(1(3))(4
(2(0(2(1(1(1(2))))(2(2(2(2(0(0(0(0(0)))))))))))(3))))))))))))))))))))))(4(0(0(0(0(0(0(0(0(0(0(4(0(0(0(0(
0(0(3(1(0(0(0(0(0(0(0(0(0(0(1(1)))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(1(1(2))))(
1(1(2))))(1(1(2))))(1(1(2))))(2(2(2(2(0(0(0(0(0(1(0(0(0)(3))(3(3)))(3(3(2))))(4(2(2(2)))))))))))))(4(0(0
(0(0(0(0(0(0))))))))))))))))))))))))))))))
```


Un comentario acerca de las vecindades planas

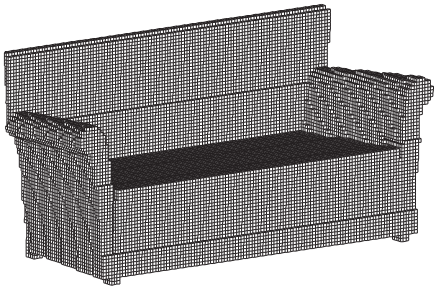
En la primera fase del procedimiento para el cálculo de un AdB se eliminan los vértices que están en las caras planas del sólido. Al hacer esta operación la superficie del objeto puede fragmentarse en varios componentes conexas que evitan el trazado de toda la superficie. La Figura 5.3(a) muestra el objeto original y 5.3(b) presenta la superficie una vez que han sido eliminados los vértices con vecindades planas. Si se inicia el proceso de trazado del AdB sobre la componente C, es claro que no se podrá acceder a la componente D. Dado que el objeto original es conexo, es posible encontrar dos vértices uno en C y otro en D con una distancia mínima entre ellos. Una vez encontrados estos vértices, en el objeto original se puede trazar un camino entre ellos. Este camino puede ser utilizado para continuar el trazado en la segunda componente. En la Figura 5.3(b) el camino es la línea punteada entre los puntos. El vértice en C puede considerarse un nuevo nodo *raíz* análogo al vértice inicial. La Figura 5.4 muestra un objeto en el cual se ha trazado este camino y por consiguiente, el procedimiento ha podido trazar el borde de todo el objeto.



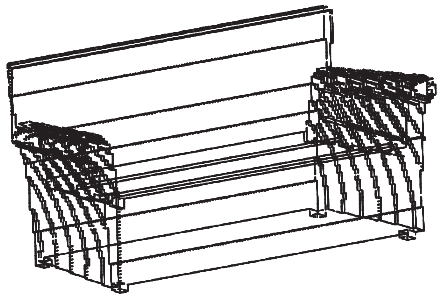
(a) Sólido original.



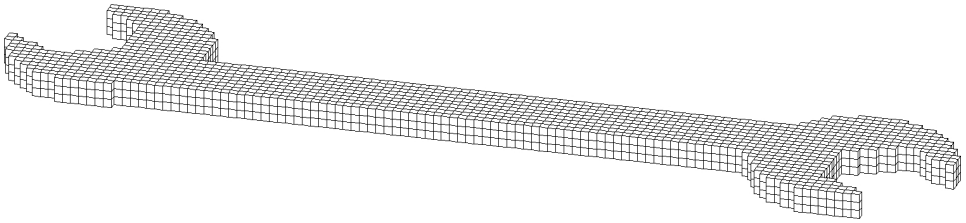
(b) Árbol de bordes planos.



(c) Sólido original



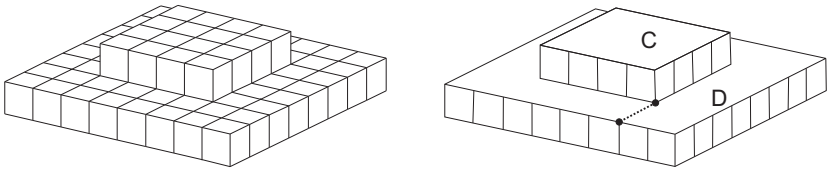
(d) Árbol de bordes planos



(e) Objeto original

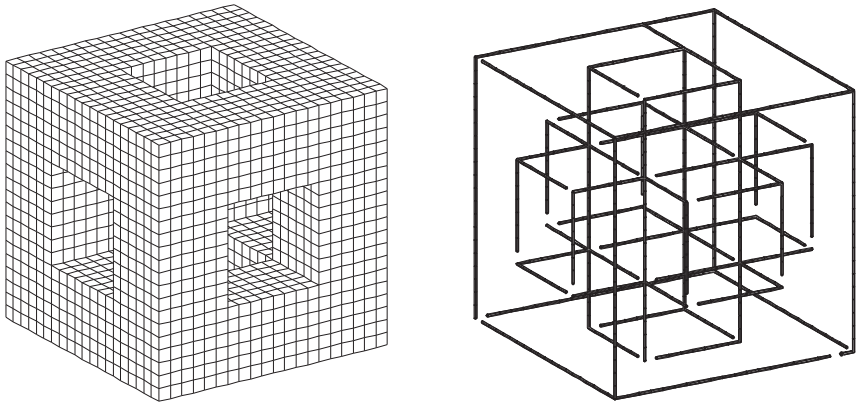
(f) Árbol de bordes planos

Figura 5.2: Ejemplos de ABPs de objetos manufacturados.



(a) (b)

Figura 5.3: El camino que conecta los componentes C y D.



(a) Objeto original. (b) AdB calculado.

Figura 5.4: El camino que conecta los componentes de un objeto.

5.3. Árboles de bordes generalizados

En geometría digital, la descomposición de un objeto discreto en piezas o segmentos de planos discretos ha sido un tema relevante de estudio ya que está relacionado con problemas de reconstrucción a partir de poliedros [SDC05, ABB09]. Los AdBs recorren las intersecciones de los planos digitales que conforman la superficie del sólido y se representa por medio del código 5OT. Al igual que en el caso de los AEs, los AdBs son árboles con grádo máximo de seis. El método propuesto para representar sólidos voxelizados como descriptor 5OT unidimensional preserva la geometría del objeto y permite una reducción en la longitud de la cadena.

Generalmente la segmentación de la superficie de un objeto discreto en piezas de planos discretos es un primer paso en procedimientos con objetivos de mayor envergadura como la poliedrización de superficies. La detección o reconocimiento de los planos que conforman una superficie ha sido un problema muy estudiado en geometría digital y consiste en determinar cuándo un conjunto discreto de puntos es parte de un plano digital o no [Bu03, KR04, SDC04, SDC05]. En este trabajo se utiliza el algoritmo publicado por E. Charrier en 2008 [CB08]. Más adelante se dan las razones de la selección de este algoritmo.

Definiciones básicas

Esta sección introduce las definiciones básicas de geometría digital que serán utilizadas en secciones subsiguientes. Las rectas digitales se presentan como concepto preliminar para definir los planos digitales.

En el espacio euclidiano bidimensional $\mathbb{R}^2$ el conjunto de rectas que pasan por el origen pueden ser descritas por medio de la ecuacion $ax + by = 0$ donde a y b son primos relativos. Cuando al conjunto de rectas se le suma un parámetro γ de traslación, cada línea divide al plano en dos regiones: $\gamma < ax + by$ y $ax + by < \gamma$.

Definición 5.1. Una *recta digital* es el conjunto $D(a, b, \gamma, \rho)$ de puntos definido como $D(a, b, \gamma, \rho) = \{(x, y) \in \mathbb{Z}^2 \mid \gamma \leq ax+by < \gamma+\rho\}$; donde (a, b) es el vector normal a la recta, γ el parámetro de traslación y ρ el espesor aritmético.

Intuitivamente puede decirse que una recta digital es el conjunto de pixeles (o puntos) comprendido entre dos rectas euclidianas. La Figura 5.5 muestra dos conjuntos de puntos en $\mathbb{Z}^2$ correspondientes a dos rectas digitales cuya base es la recta $x-3y = -3$: $D(1, -3, -3, 3)$ y $D(1, -3, -3, 8)$. La recta base

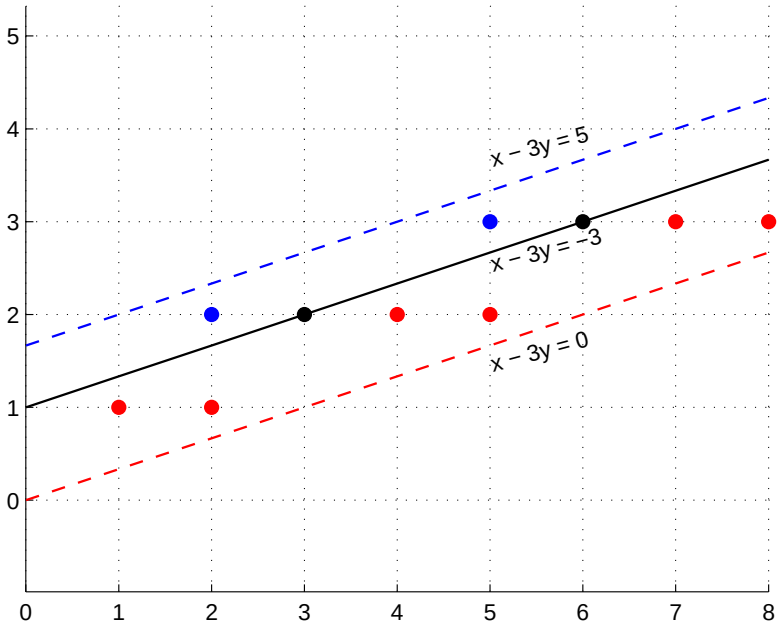


Figura 5.5: Gráficas de las rectas digitales $D(1, -3, -3, 3)$ y $D(1, -3, -3, 8)$.

es $x-3y=-3$ ya que $\gamma=-3$. La recta digital $D(1,-3,-3,3)$ corresponde a los puntos entre las recta base y $x-3y=0$. La recta digital $D(1,-3,-3,8)$ es el conjunto de puntos localizados entre las rectas base y $x-3y=5$. La línea $x-3y=-3$ y los puntos sobre ella son comunes a ambas rectas digitales.

Un subconjunto importante de rectas digitales son aquellas denominadas *rectas digitales simples*[1] en las cuales $\rho = \text{máx}(|a|,|b|)$. Estas líneas se caracterizan porque los conjuntos de puntos son 8-conexos y por el hecho de que para cada abcisa entera solo existe un punto en la banda definida por la recta digital. En la Figura 5.5, la recta $D(1,-3,-3,3)$ es una recta digital simple, ya que puede observarse que para cada abcisa, solo existe un punto contenido mayor o igual que $x-3y=0$. Por el contrario, la recta $D(1,-3,-3,8)$ es una recta digital *más delgada* que en algunas abcisas no tiene un punto como en $x=4$. Por consiguiente, puede afirmarse que el parámetro ρ controla la conexidad de la recta digital.

El efecto de ρ sobre la conexidad de la recta puede dividirse en cuatro casos: a) cuando $\rho < \text{máx}(|a|,|b|)$ la recta es disconexa, b) si $\rho = \text{máx}(|a|,|b|)$ la recta es estrictamente 8-conexa, c) si $\rho = |a|+|b|$ la recta es estrictamente 4-conexa; y d) si $\rho > |a|+|b|$ la recta puede considerarse como gruesa porque simplemente admite más de un punto en algunas abcisas.

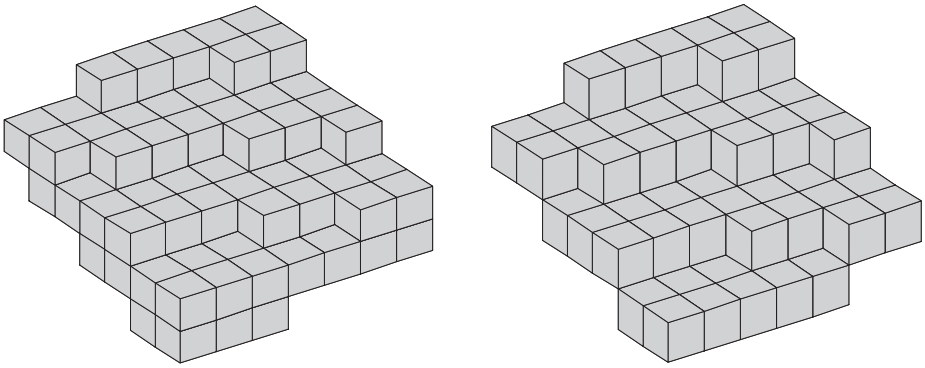
Las definiciones previas sobre rectas digitales pueden extenderse a planos digitales en el espacio tridimensional.

Definición 5.2.

- Un *plano digital* con vector normal (a,b,c), parámetro de traslación γ y grosor aritmético ρ es el conjunto de puntos $(x,y,z) \in \mathbb{Z}^3$ definido como $D(a,b,c,\gamma,\rho) = \{(x,y,z) \in \mathbb{Z}^3 \mid \gamma \leq ax+by+cz < \gamma+\rho\}$. Donde a,b,c no son todos cero y son primos relativos.
- Un *plano digital estándar* es aquel en el cual $\rho = |a|+|b|+|c|$ y *simple* cuando $\rho = \text{máx}(|a|,|b|,|c|)$.

La Figura 5.6 muestra dos ejemplos de planos digitales. En (a) se tiene el plano estándar $D(6,15,35,1,56)$ y en (b) el plano simple $D(6,15,35,1,35)$. De manera similar a lo que sucede en el espacio bidimensional, los planos simples son los planos *más delgados* que mantienen la 18-conexidad sin que presenten 6-hoyos como se puede observar en (b) [SDC05]. Las propiedades relativas a la relación mencionadas previamente entre el grosor aritmético y la conexidad en dos dimensiones aplican también al espacio tridimensional.

[1] *naive digital lines* en inglés



(a) Plano estándar. (b) Plano simple.

Figura 5.6: El plano discreto $\gamma \leq 6x + 15y + 35z < \gamma + \rho$ con $\gamma = 1$ y dos grosores: $\rho = 56, 35$.

Descomposición de sólidos en planos digitales

El reconocimiento de planos digitales simples ha sido un problema muy estudiado en geometría digital. Consiste en determinar cuándo un conjunto de n puntos en $\mathbb{Z}^3$ es una parte de un plano simple o no. Los métodos de reconocimiento pueden dividirse en tres grandes grupos: a) los basados en programación lineal, b) los que utilizan conjuntos convexos y métodos geométricos, y c) los que usan métodos de optimización combinatoria. En este trabajo se utilizó el algoritmo COBA desarrollado por E. Charrier y L. Buzer que pertenece al tercer grupo, debido a que está programado en la biblioteca DGtal [DGtal] y porque es óptimo en la determinación de si un conjunto finito de puntos en $\mathbb{Z}^3$ pertenece a un plano digital [CB08]. Este algoritmo transforma el problema de reconocimiento de los planos digitales en un problema de programación lineal relacionado con la definición de los planos simples [Bu03].

El algoritmo COBA reconoce planos simples del tipo $\mu \leq ax+by+z < \mu+\epsilon$, donde los vectores normales son del tipo $(a, b, 1)$, $\mu = \gamma/\rho$ y $\epsilon = 1$. La idea es transformar la búsqueda de los planos en un problema de programación lineal en el plano $\mathbb{Z}^2$. Dado un conjunto de puntos $p_i = (x_i, y_i), i = 1, \ldots, n$ y considerando al eje principal de búsqueda como z, se pretende encontrar un conjunto de vectores (a, b) que cumplan con las restricciones $\mu = ax_i + by_i$ y $ax_i + by_i < \mu + \epsilon$.

Dada una descomposición de un sólido en planos digitales, los voxeles contiguos de dos planos distintos comparten un subconjunto de vértices.

El *árbol de bordes* es aquel que se obtiene trazando la intersección de la descomposición de los planos digitales mediante el código de cadenas 5OT. En la Figura 5.7 los planos digitales tienen diferentes tonos de gris y la intersección está indicada por la línea gruesa.

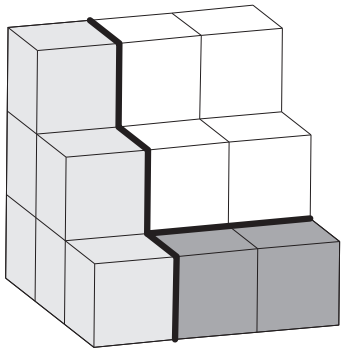


Figura 5.7: El árbol de bordes traza la intersección de la descomposición en planos digitales de un sólido.

Ejemplos de árboles de bordes generalizados

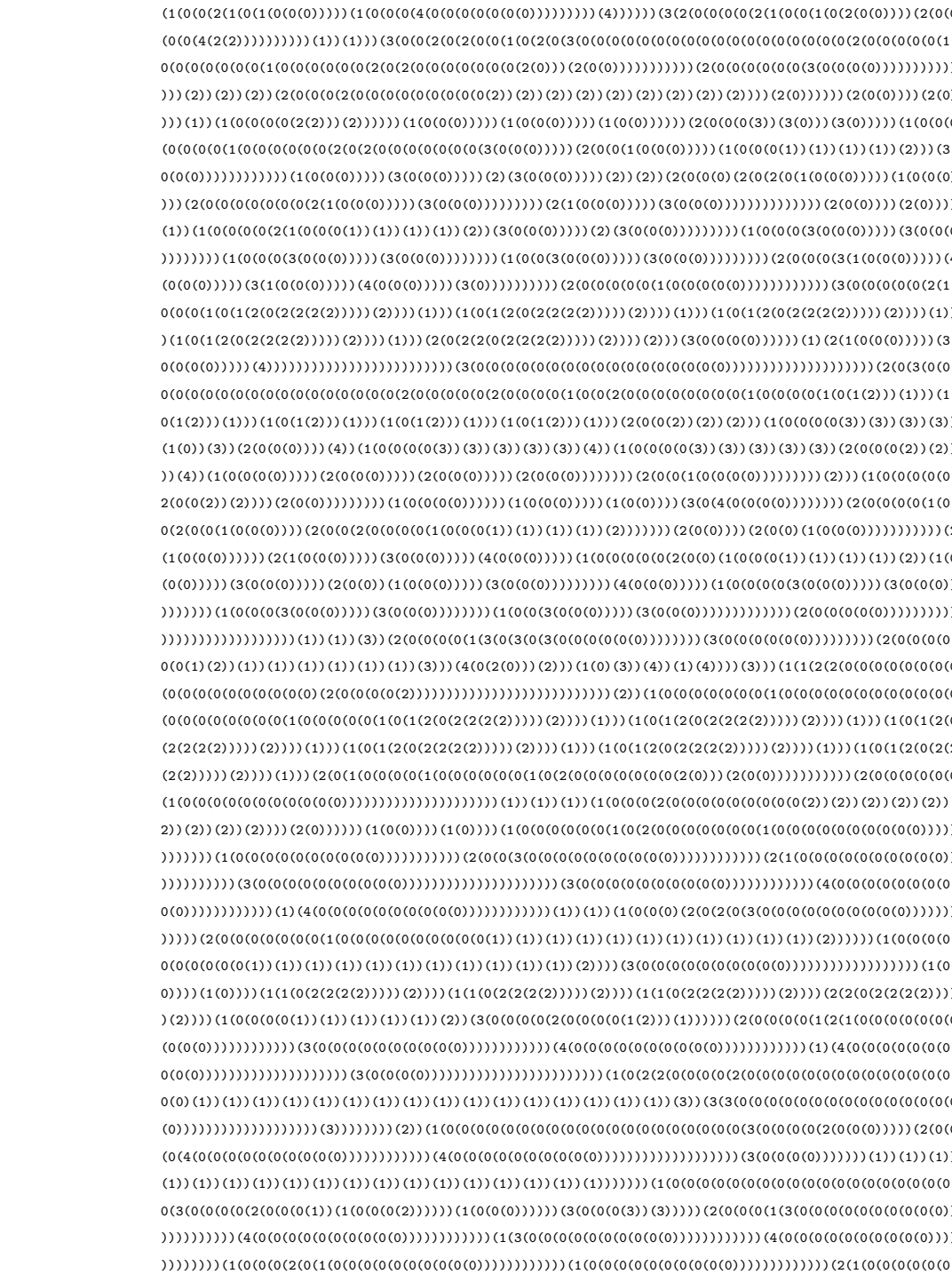
En la Figura 5.8 se muestran dos ejemplos sencillos de AdBs. En (a) se muestra un cubo rotado 45 respecto a dos planos coordenados. La Figura 5.8(b) muestra el AdB de un cubo. El código de la Figura 5.8(b) es el siguiente:

(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(1(2(1(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2(0(2(1(0(0(0(0(0(0(0(0(
0(0(0(0(0(0)))))))))))))))))(2))))))))))))))))))))))))(3(0(0(0(0(0(0(0(0(0(0(0(0(0(0(3(2(2(2(2(2(2(2(2(2(2(2
(2(2(2(2(2(2(0(2))(2))))))))))))))))))))))(4))))))))))))))))))))))))))))))))(4(0(2(2(2(2(2(2(2(2(2(2(
2(2(2(2(2(2(2(1(0(0(0(0(0(0(0(0(0(0(0(2(1(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2))))))))))))))))))))))))))))))))))
))))))))(3(2(3(2(2(2(2(2(2(2(2(2(2(2(2(2))(3(0(0(0(0(0(0(0
(0(0(0(0(0(1(3))(2(1(2(2(2(2(2(2(2(2(2(2(2(2(2(2(2))))))))))))))))))))))))))(4(0(3(2(2(2(2(2(2(2(2(2(2(2
(2(2(2))))))))))))))))))))))))))))))))))).

El código de la Figura 5.8(d) es el siguiente:

(0(0(0(0(0(0(0(0(0(0(1(0(0(0(0(0(0(0(0(0(3(0(0(0(0(0(0(0(0(0(0(4(1(0(0(0(0(0(0(0(0))))))))))))))))))))))))(4(0(0
(0(0(0(0(0(0(0(0(3(0(0(0(0(0(0(0(0(0)(1(0))))))))))))(1(0(0(0(0(0(0))))))))))))))))))))))))))(3(1(0(0(0(0(0
(0(0(0(0(0(0(4(0(0(0(0(0(0(0))))))))(3(0(0(0(0(0(0(0(0(0(0(1(0(0(0(0(0(0(0(0))))))))(1(0(0(0(0))))))))))))))))
))))))))))))))))))))))).

En la Figura 5.9 se muestra el AdB generalizado de la Figura 4.5, su código es el siguiente:



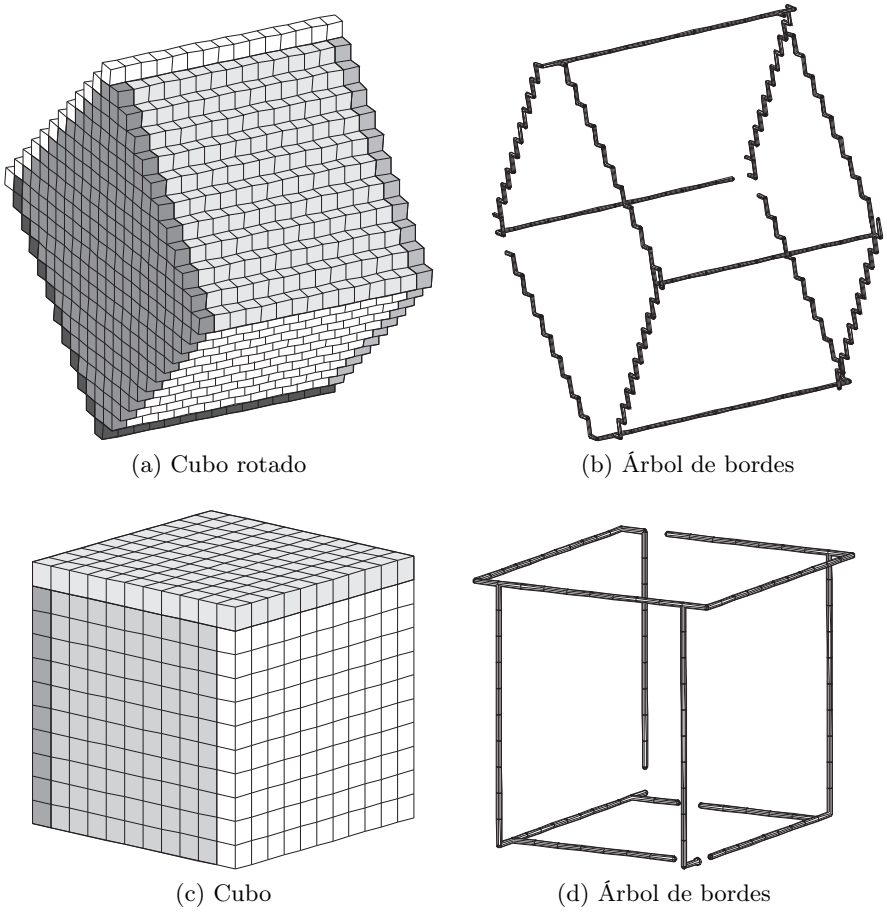
```
0(0(0(0))))))))))))(3(0(0(0(0(0(0(0(0(0(0))))))))))))))))(3(0(0(0(3))(3)))))))))(2(0(0(0))))(3(0(0(0(0(3(0
(0(0))))))))))(2(0(0(0))))))))))))))))))))))))))(1(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(0(1(0(0(0(0(0))))))
(3(0(0(0(0(1(0(1(0(0(0(0(0(0(0(2(0(0(0(0(2(0(0(0(0))))))(2(0(0(0(0(0(0(3(0(0(0(0(0(0(0(0(0(0(0))))))))))))
))))))))(2(0(0))))(2(0(0))))(2(0(0(0(0(0(2(0(1(0(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1
))(1))(1))(2))(1(0(0(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(2))(3(0(0(0(0(0(0(0
(0(0(0(0))))))))))))))(2(0(1))(1))(3(0(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(
4))(3(0(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(4))(3(0(0(0(0(0(0(0(0(0(0(0(1)
)(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(4))(3(0(0(0(0(0(0(0(0(0(0(0))))))))))))))))(3(0(0(0(0(0(0(0(0
(0(0(0))))))))))))))(2))(2))(2))(2(0(0(0(2(0(0(2))(2))))(2(0(0)))))))(2(0(0))))(2(0))(2))(4(0(0(0(0(0(0(0(
0(0(0(0))))))))))))))(1(0(0(0(0(0)))))))(2(0(0(0(0))))))(2(0(0(0(0(3(0(0(2(0(0(2(0(0(2(0(0(0(0(2(1(0(0(0(0(
0(0(0(0(0(0))))))))))))(3(0(0(0(0(0(0(0(0(0(0))))))))))))))))(2(0(0))))(2(0(0)(3(0(0(0(0(0(0(0(0(0(0)))))
))))))))(3(0(0(0(0(0(0(0(0(0(0))))))))))))))(2(3(0(0(0(0(0(0(0(0(0(0))))))))))))(1(0(0(0(0(0(0(0(0(0(0(
1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(2))(4(0(0(0(0(0(0(0(0(0(0))))))))))))(3(0(0(0(0(0(1(0(0(0(0(0(
0(0(0(0(0(1(0(1))))(1(0(1))))(1(0(1))))(1(0(1))))(1(0(1))))(1(0(1))))(1(0(1))))(1(0(1))))(1(0(1))))(1(0(1)
)))(1(0(1))))(2(0(0)(2))))(1(0(0(0(0(0(0(0(0(0(0(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(1))(2(0(0)))))))
(4(0(0(0(0(0(0(0(0(0(0))))))))))))(3(0(0(0(0(1(0(0(0(0(0(0(0(0(0(0))))))))))))(1(0(0(0(0(0(0(0(0(0(0))))))
))))))))(3(0(0(0(1(0(0(0(0(0(0(0(0(0(0))))))))))))(1(0(0(0(0(0(0(0(0(0(0))))))))))))))(3(0(0(1(0(0(0(0(
0(0(0(0(0(0))))))))))))(1(0(0(0(0(0(0(0(0(0(0))))))))))))))))))(3(0(0(0(0)))))))))))))))))))))))))(2))(4
(0(0(0(0(0(0(0))))))))))(2(1))(3(1))(4(0(0(0(0(0(0(0(3))))))))))(4))))))(1(0(0(0(0(0(0(0(0(1(1(0(0(0(0)))))
))(1(1(0(0(0(0)(3))))))(2(0(0(0(0(0(1(0(0(1(1(0(1(0(0))))(1(0(0(0(0(4(2(2))))))))))(2(0(2(0(0))))(2(0(0(0(
0(3)(4(0(2(0)))(2)))(3(0)(1))(4))(3)(4))))))(1(1))(2))(1(1))(2))(2))(1(1))(2))(4)))))(3))(1(1))(2)(3))(1))
(1))(1))(1))(1))(1))(2(1(0)))(3(0)(1))(4))(1(0(0(0(0(0(0(0(0)(3))(3))(3))(3))(3))(3))(3))(3))(3(0)(1))(4))
(2(0(0(0(0(0(0(0(0(4(1(2(0(0(0(0(0(0(1(0(3(0(3(0(2))(2)))(3)))(3(0(3(0(2(0(0(0(0(1)))))))(2(0(0(0(0(1)))))
)))(3)))(4(2(2(2(2(0(0(0(0(0))))))))))))(1(0(3))(3))))))))))(4(0(1))(1)))(4(0(1))(1)))(4(1)))(4(1)))(4(1))
)(4(1)))(4(1)))(4(1(3(0(3(1)))(3(1))))))(4(0(4(3(2(0(0(0(0(0(0(3(0(0(0(0(0(0(0(1(1(2(2)))))(1(1(2(2)))))(1
(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(1(2(2)))))(1(0(1(0(2))(2)))(1)))(1(0(1(0(2(0(0(0(0(3)))))))(2(0(0(
0(0(3))))))))(1)))(4(2(2(2(2(0(0(0(0(0)))))))))))(3(0(1))(1))))))))))))
```

5.4. Cálculo de los árboles de bordes

Para calcular los árboles de bordes puede utilizarse el algoritmo 3 solo que aplicado a conjuntos distintos de vértices. En el caso de los ABPs, el conjunto de vértices está constituido por aquellos que no tienen vencindades planas. Para los árboles generalizados el conjunto de vértices será el obtenido de la intersección de una descomposición en planos digitales.

5.5. Propiedades de los árboles de bordes

Los AdBs comparten la mayoría de las características que los AEs, sin embargo la diferencia más importante es la reducción en la longitud del descriptor.



(a) Cubo rotado (b) Árbol de bordes

(c) Cubo (d) Árbol de bordes

Figura 5.8: Ejemplos de árboles de bordes.

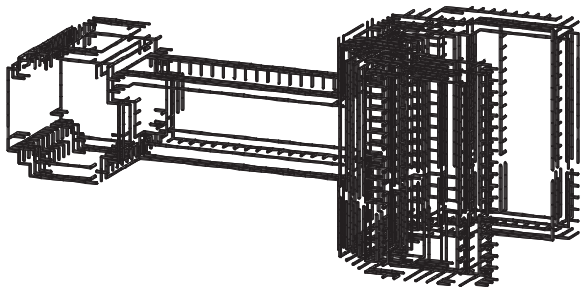


Figura 5.9: Árbol de bordes generalizado de la palanca mostrada en la Figura 4.5.

- *Los AEs son invariantes bajo rotaciones.* Como se mencionó en el capítulo 3 los cambios de dirección dependen del sólido mismo. Por consiguiente el AdB es invariante a rotaciones.

- *Facilidad en el cálculo de la imagen especular.* La imagen especular de un AdB se puede obtener fácilmente intercambiando las direcciones 1 y 3, al igual que en el caso de las curvas 3D.

- Por el hecho de utilizar menos vértices para trazar el objeto, los descriptores de los AdBs son más cortos que los AEs. En particular, los objetos manufacturados que contengan caras planas como el mostrado en la Figura 5.2(f) se ven más favorecidos en su representación mediante un AdB que por un AE.

Capítulo 6

Análisis de árboles envolventes y árboles de bordes

Dado un esquema de representación, un aspecto que surge inmediatamente es el que se refiere a la capacidad de los descriptores para facilitar el análisis de objetos y cuantificar las similitudes así como las diferencias entre ellos. En este capítulo se aborda el problema del análisis de los AEs y AdBs, y se proponen algunas métricas que pueden ser utilizadas para comparar objetos voxelizados.

6.1. Introducción

Dados dos sólidos A y B representados por las cadenas a y b, respectivamente, existen diversas maneras de *comparar* A y B a través de a y b de manera análoga como ocurre con los contornos de figuras bidimensionales [IBS96] o curvas tridimensionales [AB15]. En particular, los descriptores 5OT tienen características que plantean dos alternativas principales para la comparación de códigos de cadenas. Por una parte, los descriptores pueden ser considerados como secuencias de caracteres y analizados mediante técnicas de comparación de cadenas[1]. Por otro lado, puesto que el descriptor es un árbol, es posible aprovechar elementos de la teoría de gráficas para establecer una comparación entre A y B. Como opciones de métricas de similitud para la comparación de AEs y AdBs, en este capítulo se exploran las distancias de edición entre cadenas y entre árboles. Respecto a la teoría de gráficas, se utiliza la teoría espectral para comparar descriptores de sólidos voxelizados.

Cabe recordar que en términos generales una medida de similitud es

[1] *string matching* en inglés

una función que asocia un valor numérico a un conjunto de elementos tal que será mayor cuando los elementos sean más parecidos. D. Lin [DL98] propone una lista de tres características, que denomina intuitivas, que debe cumplir toda medida de similitud. Dados dos objetos A y B que interesa comparar, a) la similitud entre A y B está relacionada con las cualidades que tienen en común, entre más cualidades compartan más similares serán; b) la similitud entre A y B está relacionada con sus diferencias, entre más diferencias tengan, su similitud será menor; c) la máxima similitud se alcanza cuando A y B son idénticos. Una medida de disimilitud opera de manera inversa a la similitud, entre más pequeño sea el valor asociado a un conjunto de elementos, más similares serán.

Debido a que frecuentemente los objetos 3D se obtienen de procesos de adquisición de imágenes o experimentos sujetos a errores de medición, en lugar de evaluar exactamente las cadenas que representan a los sólidos es común determinar de manera aproximada su coincidencia o similitud. Para lograr lo anterior se han desarrollado una gran diversidad de métodos de comparación entre cadenas y la selección de uno en particular depende de los objetivos específicos de las aplicaciones. Por ejemplo, en ciertos casos puede interesar una comparación global de las cadenas, mientras que en otras ocasiones se busca establecer si un patrón de referencia está incluido como subcadena en diferentes regiones de otra más larga. En áreas como biología molecular se han desarrollado métodos que se denominan de alineamiento en los cuales, dadas dos cadenas se introducen espacios en blanco en las cadenas buscando una mayor coincidencia [BB07].

En términos generales los métodos para comparar cadenas de caracteres pueden dividirse en tres grandes grupos: los de análisis global en los cuales se hace una comparación utilizando la totalidad de las cadenas, los de análisis local que son aquellos en los que se busca determinar si una subcadena se repite dentro de otra cadena mayor y los métodos híbridos que combinan el análisis global y el local [BB07]. Existe una métrica híbrida de disimilitud para curvas 3D propuesta por E. Bribiesca y W. Aguilar [BA06] desarrollada para el código 5OT. Dadas dos curvas por comparar, esta métrica cuantifica las secciones de ambas curvas que son comunes y las secciones que no lo son. Cabe señalar que esta métrica únicamente se aplica a curvas y no a árboles.

Para hacer una evaluación inicial sobre cadenas se escogieron dos métricas globales: la distancia de edición de cadenas de Levenshtein [BB07] y la distancia de edición de árboles conocida como RTED [PA11]. Para comparar los árboles como gráficas se utilizó la teoría espectral de gráficas [WZ08]. Algunas de las razones principales para la selección de estas métricas son las

siguientes: la distancia de Levenshtein se utilizó porque es la métrica más sencilla que ofrece cuantificar las diferencias globales entre dos cadenas mediante operaciones de inserción, eliminación y sustitución de caracteres. RTED hace una búsqueda de estrategias basadas en trayectorias y presenta una alternativa de evaluación distinta a la distancia de Levenshtein. En teoría de gráficas el espectro se utiliza comúnmente para caracterizar las propiedades de una gráfica y extraer información acerca de su estructura [CRS10].

Cabe señalar que para abordar el problema de la comparación y clasificación de sólidos, es necesario explorar un tema importante relacionado con la orientación de los objetos a analizar. Se requiere obtener un método adecuado para orientar los objetos de manera que los AdBs tengan puntos iniciales con ciertas propiedades comunes a todos los sólidos. Para efectos del presente trabajo y debido a la complejidad del problema de la orientación, se considerará que los objetos utilizados y sus respectivas cadenas están orientados adecuadamente. Ejemplos de métodos para orientar objetos son los ejes principales [JL93] o los momentos tridimensionales [GC93].

En lo que resta del capítulo, se presentarán los aspectos básicos de las métricas de distancia de edición propuestas, así como la teoría espectral de gráficas. Como ejemplos del uso de las métricas se utilizarán dos conjuntos de objetos: uno de sólidos y un conjunto de modelos digitalizados de percheros utilizado por E. Bribiesca [EB08] para hacer una evaluación empírica de similitud con el código 5OT.

6.2. Distancia de edición en cadenas

Dadas dos cadenas o secuencias de caracteres, la manera más sencilla de evaluar su similitud es mediante la *distancia de Levenshtein* [VL66] también conocida como *distancia de edición*. La distancia de Levenshtein es una métrica que permite hacer una comparación global de ambas cadenas fácil de calcular y aunque el algoritmo presentado más adelante no es muy eficiente tiene a su favor el ser muy flexible para adaptarse a diversas aplicaciones. Esta métrica ha sido la base sobre la cual se han desarrollado otras técnicas para diversos usos como la revisión ortográfica de textos, la búsqueda de secuencias de ADN, la detección de plagios en textos, etc.

La distancia de Levenshtein $d_L(a, b)$ entre dos cadenas $a = a_1a_2 \dots a_m$ y $b = b_1b_2 \dots b_n$ de longitudes m y n, es el menor costo de una secuencia de inserciones, eliminaciones y sustituciones de los caracteres necesarios para transformar a en b. El costo de una secuencia de operaciones es la suma del costo individual de cada operación.

El siguiente algoritmo para calcular $d_L(a, b)$ hace uso de técnicas de programación dinámica y aunque en el pasado fue descubierto por varios autores, P. Sellers fue quien primero lo aplicó a la búsqueda de cadenas y al cálculo de la distancia en secuencias de caracteres [GN01]. El algoritmo trata de encontrar una manera eficiente de transformar la cadena de inicio en la cadena destino mediante la búsqueda de todos los cambios posibles a través de las operaciones de inserción, eliminación y sustitución. Se asume que si una operación transforma una subcadena $a_1 \dots a_i$ en $b_1 b_2 \dots b_j$, la primera no puede volver a modificarse.

Para calcular $d_L(a, b)$, se inicia creando una matriz $M_{0..m \times 0..n}$ donde el valor de $M_{i,j}$ se calcula de acuerdo con la siguiente función:

$$M_{i,j} = \begin{cases} i & \text{si } j = 0, \\ j & \text{si } i = 0, \\ M_{i-1,j-1} & \text{si } a_i = b_j, \\ \min \begin{cases} M_{i-1,j-1} + c_s \\ M_{i-1,j} + c_e \\ M_{i,j-1} + c_i \end{cases} & \text{si } a_i \neq b_j, \end{cases} \tag{6.1}$$

donde c_s, c_e y c_i son los costos de las operaciones de sustitución, eliminación e inserción, respectivamente. La distancia de Levenshtein $d_L(a, b) = M_{m,n}$. En la propuesta original de Levenshtein $c_s = c_e = c_i = 1$. Posteriormente, otros autores han propuesto diferentes combinaciones de valores para estas operaciones dependiendo de los usos de esta métrica.

$M_{i,0}$ y $M_{0,j}$ representan la distancia de edición entre una cadena de longitud i o j, y la cadena vacía. Para dos cadenas no vacías $a_1 a_2 \dots a_i$ y $b_1 b_2 \dots b_j$ de longitud i y j, se asume inductivamente que las distancias de todas las cadenas de longitud menor han sido previamente calculadas, antes de calcular la correspondiente a la trasformación de $a_1 a_2 \dots a_i$ en $b_1 b_2 \dots b_j$. Nótese que si $a_i = b_j$, entonces no se requiere realizar ninguna operación o bien puede considerarse como una sustitución de un carácter por sí mismo con costo cero. Por el contrario, si $a_i \neq b_j$, es necesario utilizar las operaciones básicas para transformar $a_1 a_2 \dots a_{i-1}$ en $b_1 b_2 \dots b_j$.

A continuación se muestra un ejemplo del cálculo de la distancia de edición para transformar la palabra *pumas* en *unam*, asignando un costo de uno a cada operación. La Figura 6.1 muestra la matriz inicial en la que se muestran el renglón y la columna cero. En la Figura 6.2 se muestra el valor de $M_{1,1}$ que se obtiene de mín$(0 + 1, 1 + 1, 1 + 1)$ = mín$(1, 2, 2) = 1$

según la función (6.1). El proceso continúa hasta calcular todos los valores de la matriz como se muestra en la Figura 6.3. Como puede observarse $d_L(\text{pumas}, \text{unam}) = M_{5,4} = 3$. La secuencia en negritas indica una trayectoria para transformar *pumas* en *unam*.

		u	n	a	m
	0	1	2	3	4
p	1				
u	2				
m	3				
a	4				
s	5				

Figura 6.1: Matriz inicial para el cálculo de $d_L(\text{pumas}, \text{unam})$.

		u	n	a	m
	0	1	2	3	4
p	1	1			
u	2				
m	3				
a	4				
s	5				

Figura 6.2: $M_{1,1} = \text{mín}(0+1, 1+1, 1+1) = \text{mín}(1, 2, 2) = 1$.

		u	n	a	m
	0	1	2	3	4
p	1	**1**	2	3	4
u	2	**1**	2	3	4
m	3	2	**2**	3	3
a	4	3	3	**2**	3
s	5	4	4	3	**3**

Figura 6.3: Matriz con todos los valores calculados, $d_L(\text{pumas}, \text{unam}) = 3$.

La secuencia de operaciones en negritas se puede traducir de la siguiente manera: 1) sustitución de p por u (pumas $\rightarrow$ uumas), 2) eliminación de u (uumas $\rightarrow$ umas), 3) sustitución de m por n (umas $\rightarrow$ unas), 4) sustitución

de a por a, esta operación tiene costo cero (umas $\rightarrow$ unas), 5) sustitución de s por m (unas $\rightarrow$ unam).

Como puede observarse, al moverse en diagonal hacia abajo se realiza una sustitución. Al hacerlo hacia la derecha se efectúa una inserción y moverse hacia abajo se indica una eliminación. Para una descripción más amplia de la distancia de Levenshtein se remite al lector a las referencias [GN01, BB07].

Si bien en el transcurso del cálculo de esta métrica las operaciones individuales de transformación entre cadenas pueden carecer de sentido en el contexto de los árboles; por ejemplo, en casos donde un "(" puede ser sustituido por un "0", la distancia de Levenshtein puede ser considerada como una métrica adecuada para establecer una comparación inicial entre códigos de cadenas ya que ofrece una cuantificación de la similitud global entre cadenas.

6.3. Distancia de edición en árboles

La distancia de edición en árboles es una métrica de similitud que fue introducida por K. Tai en 1979 como una generalización del problema de la distancia de edición en cadenas presentado en la sección anterior [KT79]. La distancia de edición entre dos árboles es el costo menor entre las secuencias de operaciones que transforman un árbol en el otro, donde el costo de la secuencia es la suma de los costos individuales de las operaciones aplicadas.

Las operaciones básicas son: eliminar un nodo y conectar sus nodos hijos al nodo padre manteniendo el orden en aristas y nodos, insertar un nodo y en su caso reordenar el subárbol formado por los hijos, y renombrar un nodo. La Figura 6.4 muestra un ejemplo sencillo de la aplicación de las operaciones para transformar el árbol (a) en (c).

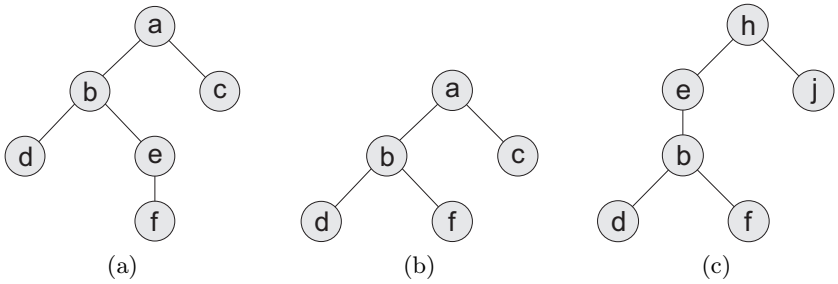
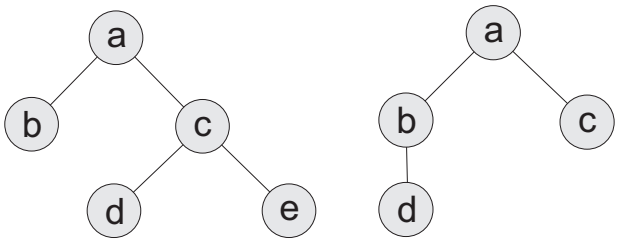


Figura 6.4: Transformación de (a) en (c) mediante operaciones básicas. (a) es el árbol original, (b) es el árbol obtenido al eliminar el nodo e. (c) es el árbol resultante de agregar el nodo e y renombrar los nodos a por h y c por j.

Con el transcurso del tiempo, varios han sido los métodos propuestos para resolver el problema de la distancia de edición en árboles, la solución original de K. Tai fue revisada y mejorada en diversos aspectos por K. Zhang y D. Shasha [ZS89], E. Demaine [DMR09] y P. Klein [PK98] entre otros autores. En general, para resolver el problema de la distancia de edición en árboles se utiliza una solución recursiva que descompone los árboles en subárboles más pequeños. Puesto que existen muchas maneras de descomponer los árboles, las propuestas de solución provienen de la implementación de distintas estrategias para recorrer el espacio generado por los subárboles del árbol origen que se va a transformar en el árbol destino.

Como alternativa para calcular la distancia de edición entre árboles se propone utilizar el algoritmo RTED planteado por M. Pawlik y N. Augsten [PA11]. Esta solución emplea estrategias de trayectorias que incluyen las soluciones publicadas previamente de tal manera que el número de subproblemas derivados de la descomposición en subárboles es menor o igual que en el caso de los algoritmos publicados previamente.

Ejemplo de una estrategia muy sencilla para calcular la distancia de edición entre árboles que incluye el RTED es la que según P. Bille [PB05] se debe originalmente a P. Klein [PK98]. Dados dos árboles F y G cuyos nodos han sido recorridos en postorden, la distancia de edición $d_A(F, G)$ entre F y G se obtiene mediante la aplicación de la siguiente función recursiva:



(a) Árbol F (b) Árbol G

Figura 6.5: La distancia de edición entre ambos es igual a tres.

$$d_A(F,G) = \begin{cases} d_A(\emptyset,\emptyset) = 0, \\ d_A(F,\emptyset) = d_A(F-v,\emptyset) + c_e(v), \\ d_A(\emptyset,G) = d_A(\emptyset,G-w) + c_i(v), \\ \min \begin{cases} d_A(F-v,G) + c_e(v) \\ d_A(F,G-w) + c_i(w) \\ d_A(F-v,G-w) + c_s(v,w) \end{cases} \end{cases} \tag{6.2}$$

donde c_s, c_e y c_i son los costos de las operaciones de sustitución, eliminación e inserción de nodos, respectivamente. La selección de los nodos v y w depende de la estrategia empleada en la descomposición de subárboles.

Para ejemplificar el uso de la función (6.2), en la Figura 6.5 se muestran dos árboles para los cuales se calcula su distancia de edición. Los vértices de ambos árboles se asumen recorridos en postorden. Para F (mostrado en la Figura 6.5(a)), la lista de vértices es *dbeca*. Para G (Figura 6.5(b)) la lista de vértices es *dbca*. En cada iteración, $F - v$ son los subárboles obtenidos de F al remover el nodo v y todas sus aristas. La notación $F - v - w$ indica que primero se elimina v y posteriormente w.

$$d_A(F,G) = \begin{cases} d_A(F-a,G) + c_e(a) \\ d_A(F,G-a) + c_i(a) \\ d_A(F-a,G-a) + c_s(a,a) \end{cases} \tag{6.3}$$

$$d_A(F-a,G) = \begin{cases} d_A(F-a-c,G) + c_e(c) \\ d_A(F-a,G-a) + c_i(a) \\ d_A(F-a-c,G-a) + c_s(c,a) \end{cases} \tag{6.4}$$

	F	$F-a$	$F-a-c$	$F-a-c-e$	$F-a-c-e-d$	$F-a-c-e-d-b$
G	**3**	4	3	3	3	4
$G-a$	4	3	3	2	2	3
$G-a-c$	4	3	2	2	1	2
$G-a-c-b$	4	3	2	1	1	1
$G-a-c-b-d$	5	4	3	2	1	0

Tabla 6.1: Distancias de edición calculadas para los subárboles de F y G cuando los vértices son recorridos en postorden. $d_A(F,G) = 3$, nótese que $F-a-c-e-d-b = G-a-c-b-d = \emptyset$.

$$d_A(F, G-a) = \begin{cases} d_A(F-a, G-a) + c_e(a) \\ d_A(F, G-a-c) + c_i(c) \\ d_A(F-a, G-a-c) + c_s(a,c) \end{cases} \tag{6.5}$$

$$d_A(F-a, G-a) = \begin{cases} d_A(F-a-c, G-a) + c_e(c) \\ d_A(F-a, G-a-c) + c_i(c) \\ d_A(F-a-c, G-a-c) + c_s(c,c) \end{cases} \tag{6.6}$$

La iteración inicial corresponde a la función (6.3). La segunda iteración de d_A corresponde a las funciones (6.4), (6.5) y (6.6). Cabe señalar que en el caso de las funciones (6.3) y (6.6), el costo de la operación de sustitución de un nodo por otro con la misma etiqueta es cero. La aplicación recursiva de la función (6.2) genera la Tabla 6.1. Como puede observarse $d_A(F,G) = 3$.

Debido a que RTED incluye un conjunto de estrategias en el cálculo de la distancia de edición en árboles, resulta ser una métrica más adecuada para comparar AEs y AdBs que la distancia de edición en cadenas presentada en la sección anterior. En la siguiente sección se introduce otra alternativa para calcular la similitud entre árboles distinta a las dos ya presentadas, basada en los aspectos matriciales de la teoría de gráficas.

6.4. Teoría espectral de gráficas

En teoría de gráficas, el espectro ha sido utilizado para describir las características de una gráfica y extraer información acerca de su estructura. Su uso ha tenido poca difusión debido a dos factores: por una parte, varias gráficas pueden tener el mismo espectro; y por otra, el espectro puede cambiar

significativamente con un pequeño cambio en la estructura de la gráfica. No obstante lo anterior, estos factores pueden ser de importancia o no dependiendo del uso específico que se le de al espectro de una gráfica [WZ08]. Un ejemplo de aplicación en la cual no son importantes es el caso de J. Zhang *et al.* que han usado el espectro para comparar representaciones mediales de objetos 3D mediante gráficas dirigidas acíclicas cuyos nodos son las partes constituyentes de dichos objetos [ZSM05].

La teoría espectral de gráficas permite la cuantificación de la similitud en gráficas a través de la distancia espectral que se obtiene mediante el cálculo de los eigenvalores de las representaciones matriciales vistas como transformaciones lineales asociadas a una gráfica [CRS10]. En lo que resta de la sección, las únicas representaciones que se discutirán son las matrices Laplaciana y de adyacencia.

Sea G una gráfica, V su conjunto de vértices y $E \subset V \times V$ el de aristas. Una *representación matricial* de G es una matriz M de tamaño $|V|^2$, tal que un elemento M_{ij} de M representa alguna propiedad del par de vértices i, j y los elementos diagonales M_{ii} únicamente muestran la información relacionada con el vértice i. Un ejemplo muy conocido de representación matricial de G es su matriz de adyacencia A definida como

$$A(u,v) = \begin{cases} 1 & \text{si } (u,v) \in E \\ 0 & \text{en otro caso.} \end{cases} \tag{6.7}$$

Otra representación es la matriz laplaciana, que se calcula de la siguiente manera: sea $\mathbf{D}$ una matriz diagonal cuyos elementos diagonales corresponden al grado de los vértices. La matriz laplaciana se obtiene de la diferencia entre la matriz de grados y la de adyacencia:

$$\mathbf{L} = \mathbf{D} - \mathbf{A}. \tag{6.8}$$

El espectro de una gráfica se obtiene de la representación matricial utilizando un proceso de eigen-descomposición. Sea M una representación matricial de G. La *eigen-descomposición* de M está dada por $M = \Phi\Lambda\Phi^T$ donde $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{|V|})$ es la matriz diagonal con los eigenvalores ordenados decrecientemente y $\Phi = (\phi_1|\phi_2|\ldots|\phi_{|V|})$ es la matriz con los eigenvectores ordenados por columnas. El *espectro* de G es el conjunto de eigenvalores $S = \{\lambda_1, \lambda_2, \ldots, \lambda_{|V|}\}$ con $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{|V|}$.

Hay dos aspectos relacionados con la obtención de la eigen-descomposición de una representación matricial M que vale la pena destacar y en las cuales radica la motivación para utilizar la distancia espectral como una característica de las gráficas:

1. Puesto que los eigenvalores son las soluciones del polinomio característico de M, existe la posibilidad de que las raíces tengan una componente compleja; sin embargo, si G es una gráfica no dirigida, la matriz A es simétrica y por consiguiente, los eigenvalores de A serán reales [CRS10].

2. Dada una gráfica G, existen diferentes maneras de etiquetar sus nodos y aunque cada ordenamiento de los nodos de G dará origen a una matriz de adyacencia distinta, se puede afirmar que el polinomio característico es el mismo en virtud de lo siguiente: si P es una matriz de permutación que reordena los vértices de G, P es invertible puesto que su inversa es P^T. Si $\det(xI - A)$ es el polinomio característico de G, entonces

$$\begin{aligned}\det(xI - A) &= \det(xI - P^T AP) \\ &= \det(P^T xIP - P^T AP) \\ &= \det(P^T(xI - A)P) \\ &= \det(P^T)\det(xI - A)\det(P) \\ &= \det(xI - A).\end{aligned}$$

   De lo anterior se desprende que el espectro de una gráfica es invariante bajo la transformación $M' = P^T MP$. Esto es, dos gráficas isomorfas tendrán el mismo espectro.

La *distancia espectral* $d_S(G_1, G_2)$ entre dos gráficas G_1 y G_2 es la distancia euclidiana entre sus espectros:

$$d_S(G_1, G_2) = \sqrt{\sum_i (s_i^{(1)} - s_i^{(2)})^2} \tag{6.9}$$

donde $s_i^{(1)}$ y $s_i^{(2)}$ son los valores ordenados de los espectros de G_1 y G_2, respectivamente [WZ08]. Cuando los espectros son de tamaños distintos, el espectro de menor tamaño debe ser completado al tamaño del espectro mayor con ceros manteniendo el ordenamiento de los valores originales. Esto es equivalente a añadir vértices disjuntos a la gráfica con el espectro de menor tamaño para que ambas gráficas tengan la misma cardinalidad.

Sobre la coespectralidad de gráficas

Dos gráficas son *coespectrales* si tienen el mismo espectro con respecto a la misma representación matricial [CRS10]. Aunque A. Schwenk [WZ08] demostró que en el caso de árboles, es posible encontrar muchos ejemplos en los cuales dos árboles distintos tienen el mismo espectro, R. Wilson y P. Zhu [WZ08] indican que la representación laplaciana es una mejor representación matricial que la matriz de adyacencia ya que es menos probable que dos gráficas sean coespectrales si se utiliza la matriz laplaciana como representación en lugar de la matriz de adyacencia. Asímismo, muestran que si se combina más de un espectro para caracterizar a una gráfica se puede reducir aún más la probabilidad de la coespectralidad.

En la Tabla 6.2 (tomada de [WZ08]) se muestra la fracción de árboles coespectrales en función del número de vértices para las matrices de adyacencia y laplaciana. En la primera columna (n) se tiene el número de vértices, $t(n)$ es el número de árboles no isomorfos para cada n, las columnas restantes indican las fracciones respecto a $t(n)$ del número de árboles coespectrales para las matrices de adyacencia $\mathbf{A}$, laplaciana $\mathbf{L}$ y cuando son coespectrales en $\mathbf{A}$ y $\mathbf{L}$ al mismo tiempo ($\mathbf{A} + \mathbf{L}$). Considerando la fracción tan pequeña de árboles coespectrales (que además va disminuyendo conforme aumenta el número de vértices) y la facilidad en el cálculo de su espectro, la representación matricial laplaciana ofrece un método para cuantificar la similitud de AEs y AdBs que vale la pena considerar ya que el problema de la coespectralidad puede ser prácticamente ignorado. Por ejemplo, en el caso de árboles con 21 vértices, solo hay 134 árboles coespectrales en más de dos millones de árboles cuando se combinan ambas representaciones matriciales.

Un aspecto de interés para ser estudiado posteriormente es la relación que pueda establecerse entre la estructura de AEs y AdBs con respecto a su espectro, ya que es posible estimar los cambios inducidos por las operaciones de eliminación e inserción de vértices en el polinomio característico de la representación matricial; y por consiguiente, en la distancia espectral [CRS10].

Ejemplos de cálculo de la distancia espectral

A continuación se muestran dos ejemplos del cálculo de la distancia espectral. En el primer ejemplo se calcula la distancia espectral entre el AE más sencillo que corresponde a un voxel y el voxel mismo visto como una gráfica. El segundo ejemplo calcula la distancia espectral entre dos AEs de un mismo sólido. En ambos ejemplos se usa la matriz laplaciana como representación

n	$t(n)$	**A**	**L**	**A + L**
8	23	0.087	0	0
9	47	0.213	0	0
10	106	0.075	0	0
11	235	0.255	0.0255	2
12	551	0.216	0.0109	2
13	1301	0.319	0.0138	2
14	3159	0.261	0.0095	10
15	7741	0.319	0.0062	2
16	19320	0.272	0.0035	14
17	48629	0.307	0.0045	40
18	123867	0.261	0.0019	38
19	317955	0.265	0.0014	64
20	823065	0.219	0.0008	148
21	2144505	0.213	0.0005	134
22	5623756	0.177	0.00028	378
23	14828074	0.168	0.00019	814
24	39299897	-	0.00009	-
25	104636890	-	0.00007	-
26	279792891	-	0.00005	-

Tabla 6.2: Fracción de árboles coespectrales respecto a las matrices de adyacencia (**A**), laplaciana (**L**) y ambas simultáneamente (**A**+**L**), n es el número de vértices y $t(n)$ el número de árboles no isomorfos.

matricial.

1. En la Figura 6.6 se muestra la numeración utilizada para calcular la distancia espectral entre la gráfica del voxel subyacente y su correspondiente árbol envolvente. L_v y L_{AE} son las matrices laplacianas del voxel y el árbol envolvente, S_v y S_{AE} son sus respectivos espectros.

$$L_v = \begin{pmatrix} 3 & -1 & 0 & -1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & -1 & 0 \\ -1 & 0 & -1 & 3 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 3 & -1 & 0 & -1 \\ 0 & -1 & 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & -1 & 0 & -1 & 3 \end{pmatrix}$$

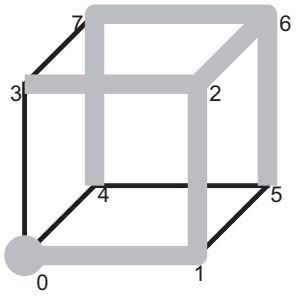


Figura 6.6: Numeración de los vértices de un voxel para el cálculo de la distancia espectral. El AE está indicado en color gris y el punto marca el vértice inicial de la manija.

$$L_{AE} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{pmatrix}$$

$$\begin{aligned} S_{voxel} &= \{0, 2^3, 4^3, 6\} \\ S_{AE} &= \{0, 0.2509, 0.5858, 0.7287, 2, 2.3349, 3.4142, 4.6855\} \end{aligned}$$

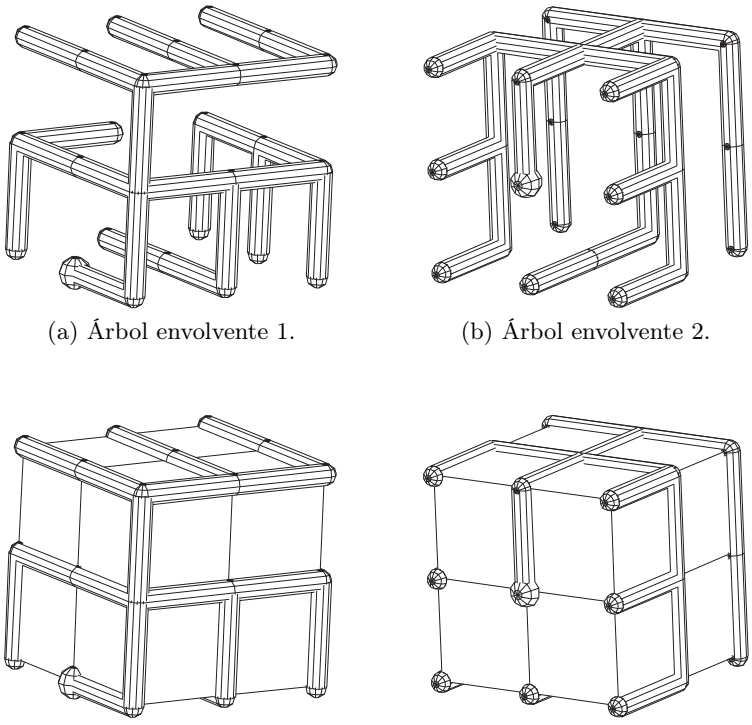
Los exponentes indican la multiplicidad de las raíces del polinomio característico.

2. En la Figura 6.7 se muestran un cubo de $2 \times 2 \times 2 = 8$ voxeles y dos AEs distintos, a los cuales se les calculó la distancia espectral. Los resultados se presentan en la Tabla 6.3. Los espectros de las matrices laplacianas son:

$$S_{cubo} = \{0, 0.8587^3, 2^3, 2.5858^2, 3, 3.4384, 3.5151^3, 4^3, 5^3, 5.4142^2, 6.6262^3, 7.5616\}$$

$$S_{AE_1} = \{-0.1251, 0.0544, 0.0928, 0.1429, 0.1558, 0.2836, 0.4544, 0.7012, 0.9065, 1^2, 1.1288, 1.2918, 1.5201, 2.0512, 2.0722, 2.3822, 2.7161, 3.0085, 3.1432, 3.2089, 3.8082, 4.0650, 4.4319, 4.7829, 5.7227\}$$

$$S_{AE_2} = \{0, 0.0715, 0.0941, 0.1449, 0.1981, 0.2634, 0.4714, 0.5108, 0.7006, 0.7167, 0.7722, 1.2233, 1.5550, 1.8516, 2.1039, 2.1674, 2.2561, 2.8238, 2.9714, 3.0442, 3.2470, 3.4886, 4.0013, 4.5302, 4.8218, 5.9707\}$$



(a) Árbol envolvente 1.

(b) Árbol envolvente 2.

(c) Cubo con el AE_1 sobrepuesto.

(d) Cubo con el AE_2 sobrepuesto.

Figura 6.7: Árboles a los cuales se les calculó su distancia espectral.

	voxel	*AE*	*cubo*	AE_1	AE_2
voxel	0	3.9394	15.4570	6.8153	6.8122
AE		0	17.5984	8.7104	8.6918
cubo			0	9.4758	9.5133
AE_1				0	0.8051
AE_2					0

Tabla 6.3: Distancias espectrales entre las gráficas del voxel, el cubo y los AEs de la Figura 6.7.

Sin demostrarlo formalmente, nótese que los valores de la Tabla 6.3 coinciden con lo que intuitivamente se espera: las distancias $d_S(cubo, AE_1)$, $d_S(cubo, AE_2)$ y $d_S(AE_1, AE_2)$ son menores que $d_S(voxel, cubo)$, ya que como gráficas el cubo y los árboles AE_1 y AE_2 tienen una mayor similitud que el cubo y el voxel.

6.5. Aplicación de las métricas a objetos 3D

Con la finalidad de hacer una breve comparación de las métricas expuestas anteriormente con otros datos de similitud, se seleccionaron tres grupos de objetos. El primer grupo lo constituyen unas curvas 3D (Figuras 6.8 y 6.9) reportadas por E. Bribiesca y W. Aguilar [BA06], cuyos valores de disimilitud son comparados con las distancias de Levenshtein calculadas entre las curvas. El segundo grupo está formado por unos percheros de grosor unitario (Figuras 6.10 y 6.11) publicados en [EB08] que tienen diferentes grados de similitud a los cuales se les calcularon las distancias de Levenshtein y la de edición de árboles utilizando el algoritmo RTED a partir de sus códigos de cadenas 5OT. El tercer grupo de objetos está formado por unas mesas de las cuales se reportan las distancias calculadas (Figura 6.12).

Las valores de disimilitud y distancia de Levenshtein calculadas para el primer grupo se resumen en las Tablas 6.4 (publicada en [BA06]) y 6.5. Puede notarse que en ambas tablas, la pareja de curvas (h)-(i) tiene los valores más pequeños. Lo que implica que son las más parecidas. Las distancias calculadas para el segundo grupo se resumen en las Tablas 6.7 y 6.8. En la referencia original [EB08] se hace una estimación intuitiva de similitud basada en el número de ramas que comparten entre sí todos los modelos. En particular, se afirma que los modelos 6-8 y 3-8 son los que guardan una mayor similitud ya que tienen más ramas en común, lo que coincide con

Curva	A	B	C	D	E	F	G	H	I	J
A	0	0.2264	0.3885	0.5823	0.6563	0.7741	0.2193	0.4092	0.4204	0.252
B		0	0.3278	0.6329	0.689	0.7314	0.4005	0.3445	0.3772	0.1131
C			0	0.8085	0.5249	0.8046	0.309	0.3988	0.3407	0.4003
D				0	0.8	0.6712	0.8618	0.8196	0.7965	0.6329
E					0	0.5612	0.5892	0.4748	0.54	0.5968
F						0	0.4772	0.7208	0.7951	0.7314
G							0	0.2722	0.3	0.4571
H								0	**0.1091**	0.3547
I									0	0.397
J										0

Tabla 6.4: Valores de disimilitud para las curvas mostradas en la Figura 6.9. En negritas se señala la disimilitud menor.

la noción que cuantifica la distancia de Levenshtein como se puede notar en la Tabla 6.7. En el caso de la distancia de edición en árboles (Tabla 6.8) también los modelos 3-8 y 6-8 son los que tienen la menor distancia y por consiguiente una mayor similitud. Llama la atención que en el caso de algunos pares de modelos, RTED los reporte como similares cuando la distancia de Levenshtein los considera poco similares. Esto podría deberse a que RTED explora más estrategias para llevar un árbol en el otro y es posible que encuentre secuencias de transformación más cortas.

Las distancias calculadas para las mesas se resumen en la Tabla 6.9. Las distancias espectrales corresponden a la matriz laplaciana. En el caso de los AdBs, únicamente se han considerado los árboles de bordes planos. Como puede observarse en los datos reportados, de manera consistente las mesas 1 y 3 tienen una mayor similitud que respecto a la 4, ya que sus distancias son menores.

Cabe señalar que las distancias calculadas en este capítulo únicamente permiten una comparación global. Para obtener mayor información acerca de la similitud de la estructura de los objetos en ciertas zonas específicas habría que hacer una comparación de similitud local o híbrida que incluya aspectos globales y locales. De las tres métricas evaluadas la que ofrece mayores beneficios es la distancia de edición en árboles, ya que opera sobre la estructura del árbol y evita hacer operaciones intermedias que a nivel de cadenas o símbolos carecen de sentido como por ejemplo, la sustitución de un paréntesis "(" por un "0", cuando en el contexto de los árboles tendría más sentido sustituir toda una cadena entre dos paréntesis "(" ")" que es lo que determina una rama del árbol a partir de un nodo. Nótese que aún cuando en

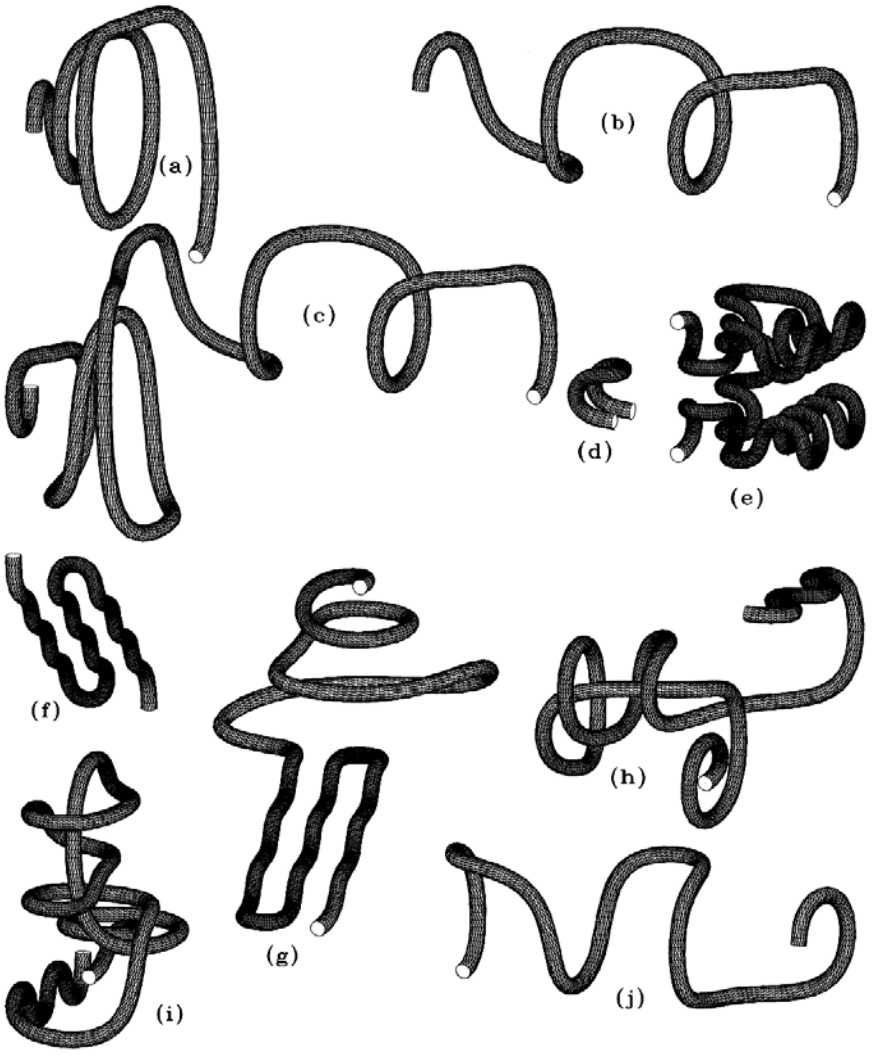


Figura 6.8: Curvas 3D cuya disimilitud se comparó con la distancia de Levenshtein.

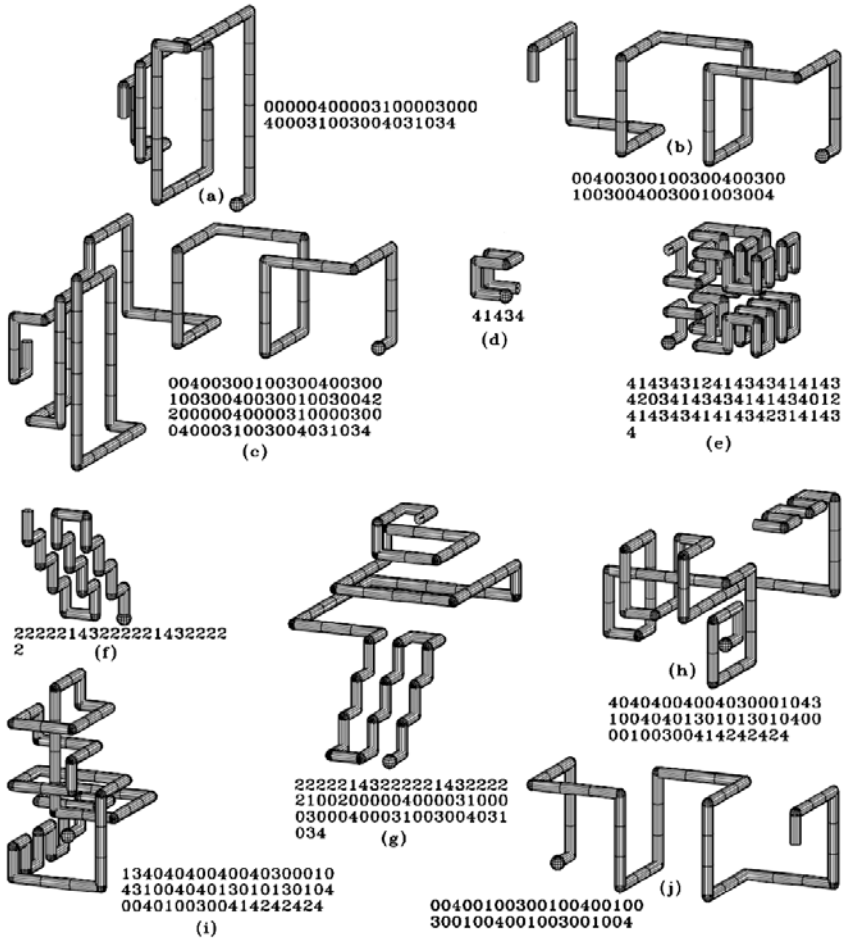


Figura 6.9: Versiones digitalizadas de las curvas de la Figura 6.8.

Curva	A	B	C	D	E	F	G	H	I	J
A	0	14	41	33	47	34	25	25	28	15
B		0	40	34	46	34	34	24	27	9
C			0	74	56	71	33	40	41	41
D				0	56	17	58	52	54	34
E					0	51	47	39	39	46
F						0	42	48	50	34
G							0	38	39	34
H								0	**3**	24
I									0	26
J										0

Tabla 6.5: Valores de distancia de Levenshtein para las curvas mostradas en la Figura 6.9. En negritas se señala la distancia menor.

Modelo	Código 5OT
1	(00000000000(0(0(0(0(0(00)(20))(10))(40))(300))(200))(100))(100422)(200422)(300422)(400422)
2	(000000000(12022)(22022)(32022)(42022))(142222022)(242222022)(342222022)(442222022)
3	(000000000(0000(102)(202)(302)(402))(1002)(2002)(3002)(4002))(100422)(200422)(300422)(400422)
4	(00000000000000100)(2000(30001)(4))(4000(10003)(4))
5	(0000000000000(10123)(20123)(30123)(40123))(100422)(200422)(300422)(400422)
6	(000000000(000(00)(10)(20)(30)(40))(100)(200)(300)(400))(1004)(2004)(3004)(4004)
7	(0000000000(0000)(12222)(22222)(32222)(42222))(100424)(200424)(300424)(400424)
8	(000000000(0000(102)(202)(302)(402))(100)(200)(300)(400))(1004)(2004)(3004)(4004)

Tabla 6.6: Códigos de los percheros de la Figura 6.10.

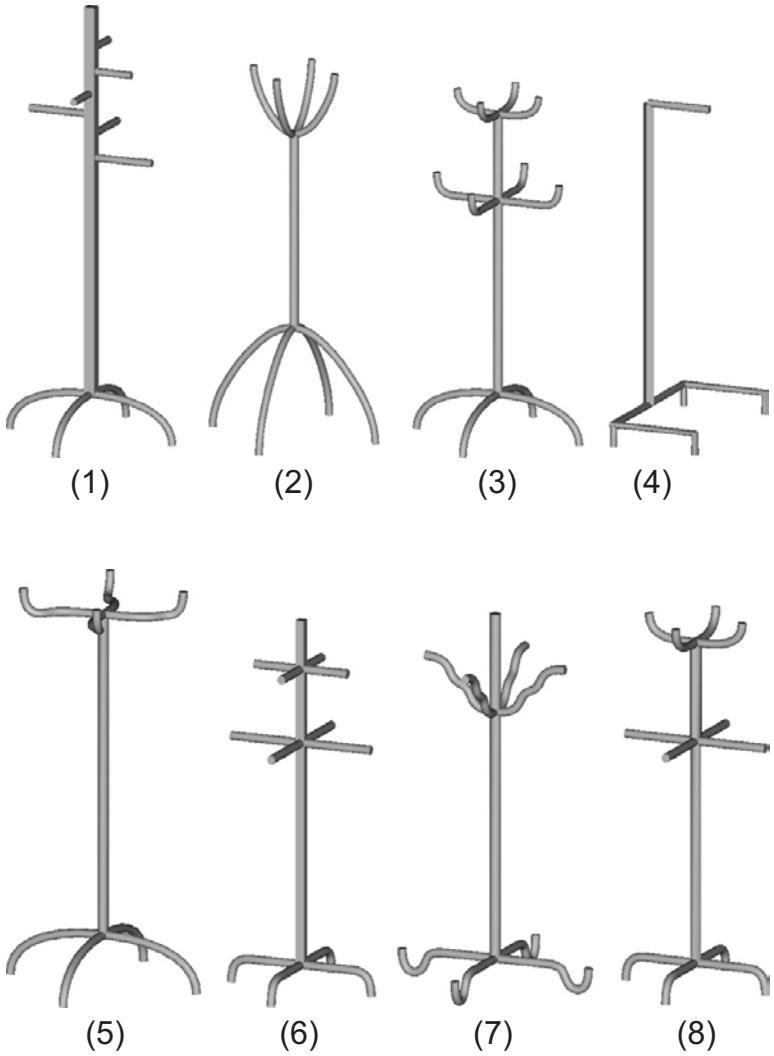


Figura 6.10: Percheros a los cuales se les calcularon las distancias de Levenshtein y la edición en árboles.

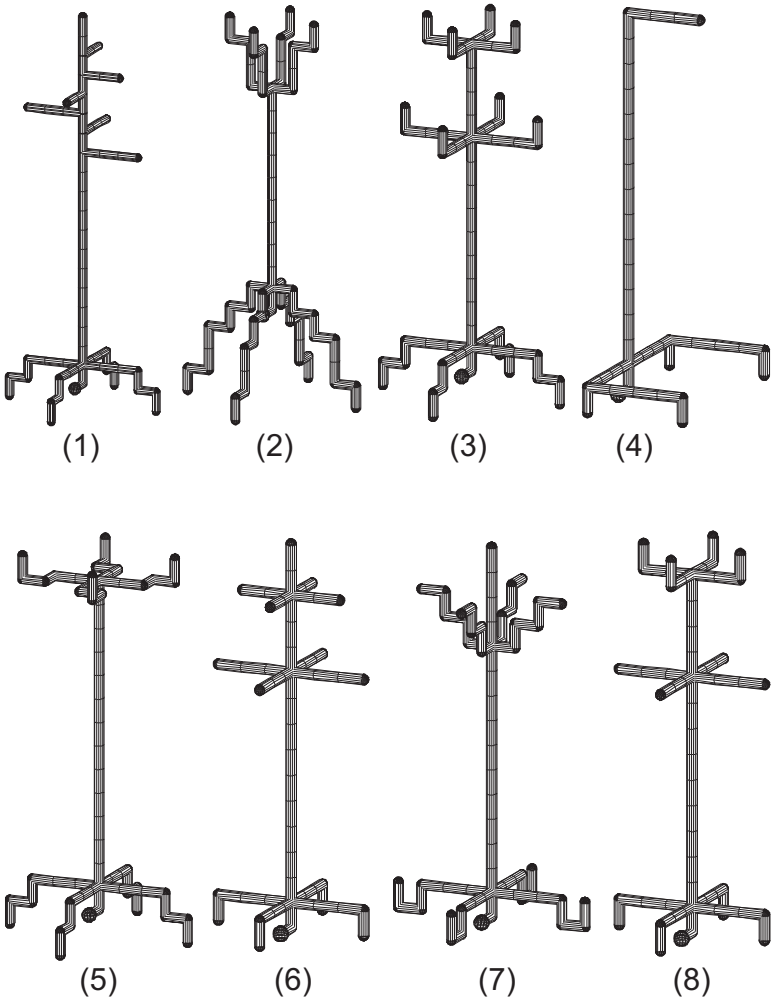


Figura 6.11: Versiones discretas de los percheros representados por los códigos de la Tabla 6.6.

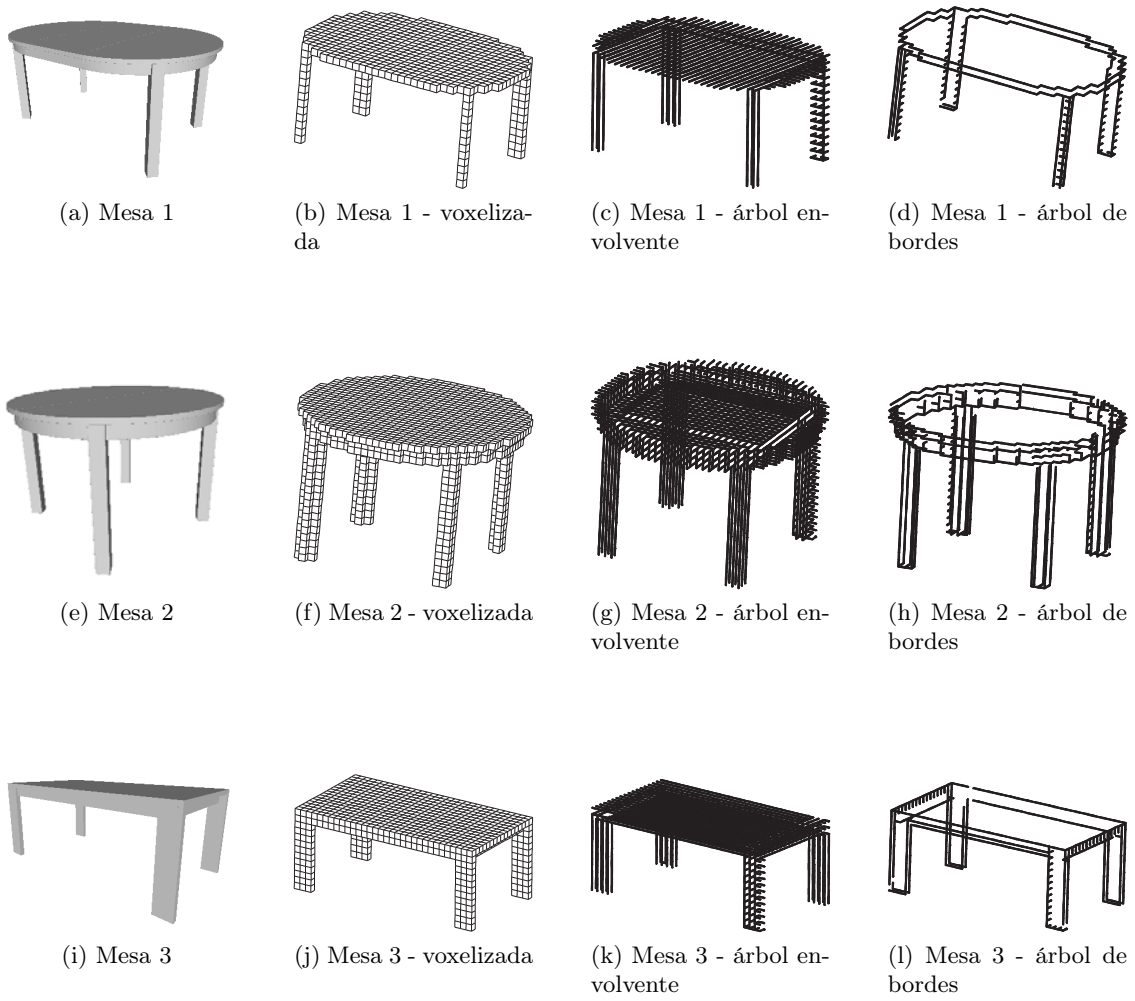
Modelo	1	2	3	4	5	6	7	8
1	0	50	22	48	28	25	37	27
2		0	40	52	40	47	39	43
3			0	50	27	19	32	12
4				0	46	37	41	38
5					0	33	19	32
6						0	37	**7**
7							0	35
8								0

Tabla 6.7: Distancias de Levenshtein para los percheros de la Figura 6.11. En negritas se señala la distancia menor.

Modelo	1	2	3	4	5	6	7	8
1	0	13	14	13	13	11	13	13
2		0	9	12	5	10	6	9
3			0	10	10	10	9	**4**
4				0	8	11	7	10
5					0	11	6	10
6						0	11	6
7							0	9
8								0

Tabla 6.8: Distancias de edición en árboles calculadas con RTED para los percheros de la Figura 6.11. En negritas se señala la distancia menor.

el cálculo de la distancia de Levenshtein se realicen operaciones intermedias que posiblemente carezcan de utilidad en términos de árboles, no invalida su uso como medida global puesto que la cadena sigue siendo una representación unidimensional del sólido. De las opciones revisadas en la literatura se prefirió la implementación RTED [PA11] de la distancia de edición en árboles porque incluye las estrategias más utilizadas en los procesos de descomposición de los árboles.



(a) Mesa 1 (b) Mesa 1 - voxelizada (c) Mesa 1 - árbol envolvente (d) Mesa 1 - árbol de bordes

(e) Mesa 2 (f) Mesa 2 - voxelizada (g) Mesa 2 - árbol envolvente (h) Mesa 2 - árbol de bordes

(i) Mesa 3 (j) Mesa 3 - voxelizada (k) Mesa 3 - árbol envolvente (l) Mesa 3 - árbol de bordes

Figura 6.12: Ejemplos de mesas a los cuales se les calcularon las distancias de Levenshtein, edición en árboles y espectral.

	M_1	M_2	M_3
M_1	0	4021	**1740**
M_2		0	3695
M_3			0

(a) Distancias de Levenshtein (AEs)

	M_1	M_2	M_3
M_1	0	1741	**492**
M_2		0	1587
M_3			0

(b) Distancias de Levenshtein (AdBs)

	M_1	M_2	M_3
M_1	0	1774	**1157**
M_2		0	1902
M_3			0

(c) Distancias RTED (AEs)

	M_1	M_2	M_3
M_1	0	831	**423**
M_2		0	830
M_3			0

(d) Distancias RTED (AdBs)

	M_1	M_2	M_3
M_1	0	154.5448	**59.0559**
M_2		0	161.329
M_3			0

(e) Distancias espectrales

Tabla 6.9: Distancias calculadas para las mesas de la Figura 6.12.

Capítulo 7

Conclusiones y líneas de trabajo futuras

7.1. Conclusiones

El objetivo del presente proyecto de investigación ha sido extender el uso del código de cadenas 5OT para representar sólidos 3D introduciendo el concepto de *árbol envolvente*. Adicionalmente se han explorado dos alternativas, denominadas *árboles de bordes* con la finalidad de obtener una representación basada en lo que podría considerarse un boceto que captura las características globales más importantes del objeto que se representa. Por una parte, se han definido los *árboles de bordes planos* los cuales trazan la superficie envolvente sin considerar ciertos vértices en las caras planas de los sólidos. Por otro lado se han introducido los *árboles de bordes generalizados* que trazan los vértices comunes de una descomposición de la superficie del sólido en planos digitales. Se ha buscado que los árboles de bordes sean una representación más compacta que los árboles envolventes.

Debido a que los diversos autores han publicado sus contribuciones con notaciones y definiciones particulares, un objetivo secundario a lo largo del presente texto ha sido el introducir todos los conceptos y algoritmos bajo un mismo marco de referencia con el objeto de que el lector pueda tener una mejor perspectiva y una mayor comprensión de los problemas, así como la relación que guardan unos con otros. En especial lo relativo al código de cadenas 5OT. A continuación se comentan los resultados obtenidos.

- Se hace una exposición de los modelos reticulares de puntos y de células. Con base en estos modelos se establece un marco de referencia para agrupar los códigos de cadenas 2D y 3D. En especial, se describe

detalladamente el modelo reticular de células (Apéndice A) como antecedente del código 5OT que será la herramienta fundamental para definir los AEs.

- Se proponen los AEs como una representación de sólidos voxelizados en el contexto del modelo reticular de células [BGM12].

- Ya que los AEs únicamente utilizan los vértices de la superficie de los sólidos subyacentes, se propone una extensión de los AEs en $\mathbb{Z}^3$ utilizando los baricentros de los voxeles en lugar de sus vértices en el marco del modelo reticular de puntos que puede ser utilizado para obtener códigos de cadenas 5OT de esqueletos conexos por caras.

- Se proponen dos definiciones de árboles de bordes (AdBs): los planos y los generalizados como una extensión de los AEs. Se ha buscado que ambos tipos de árboles sean un bosquejo que contengan las características geométricas que distinguen al sólido del cual provienen. En los árboles de bordes planos se eliminan vértices sobre las caras planas de la superficie del sólido que se consideran redundantes para este propósito [MBG13, MBG14]. Los árboles de bordes generalizados se utilizan para trazar las intersecciones de los planos provenientes de una descomposición de la superficie del sólido en planos digitales [CB08].

- Se propone un algoritmo susceptible de ser programado que permite calcular los AdBs y los AEs, así como sus respectivos códigos de cadenas [MBG14].

- Puesto que los árboles envolventes y de bordes son una representación unidimensional, se han propuesto tres métricas globales que pueden ser utilizadas para comparar la similitud de sólidos 3D: la distancia de Levenshtein [VL66], la distancia de edición en árboles RTED [PA11] y la distancia espectral [WZ08]. En el caso de la distancia de Levenshtein se usa la descripción del código de cadenas como una cadena de símbolos y el costo que implica el llevar la cadena que representa un sólido en otro mediante operaciones básicas que consisten en insertar, eliminar o sustituir un símbolo. La distancia de edición en árboles, utiliza la dualidad en la representación del código de cadenas. Es un cadena unidimensional, pero al mismo tiempo representa un árbol; por consiguiente, es posible establecer un costo para llevar un árbol en otro a partir de operaciones básicas de inserción, eliminación o sustitución, pero ahora sobre los nodos que constituyen los árboles. La tercera

métrica considera a los árboles como gráficas en general y se usa el espectro de la gráfica para establecer una distancia espectral y asociarla a la similitud. De los experimentos realizados puede concluirse que la distancia de edición en árboles es una métrica adecuada para hacer estimaciones de similitud global entre sólidos representados por AEs y AdBs.

7.2. Líneas de trabajo futuras

Varias son las líneas de trabajo que quedan abiertas a partir de los resultados preliminares del presente proyecto.

- *Uso de voxeles de diferentes resoluciones en AdBs.* En los ejemplos de AdBs mostrados en el texto se observa una tendencia natural a tener muchas cadenas "0", sería deseable poder incluir voxeles de diferente tamaño para eliminarlas, de tal manera que un objeto pudiera ser representado por un árbol A con pocos elementos si el voxel es grande y la resolución es baja. El mismo objeto podría representarse mediante un árbol B con muchos elementos si el voxel es más pequeño (alta resolución). Sería deseable encontrar un método en el que dado un árbol B sea posible deducir el árbol A sin necesidad de reconstruir la superficie del objeto y luego volver a "desbaratarla" para generar el árbol B.

- *Extensión de los AdBs.* El concepto de AdBs puede extenderse en dos direcciones. En la primera, si se aplican diferentes criterios que obedezcan a alguna propiedad física de eliminación es posible trazar objetos complejos tales como montañas o nubes. La segunda, es trazar los AdBs a través de los centros de los voxeles en lugar usan únicamente los vértices de la superficie.

- *Análisis de similitud.* Como se mencionó anteriormente, es necesario explorar con más detalle las características de las métricas propuestas ya que pueden ofrecer información sobre la estructura de los sólidos que pudiese ser de interés para los usuarios. Otra área de oportunidad lo constituyen las métricas locales o las híbridas que combinan aspectos globales y locales. Asímismo, haría falta un análisis cuantitativo con respecto a otros métodos de representación.

- Cabe señalar que para abordar el problema de la comparación y clasificación de sólidos, es necesario explorar un tema fundamental en relación con los objetos 3D que de momento rebasa los alcances del presente

proyecto. Se requiere obtener un método adecuado para orientar los objetos de manera que los AEs y AdBs tengan puntos iniciales con ciertas propiedades comunes a todos los sólidos que facilite su comparación o análisis. Cabe señalar que la eliminación de la ambigüedad en la orientación en objetos 3D es un tema complejo y ha dado origen a investigaciones publicadas como [GC93] o [BDW00].

Apéndice A
Complejos CW

En este apéndice se presentan las definiciones que permiten considerar a los pixeles (para el caso 2D) y voxeles (para el caso 3D) como complejos CW. Un complejo CW, es un espacio que se obtiene a partir de un conjunto discreto de puntos al cual se le van agregando células de dimensiones sucesivamente superiores. El objetivo de este apéndice es mostrar que $\mathbb{R}^3$ con la topología estándar puede ser considerado un complejo CW en el cual cada voxel tiene coordenadas enteras y está formado de componentes de menor dimensión que impiden la construcción de objetos 3D que no correspondan con la noción intuitiva de pixel o de voxel como se explica más adelante.

Cabe mencionar que la relación de los pixeles y voxeles con los complejos CW se menciona en una gran cantidad de textos y aunque intuitivamente es clara la asociación no se dan explícitamente las ecuaciones y esa es la razón del presente apéndice. En lo que sigue se usan las definiciones de las referencias [Ha02], [JM84] y [Ma82].

A.1. Definiciones básicas

Definición A.1. Un *espacio topológico* es una pareja ordenada $(X, \mathcal{T})$ donde X es un conjunto y $\mathcal{T}$ es una familia de subconjuntos de X (llamada la *topología* de X) cuyos elementos son llamados *conjuntos abiertos* que satisfacen las siguientes condiciones:

a) $\emptyset, X \in \mathcal{T}$.

b) Si $A_1, A_2, \ldots, A_n \in \mathcal{T}$, entonces $A_1 \cap A_2 \cap \ldots \cap A_n \in \mathcal{T}$.

c) Si $\{A_\alpha\}_{\alpha \in A} \in \mathcal{T}$, entonces $\bigcup_{\alpha \in A} A_\alpha \in \mathcal{T}$.

Dos ejemplos de espacios topológicos son los siguientes:

1. Sea $X = \{a, b, c\}$, la familia $\mathcal{T} = \{\emptyset, \{a\}, \{a, b\}, X\}$ es una topología en X.

2. Si todos los subconjuntos de $\mathbb{Z}$ se declaran conjuntos abiertos, entonces $\mathbb{Z}$ será un espacio topológico con la *topología discreta*. $\mathbb{Z}^2$ y $\mathbb{Z}^3$ son espacios topológicos con la topología discreta.

Definición A.2. Un conjunto A es *cerrado* en X si y solamente si $X - A \in \mathcal{T}$. Si $x \in X$, entonces un conjunto abierto que contenga a x se dice que es una *vecindad* de x.

Definición A.3. El *disco unitario n-dimensional* (o n-disco) de $\mathbb{R}^n$ es el subespacio $D^n = \{\boldsymbol{x} \in \mathbb{R}^n : |\boldsymbol{x}| \leq 1\}$, donde $|\cdot| : \mathbb{R}^n \to [0, \infty)$ es la norma usual en $\mathbb{R}^n$. El *n-disco abierto*, denotado por $\text{int}(D^n)$ es el interior de D^n en $\mathbb{R}^n : \text{int}(D^n) = \{\boldsymbol{x} \in \mathbb{R}^n : |\boldsymbol{x}| < 1\}$. La *frontera* de D^n en $\mathbb{R}^n$ es la $(n-1)$-esfera: $\partial D^n = S^{n-1} = \{\boldsymbol{x} \in \mathbb{R}^n : |\boldsymbol{x}| = 1\}$. Nótese que $\text{int}(D^n) = D^n - \partial D^n$. El cubo *n-dimensional* está definido como $[0,1]^n = \underbrace{[0,1] \times \ldots \times [0,1]}_{n\text{-veces}}$.

Para mantener la consistencia de la notación se tienen las siguientes convenciones. El $0-$disco, D^0 es igual a $\mathbb{R}^0$, que es el conjunto que consta de un solo punto, e $\text{int}(D^0) = \partial D^0 = S^{-1} = \emptyset$.

Definición A.4. Una *n-célula* c^n es un espacio homeomorfo a $\text{int}(D^n) = D^n - \partial D^n$. La *dimensión* de una n-célula será precisamente n.

De acuerdo con lo anterior, una 0-célula será un conjunto de consta de un punto. Una 1-célula será un conjunto homeomorfo a $(-1, 1)$, una 2-célula será un conjunto homeomorfo al disco centrado en el origen con radio igual a uno sin incluir su frontera, y una 3-célula será un conjunto homeomorfo a una esfera sin incluir la superficie de ésta.

Definición A.5. Un *complejo CW* es un espacio X consistente en una sucesión creciente $X^0 \subset X^1 \subset X^2 \subset \cdots$ que cumple con las siguientes condiciones:

a) Existe un conjunto discreto X^0, cuyos puntos serán considerados como 0-células.

b) De manera inductiva; para cada $n > 0, X^n$ se obtiene adjuntando una colección $\{c_1^n, c_2^n, \ldots, c_m^n\}$ de n-células a X^{n-1}, mediante un conjunto de funciones continuas $\varphi_i : D^n \to X, i \in \{1, 2, \ldots, m\}$ tales que mapean homeomórficamente $\text{int}(D^n)$ en c_i^n y ∂D^n en una unión finita de células, cada una de dimensión menor que n.

c) $X = \bigcup_n X^n$ y tal que un conjunto A en X es abierto (o cerrado) si y solo si $A \cap X$ es abierto (o cerrado) en X^n, para cada n.

Al conjunto X^n se le llama *n-esqueleto* de X. La denominación CW proviene de la frase en inglés *Closure-finiteness with the Weak topology* que utilizó J. H. C. Whitehead cuando introdujo este tipo de complejos [JM84]. La cerradura finita (Closure-finiteness) se refiere a la condición (b) y la topología débil a la condición (c).

Algunos ejemplos de complejos CW de interés son los siguientes:

1. Un conjunto que admite una descomposición como complejo CW es el intervalo $[0,1]$ con la topología usual de $\mathbb{R}$. Para seguir la notación utilizada en la definición A.5, se hará la siguiente consideración $X = [0,1]$. La condición (a) se cumple si X^0 está constituido por las 0-células $\{0\}$ y $\{1\}$. Considérese la función $\varphi : D^1 \to [0,1]$ definida como $\varphi(x) = 1 + \dfrac{x-1}{2}$, para toda $x \in X$. φ mapea homeomórficamente la 1-célula $(-1,1)$ en el intervalo $(0,1)$; por consiguiente, el intervalo abierto $(0,1)$ es una 1-célula. $\varphi(-1) = 0$ y $\varphi(1) = 1$; por lo tanto, $\varphi(\partial D^1) = \varphi(\{-1,1\}) = \{0\} \cup \{1\}$ es la unión finita de células de dimensión menor que uno (en este caso de dimensión cero), con lo cual se cumple la condición (b). Claramente se cumple la condición (c), ya que $X = [0,1] = \{0\} \cup \{1\} \cup (0,1)$. Sea $A \subset [0,1]$ un conjunto abierto, $A \cap [0,1] = A$ es un conjunto abierto. Análogamente se cumple la condición para conjuntos cerrados. De lo anterior, se puede concluir que el conjunto $X = [0,1]$ es un complejo CW constituido por $X^0 = \{0,1\}$ y $X^1 = X^0 \cup (0,1)$.

2. El conjunto de los números reales $\mathbb{R}$ con la topología usual es un complejo CW. Considérense como $X^0 = \mathbb{Z}$ y $X^1 = \{[a, a+1] : a \in \mathbb{Z}\}$. Para cada intervalo $[a, a+1] \in X^1$, la función $\varphi_a : D^1 \to \mathbb{R}$ definida como $\varphi_a(x) = a + 1 + \dfrac{x-1}{2}$ mapea homeomórficamente $\text{int}(D^1) = (-1,1)$ en $c_a^1 = (a, a+1)$ y $\varphi_a(\partial D^1) = \{a, a+1\}$. Por las razones análogas a las mencionadas en el ejemplo anterior (de hecho, es el caso cuando $a = 0$), $\mathbb{R}$ con la topología usual es un complejo CW.

3. El cubo $[0,1]^3$ es un complejo CW.

   i) Considérense en $[0,1]^3$ el conjunto de vértices $\boldsymbol{v_0} = (0,0,0), \boldsymbol{v_1} = (0,0,1), \boldsymbol{v_2} = (0,1,0), \boldsymbol{v_3} = (0,1,1), \boldsymbol{v_4} = (1,0,0), \boldsymbol{v_5} = (1,0,1), \boldsymbol{v_6} = (1,1,0), \boldsymbol{v_7} = (1,1,1)$.

ii) Las funciones $\varphi^1_{m,n} : D^1 \to L_{m,n}$, donde $L_{m,n} = \boldsymbol{v_m} + t(\boldsymbol{v_n} - \boldsymbol{v_m}), t \in (0,1)$ es el segmento de recta que une los vértices $\boldsymbol{v_m}$ y $\boldsymbol{v_n}$, mapean homeomórficamente $\text{int}(D^1)$ en $L_{m,n}$ y $\partial(D^1)$ en $\{\boldsymbol{v_m}, \boldsymbol{v_n}\}$. El conjunto de segmentos entre vértices 6-adyacentes está integrado por $L_{0,1}, L_{0,2}, L_{0,4}, L_{1,3}, L_{1,5}, L_{2,3}, L_{2,6}, L_{3,7}, L_{4,5}, L_{4,6}, L_{5,7}, L_{6,7}$.

iii) La función $\alpha : D^2 \to [0,1]^2$, definida como

$$\alpha(\boldsymbol{x}) = \begin{cases} (0.5, 0.5) & \text{si } \boldsymbol{x} = \boldsymbol{0} \\ (0.5, 0.5) + \dfrac{\max(|x_1|, |x_2|)}{2|\boldsymbol{x}|}\boldsymbol{x} & \text{si } \boldsymbol{x} \neq \boldsymbol{0} \end{cases}$$

donde $\boldsymbol{x} = (x_1, x_2) \in D^2$, mapea homeomórficamente $\text{int}(D^2)$ en $(0,1)^2$. Si $A = (0,0), B = (1,0), C = (1,1)$ y $D(0,1)$; entonces, $\alpha(\partial(D^2))$ es la unión de los segmentos de recta $\overline{AB}, \overline{BC}, \overline{CD}$ y $\overline{DA}$.

iiii) El conjunto de funciones $\beta_{0,1,2}, \beta_{0,2,4}, \beta_{0,1,4}, \beta_{1,3,5}, \beta_{2,3,6}, \beta_{4,5,6}$ definidas por $\beta_{i,j,k} : [0,1]^2 \to [0,1]^3$, donde $s, t \in [0,1]$; son homeomorfismos entre $[0,1]^2$ y alguna de las seis caras que constituyen $[0,1]^3$.

$$\beta_{i,j,k}(s,t) = \boldsymbol{v_i} + (\boldsymbol{v_j} - \boldsymbol{v_i})s + (\boldsymbol{v_k} - \boldsymbol{v_i})t$$

Las secuencias i, j, k indican que la cara $C_{i,j,k}$ pasa por el vértice $\boldsymbol{v_i}$ y está determinada por los vectores $\boldsymbol{v_j} - \boldsymbol{v_i}$ y $\boldsymbol{v_k} - \boldsymbol{v_i}$. Cada función mapea $\text{int}([0,1]^2)$ en el interior de la cara correspondiente $(0,1)^2$ y $\partial[0,1]^2$ en las fronteras de cada cara.

v) Considérese el conjunto de funciones definidas por la composición de las funciones mencionadas en los dos incisos anteriores: $\varphi^2_{i,j,k} = \beta_{i,j,k} \circ \alpha : D^2 \to [0,1]^3$. Estas funciones mapean homeomórficamente D^2 en la cara $C_{i,j,k}$ de tal manera que $\text{int}([0,1]^2)$ tiene como imagen el mismo conjunto ahora inmerso en $\mathbb{R}^3$ y $\partial[0,1]^2$ las fronteras de las caras correspondientes. Por consiguiente, se puede concluir que $[0,1]^3$ es un complejo CW.

vi) Si $\boldsymbol{x} = (x_1, x_2, x_3) \in \mathbb{R}^3$, la función

$$\varphi^3(\boldsymbol{x}) = \begin{cases} (0.5, 0.5, 0.5) & \text{si } \boldsymbol{x} = \boldsymbol{0} \\ (0.5, 0.5, 0.5) + \dfrac{\max(|x_1|, |x_2|, |x_3|)}{2\|\boldsymbol{x}\|}\boldsymbol{x} & \text{si } \boldsymbol{x} \neq \boldsymbol{0} \end{cases}$$

mapea homeomórficamente $\text{int}(D^3)$ en el $\text{int}([0,1]^3)$ y ∂D^3 en la superficie de $[0,1]^3$ que de acuerdo con los incisos ii)-v), puede considerarse una unión finita de células de dimensión menor que tres.

vii) De los incisos i) - vi) se puede afirmar que el $[0,1]^3$ es un complejo CW si se construye mediante los siguientes conjuntos:

$X^0 = \{\boldsymbol{v_0}, \boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}, \boldsymbol{v_4}, \boldsymbol{v_5}, \boldsymbol{v_6}, \boldsymbol{v_7}\}$,

$X^1 = \{L_{0,1}, L_{0,2}, L_{0,4}, L_{1,3}, L_{1,5}, L_{2,3}, L_{2,6}, L_{3,7}, L_{4,5}, L_{4,6}, L_{5,7}, L_{6,7}\}$ y el conjunto de funciones $\{\varphi^1_{0,1}, \varphi^1_{0,2}, \varphi^1_{0,4}, \varphi^1_{1,3}, \varphi^1_{1,5}, \varphi^1_{2,3}, \varphi^1_{2,6}, \varphi^1_{3,7}, \varphi^1_{4,5}, \varphi^1_{4,6}, \varphi^1_{5,7}, \varphi^1_{6,7}\}$,

$X^2 = \{C_{0,1,2}, C_{0,2,4}, C_{0,1,4}, C_{1,3,5}, C_{2,3,6}, C_{4,5,6},\}$ y el conjunto de funciones $\{\varphi^2_{0,1,2}, \varphi^2_{0,2,4}, \varphi^2_{0,1,4}, \varphi^2_{1,3,5}, \varphi^2_{2,3,6}, \varphi^2_{4,5,6}\}$

$X^3 = D^3$ y la función φ^3.

4. El espacio tridimensional $\mathbb{R}^3$ con la topología usual puede considerarse un complejo CW, si $X^0 = \mathbb{Z}^3$ y se usan las ideas de los incisos 2 y 3, agregando las traslaciones adecuadas a las familias de funciones mencionadas en el inciso vii) para ir construyendo el complejo CW análogo a $[0,1]^3$ en la posición correspondiente.

Cabe señalar que el proceso incluido en los incisos i) - vii) no es el único para mostrar que $[0,1]^3$ es un complejo CW. Por ejemplo, se sabe que si X es un complejo CW, $X \times [0,1]$ con la topología producto es un complejo CW [Ha02] (Teorema A.6). Por consiguiente; los espacios $[0,1]^2, [0,1]^3$, y en general $[0,1]^n$ son complejos CW.

Una consecuencia importante de considerar al espacio real $\mathbb{R}^3$ como un complejo CW y su división en voxeles, es la imposibilidad de usar complejos que únicamente incluyan vértices, aristas o caras aisladas como la mostrada en la Figura 1. Aunque sean complejos CW correctamente formados por $0, 1$ o 2-células; en el contexto de $\mathbb{R}^3$ únicamente se permiten cubos sólidos o voxeles constituidos por 8 vértices o 0-células, 12 aristas o 1-células, 6 caras o 2-células y un cubo o 3-célula. En el contexto de $\mathbb{R}^2$, los pixeles están formados por cuadrados rellenos formados por 4 0-células, 4 1-células y una 2-célula.

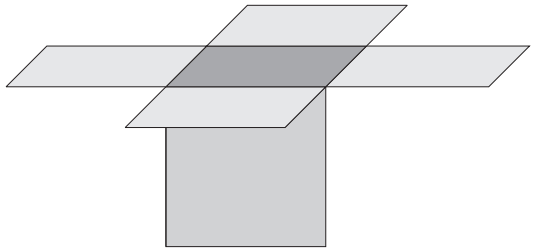


Figura 1: Ejemplo de complejo CW formado por 6 2-células que no es posible manejar en el contexto de $\mathbb{R}^3$ como complejo CW.

Referencias

[AB15] W. Aguilar, E. Bribiesca, "Symmetry detection in 3D chain coded discrete curves and trees", *Pattern Recogn.*, vol. 48, núm. 4, 1416-1435, 2015.

[ABB09] T. Asano, V. Brimkov, R. Barneva, "Some theoretical challenges in digital geometry: A perspective", *Discrete Appl. Math.*, vol. 157, págs. 3362-3371, 2009.

[AG87] A. Guzman, "Canonical shape description for 3-d stick bodies", Microelectronics and Computer Technology Corporation, Reporte técnico ACA-254-87, Austin, TX, 1987.

[ASS11] C. Arcelli, G. Sanniti di Baja, L. Serino, "Distance-Driven Skeltonization in Voxel Images", *IEEE Trans. on Pattern Anal. Mach. Intell.*, vol. 33, págs. 709-719, 2011.

[BA06] E. Bribiesca, W. Aguilar, "A Measure of Shape Dissimilarity for 3D Curves", *Int. Journal of Contemp. Math. Sciencies*, vol. 15, núm. 1, 727-751, 2006.

[BB07] H.-J. Böckenhauer, D. Bongartz, *Algorithmic Aspects of Bioinformatics*, Springer, 2007.

[BCK07] V. Brimkov, D. Coeurjolly, R. Klette, "Digital planarity - a review", *Discrete Appl. Math.*, vol. 155, págs. 468-495, 2007.

[BDW00] H. Bülow, L. Doodley, D. Wermser, "Application of principal axes for registration of NMR image sequences", *Pattern Recogn. Lett.*, vol. 21, págs. 329-336, 2000.

[BG80] E. Bribiesca, A. Guzmán, "How to describe pure form and how to measure differences in shapes using shape numbers", *Pattern Recogn.*, vol. 12, núm. 2, págs. 101-112, 1980.

[BGM10] E. Bribiesca, A. Guzmán, L.A. Martínez, "Representation of 3-D Objects Using Enclosing Trees", en *Computational Techniques and Algorithms for Image Processing: Reviews, Principles and Applications on Pattern Recognition, Image Enhancement, Compression and Watermarking*. Editores S. Ramakrishnan e I. M. M. El Emary, Lambert Academic Publishing, 2010.

[BGM12] E. Bribiesca, A. Guzmán, L.A. Martínez, "Enclosing trees", *Pattern Anal. Appl.*, vol. 15, núm. 1, págs. 1-17, 2012.

[BK08] V. E. Brimkov, R. Klette, "Border and SurfaceTracing - Theoretical Foundations", *IEEE Trans. on Pattern Anal. Mach. Intell.*, vol. 30, núm. 4, págs. 577-590, 2008.

[BM08] J. A. Bondy, U. S. R. Murty, *Graph Theory*, Springer, 2008.

[BMP02] S. Belongie, J. Malik, J. Puzicha, "Shape Matching and Object Recognition Using Shape Contexts", *IEEE Trans. on Pattern Anal. Mach. Intell.*, vol. 24, págs. 509-522, 2002.

[Bu03] L. Buzer, "A linear incremental algorithm for naive and standard digital lines and planes recognition", *Graph. Models*, vol. 65, núm. 1-3, págs. 61-76, 2003.

[BU97] H. Bunke, "On a relation between graph edit distance and maximum common subgraph", *Pattern Recogn. Lett.*, vol. 18, núm. 8, págs. 689-694, 1997.

[BV01] E. Bribiesca, C. Velarde, "A formal language approach for a 3D curve representation", *Comput. Math. Appl.*, vol. 42, núm. 12, págs. 1571-1584, 2001.

[CB08] E. Charrier, L. Buzer, "An efficient and quasi linear worst-case time algorithm for digital plane recognition", *Discrete Geometry for Computer Imagery*, Lecture Notes in Computer Science, vol. 4992, págs. 346-357, 2008.

[CEM10] J. P. Carson, D. R. Einstein, K. R. Minard, M. V. Fanucchi, C. D. Wallis, Richard A. Corley, "High resolution lung airway cast segmentation with proper topology suitable for computational fluid dynamic simulations", *Comput. Med. Imag. Grap.*, vol. 34, núm. 7, págs. 572-578, 2010.

[CL05] A. Chaturvedi, Z. Lee, "Three-dimensional segmentation and skeletonization to build an airway tree data structure for small animals", *Phys. Med. Biol.*, vol. 50, núm. 7, págs. 1405-1419, 2005.

[CRS10] D. Cvetkovic, P. Rowlinson, S. Simic, *An Introduction to the Theory of Graph Spectra*, Cambridge University Press, 2010.

[CSM07] N. D. Cornea, D. Silver, P. Min, "Curve-Skeleton Properties, Applications and Algorithms", *IEEE Trans. Vis. Comput. Graphics*, vol. 13, núm. 3, págs. 530-548, 2007.

[DGtal] *DGtal - Digital Geometry Tools and Algorithms*, abril 2015, http://dgtal.org.

[DH73] R. O. Duda, P. R. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, 1973.

[DL98] D. Lin, "An Information-Theoretic Definition of Similarity", *Proc. of the Fifteenth International Conference on Machine Learning*, Madison, Wisconsin, págs. 296-304, 1998.

[DMR09] E. D. Demaine, "An Optimal Decomposition Algorithm for Tree Edit Distance", *ACM T. Algorithms*, vol. 6, núm. 1, art. núm. 2, 2009.

[EB00] E. Bribiesca, "A chain code for representing 3D curves", *Pattern Recogn.*, vol. 33, núm. 5, págs. 755-765, 2000.

[EB08] E. Bribiesca, "A method for representing 3D tree objects using chain coding", *J. Vis. Comun. Image R.*, vol. 19, núm. 3, págs. 184-198, 2008.

[EB99] E. Bribiesca, "A new chain code", *Pattern Recogn.*, vol. 32, núm. 2, págs. 235-251, 1999.

[FLR09] Y. Fang, Y. Liu, K. Ramani, "Three dimensional shape comparison of flexible protein using the local-diameter descriptor", *BMC Struct. Biol.*, vol. 9, art. núm 29, 2009.

[FMK03] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, D. Jacobs, "A Search Engine for 3D Models", *ACM T. Graphic.*, vol. 22, núm. 1, págs. 83-105, 2003.

[GC93] J. M. Galvez, M. Canton, "Normalization and shape recognition of three-dimensional objects by 3D moments", *Pattern Recogn.*, vol. 26, núm. 5, págs. 667-681, 1993.

[GDZ05] Y. Gerard, I. Debled-Rennesson, P. Zimmermann, "An elementary digital plane recognition algorithm", *Discrete Appl. Math.*, vol. 151, núm. 1-3, págs. 169-183, 2005.

[GN01] G. Navarro, "A Guided Tour to Approximate String Matching", *ACM Comput. Surv.*, vol. 33, núm. 1, págs. 31-88, 2001.

[Go04] P.F. Gorder, "3DESS: A Search Engine Enters The Third Dimension", *Comput. Sci. Eng.*, vol. 6, núm. 6, págs. 4-7, 2004.

[GS06] G. Sanniti di Baja, "Skeletonization of digital objects", *Progress in Pattern Recognition, Image Analysis and Applications*, Lecture Notes in Computer Science, vol. 4225, págs. 1-13, 2006.

[GW07] R. F. Gonzalez, R. E. Woods, *Digital Image Processing*, Prentice Hall, 2007.

[Ha02] A. Hatcher, *Algebraic Topology*, Cambridge University Press, 2002.

[HF61] H. Freeman, "On the Encoding of Arbitrary Geometric Configurations", *IRE Trans. on Electronic Computers*, vol. EC-10, núm. 2, págs. 260-268, 1961.

[HF74] H. Freeman, "Computer processing of line drawing images", *ACM Comput. Surv.*, vol. 6, núm. 1, págs. 57-97, 1974.

[IBS96] J. M. Iñesta, M. Buendía, M. A. Sarti, "Local Symmetries of Digital Contours from their Chain Codes", *Pattern Recogn.*, vol. 29, núm. 10, 1737-1749, 1996.

[IJL05] N. Iyer, S. Jayanti, K. Lou, Y. Kalyanaraman, K. Ramani, "Three Dimensional Shape Searching: State-of-the-art Review and Future Trends", *Comput. Aided Design*, vol. 37, núm. 5, págs. 509-530, 2005.

[JKR09] S. Jayanti, Y. Kalyanaraman, K. Ramani, "Shape-based clustering for 3D CAD objects: A comparative study of effectiveness", *Comput. Aided Design*, vol. 41, núm. 12, págs. 999-1007, 2009.

[JL93] J. C. Lin, "Universal principal axes: an easy-to-construct tools useful in defining shape orientations for almost every kind of shape", *Pattern Recogn.*, vol. 26, núm. 4, 485-493, 1993.

[JM84] J. R. Munkres, *Elements of Algebraic Topology*, Addison-Wesley, 1984.

[JZ12] N. A. Jusoh, J. M. Zain, "Application of Freeman Chain Codes: An Alternative Recognition Technique for Malaysian Car Plates", en *Communications in Computer and Information Science*, Software Engineering and Computer Systems, vol. 179, págs. 581-591, Springer, 2011.

[Ko89] V. A. Kovalevsky, *Finite Topology as Applied to Image Analysis*, *Comput. Vis. Graph. Image Process.*, vol. 46, núm. 2, págs. 141-161, 1989.

[KP08] K. Palágyi, "A 3D fully parallel surface-thinning algorithm", *Theor. Comput. Sci.*, vol. 406, núm. 1-2, págs. 119-135, 2008.

[KR04] R. Klette, A. Rosenfeld, *Digital Geometry*, Morgan Kaufmann, 2004.

[KT79] K. C. Tai, "The tree-to-tree correction problem", *J. ACM*, vol. 26, núm. 3, págs. 422-433, 1979.

[LBG14] E. Lemus, E. Bribiesca, E. Garduño, "Representation of enclosing surfaces from simple voxelized objects by means of a chain code", *Pattern Recogn.*, vol. 47, núm. 4, págs. 1721-1730, 2014.

[LD10] C. Lohou, J. Dehos, "Automatic Correction of Ma and Sonka's Thinning Algorithm Using P-Simple Points", *IEEE Trans. on Pattern Anal. Mach. Intell.*, vol. 32, núm. 6, págs. 1148-1152, 2010.

[Lo98] G. Lohmann, *Volumetric Image Analysis*, Wiley, 1998.

[LRL11] Y. S. Liu, K. Ramani, M. Liu, "Computing the Inner Distances of Volumetric Models for Articulated Shape Description with a Visibility Graph", *IEEE Trans. on Pattern Anal. Mach. Intell.*, vol. 33, núm. 12, págs. 2538-2544, 2011.

[LWWZ07] Y. K. Liu, W. Wei, P. J. Wang, B. Zalik, "Compressed vertex chain codes", *Pattern Recogn.*, vol. 40, núm. 11, págs. 2908-2913, 2007.

[LZ05] Y. K. Liu, B. Zalik, "An efficient chain code with Huffman coding", *Pattern Recogn.*, vol. 38, núm. 4, págs. 553-557, 2005.

[Ma82] W. S. Massey, *A Basic Course in Algebraic Topology*, Springer, 1997.

[MBG13] L.A. Martínez, E. Bribiesca, A. Guzmán, "Voxel-based object representation by means of edging trees", *Proceedings of the 2013 International Conference on Image Processing, Computer Vision, & Pattern Recognition* (IPCV'13), págs. 36-41, WorldComp'13, vol. I, CSREA Press, 2013.

[MBG14] L.A. Martínez, E. Bribiesca, A. Guzmán, "Chain coding representation of voxel-based objects with enclosing, edging and intersecting trees", artículo enviado a la revista *Pattern Anal. Appl.*, 2015.

[MS96] C. M. Ma, M. Sonka, "A Fully Parallel 3D Thinning Algorithm and Its Applications", *Comput. Vis. Image Und.*, vol. 64, núm. 3, págs. 420-433, 1996.

[My98] H. R. Myler, *Fundamentals of Machine Vision*, Tutorial Texts in Optical Engineering, SPIE Publications, 1998.

[PA11] M. Pawlik, N. Augsten, "RTED: A Robust Algorithm for the Tree Edit Distance", *Proceedings of the VLDB Endowment*, vol. 5, núm. 4, págs. 334-345, 2011.

[PB05] P. Bille, "A survey on tree edit distance and related problems", *Theor. Comput. Sci.*, vol. 337, núm. 1-3, págs. 217-329, 2005.

[PK98] P. Klein, "Computing the edit-distance between unrooted ordered trees", *ESA 98 Proceedings of the 6th Annual European Symposium on Algorithms*, Lecture Notes in Computer Science, vol. 1461, págs. 91-102, 1998.

[PK99] K. Palágyi, A. Kuba, "Directional 3D thinning using 8 subiterations", *Discrete Geometry for Computer Imagery*, Lecture Notes in Computer Science, vol. 1568, págs. 325-336, 1999.

[PHB12] A. Pflug, D. Hartung, C. Busch, "Feature extraction from vein images using spatial information and chain codes", *Inform. Security Tech. Rep.*, vol. 17, núm. 1-2, págs. 26-35, 2012.

[RCCT06] H. J. Rea, J. R. Courney, D.E.R. Clark, N.K. Taylor, "Commercial and business issues in the e-sourcing and reuse of mechanical components", *Int. J. Adv. Manuf. Tech.*, vol. 30, núm. 9-10, págs. 952-958, 2006.

[SC14] H. Sánchez-Cruz, H.López-Valdez, F. J. Cuevas, "A new relative chain code in 3D", *Pattern Recogn.*, vol. 47, núm. 2, págs. 769-788, 2014.

[SCB08] H. Sánchez-Cruz, E. Bribiesca, "Study of compression efficiency for three-dimensional discrete curves", *Opt. Eng.*, vol. 47, núm. 7, art. núm. 077206, 2008.

[SDC04] I. Sivignon, F. Dupont, J.M. Chassery, "Decomposition of a three-dimensional discrete object surface into discrete plane pieces", *Algorithmica*, vol. 38, núm. 1, págs. 25-43, 2004.

[SDC05] I. Sivignon, F. Dupont, J. Chassery, "Discrete Surfaces Segmentation into Discrete Planes", *Combinatorial Image Analysis*, Lecture Notes in Computer Science, vol. 3322, págs. 458-473, 2005.

[SNS02] S. Svensson, I. Nyström, G. Sanniti di Baja, "Curve skeletonization of surface-like objects in 3D images guided by voxel classification", *Pattern Recogn. Lett.*, vol. 23, núm. 12, págs. 1419-1426, 2002.

[SP73] S. Papert, "Uses of technology to enhance education", Laboratorio de Inteligencia Artificial, MIT, Reporte Técnico 298, 1973.

[SSP13] J. H. Sossa-Azuela, R. Santiago-Montero, M. Pérez-Cisneros, E. Rubio-Espino, "Computing the Euler Number of a Binary Image Based on a Vertex Codification", *J. Appl. Res. Technol.*, vol. 11, núm. 3, págs. 360-370, 2013.

[TH07] T. C. Hales, "The Jordan Curve Theorem, Formally and Informally", *Am. Math. Mon.*, vol. 114, núm. 10, págs. 882-894, 2007.

[TOU13] R. Turior, D. Onkaev, B. Uyyanonvara, P. Chutinantvarodom, "Quantification and classification of retinal vessel tortuosity", *ScienceAsia*, vol. 39, núm. 3, págs. 265-277, 2013.

[TY09] J. Toriwaki, H. Yoshida, *Fundamentals of Three-Dimensional Digital Image Processing*, Springer, 2009.

[VL66] V. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", *Sov. Phys. Dokl.*, vol. 10, núm. 8, págs. 707-710, 1966.

[WB07] T. Wang, A. Basu, "A note on A Fully Parallel 3D Thinning Algorithm and Its Applications", *Pattern Recogn. Lett.*, vol. 28, núm. 4, págs. 501-506, 2007.

[WZ08] R. Wilson, P. Zhu, "A study of graph spectra for comparing graphs and trees", *Pattern Recogn.*, vol. 41, núm. 9, págs. 2833-2841, 2008.

[ZS89] K. Zhang, D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems", *SIAM J. Comput.*, vol. 18, núm. 6, págs. 1245-1262, 1989.

[ZSM05] J. Zhang, K. Siddiqi, D. Macrini, A. Shokoufandeh, S. Dickinson, "Retrieving Articulated 3-D Models Using Medial Surfaces and Their Graph Spectra", *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Lecture Notes in Computer Science, vol. 3757, págs. 285-300, 2005.