UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

"CÓDIGO CADENA PARA LA DESCRIPCIÓN DE VOLÚMENES"

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS
(COMPUTACIÓN)

P R E S E N T A:

VICTOR JESÚS LÓPEZ SANTOYO

DIRECTOR DE TESIS: DR. ERNESTO BRIBIESCA CORREA

México, D.F. **2009.**

Agradecimientos

Agradezco primeramente a mis padres, Victor y Sofía por su amor, confianza, paciencia y comprensión que me han brindado no sólo en la realización de este trabajo, sino por un trabajo de vida. Gracias por esas palabras de aliento, ese abrazo que reconforta y por siempre creer en mi.

Al Dr. Ernesto Bribiesca, que más que director de tesis ha sido un amigo al que gracias a su guianza y enseñanza este trabajo pudo ser realizado.

Gracias profundamente a mis amigos Juan Luis, Michel, Bernardo, Tiffany, Adrián, Fernando, Abraham, Eduardo y Miguel por su apoyo incondicional.

También agradezco a los Doctores Boris Escalante, Jesús Savage, Fernando Arámbula y Yann Frauel por tomarse un tiempo para revisar este trabajo y brindar valiosos comentarios sobre el mismo que ayudaron a enriquecerlo.

Por último, quisiera agradecer a la Universidad Nacional Autónoma de México por haberme permitido estudiar en sus instalaciones; y al Consejo Nacional de Ciencia y Tecnología por el apoyo económico brindado durante mis estudios.

Índice general

Índice de cuadros

Índice de figuras

Abstract

In the present work a new chain code descriptor for volumes or 3D objects is proposed. For this matter, the medial axis information (*skeleton*) of the object is obtained and subsequently encoded by means of a chain code for orthogonals trees. This new descriptor is invariant under translation and it's invariant under the starting vertex of encoding.

Also, a new dissimilarity measure for simple 3D curves is proposed. This measure can be used along with a graph matching approach in order to compare objects by using only their chain code descriptors. This measure is computed in linear time and it yet produces good results.

Resúmen

En el presente trabajo se propone un nuevo descriptor de código cadena para volúmenes u objetos tridimensionales. Para esto se obtiene la información del eje medio del objeto (*esqueleto*) a representar y se codifica usando un código cadena para árboles ortogonales. Dicho descriptor es invariante ante la traslación e invariante al punto de inicio de codificación.

También se presenta una nueva medida de disimilitud entre curvas simples tridimensionales que podría ser usada junto con alguna técnica de comparación de gráficas para hacer comparación entre objetos en base únicamente a sus respectivos descriptores. Esta medida se calcula en tiempo lineal y produce resultados satisfactorios.

Capítulo 1

Introducción

El gran número de objetos 3D que se pueden encontrar en internet o en bases de datos privadas, gracias al avance en técnicas de modelado, visualización y digitalización de formas 3D, ha hecho que la representación y comparación de objetos 3D haya tomado gran relevancia.

Esto ha llevado al desarrollo de motores de búsqueda experimentales de objetos 3D, tales como el de la Universidad de Princeton[2], el sistema de recuperación de modelos de la Universidad Nacional de Taiwán[1] o el motor de búsqueda Ephesus del Consejo Nacional de Investigación de Canadá[10].

Técnicas de búsqueda mediante el uso de etiquetas de texto pueden ser empleadas, sin embargo, las etiquetas dependen de la persona que las pone, su idioma, su especialización, el uso que da a los modelos, etc. Esto hace que las búsquedas basadas en texto no siempre funcionen para objetos 3D. Métodos basados en propiedades de la forma son así preferidos ya que se cuenta con resultados que prueban su eficacia sobre los basados en texto [17]. También, este tipo de métodos son útiles en la comparación de modelos uno a uno.

Así, un paso fundamental para el proceso de caza de patrones es la obtención de descriptores de forma de los objetos. Dentro del terreno bidimensional existe una amplio trabajo de investigación sobre descriptores de formas[19]. De los métodos bidimensionales podemos resaltar los basados en códigos cadena, lo cuales fueron primeramente propuestos por Freeman[11]. Posteriormente, E. Bribiesca y A. Gúzman logran establecer, mediante sus Números de Forma[5], un código cadena invariante a la rotación, escala, traslación y punto de

inicio de la codificación. Desafortunadamente, los métodos que funcionan en dos dimensiones no siempre se pueden generalizar a tres dimensiones [15].

En cuanto a descriptores en tres dimensiones, Sunder et al.[18] proponen un descriptor mediante el uso de gráficas jerárquicas de esqueleto que codifican información geométrica y topológica de los objetos. Después comparan dos formas mediante una comparación aproximada de sus respectivas gráficas usando un algoritmo voraz que encuentra correspondencia entre la máxima cardinalidad y mínimo peso dentro de una gráfica bipartita.

Iyer et al.[13, 14] usan características globales y gráficas de esqueleto para describir modelos de volumen obtenidos mediante la voxelización de objetos sólidos. El esqueleto es obtenido mediante un algoritmo iterativo de adelgazamiento hasta que al final queda un esqueleto de un voxel de ancho. La correspondencia la realizan mediante un algoritmo que detecta isomorfismo en gráficas y subgráficas mediante el uso de árboles de decisión.

E. Bribiesca[3] propuso que si un objeto tridimensional es envuelto mediante una curva, es posible representar esta curva mediante un código cadena y así representar el volumen. Usando una idea similar se ha logrado obtener más resultados[4]. E. Bribiesca[8] también propuso un código cadena que sirve para describir objetos tridimensionales en forma de árbol.

El presente trabajo tiene por objetivo extender el trabajo expuesto en [8] y presentar un código cadena para volúmenes (objetos tridimensionales) basados en su información del esqueleto.

Capítulo 2

Descriptor para Volúmenes

Los esqueletos de objetos tridimensionales forman naturalmente una estructura de árbol y a su vez, guardan información de la topología del objeto del que fueron extraídos. La propuesta que se presenta en este trabajo es tomar estos *árboles* y codificarlos mediante el código cadena para árboles[8] para así obtener un descriptor del objeto tridimensional.

Para ello primero introducimos una técnica de adelgazamiento de modelos tridimensionales propuesta por Ma y Sonka[7] y extendida después por Wang y Basu [20] para corregir problemas de conectividad.

Dado que el proceso de adelgazamiento produce esqueletos 26-conectados es necesario realizar un procesamiento del mismo para poderlo codificar con el código de cadena para árboles, ya que éste sólo funciona con árboles ortogonales (objetos 6-conectados). Así que un proceso de ortogonalización de estos esqueletos es también descrito.

En la última parte del desarrollo de este trabajo, proponemos una medida de disimilitud para curvas simples basados en sus códigos cadena. Esta aunque menos precisa que la anteriormente propuesta[9], es más rápida de calcular y guarda resultados comparativamente satisfactorios.

2.1. Obtención del Esqueleto

Al realizar la operación de adelgazamiento sobre un modelo se obtiene una representación compacta del mismo, llamada su *esqueleto*.

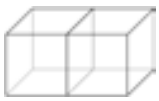
Si consideramos que una *imagen binaria tridimensional* es un mapeo que asigna 1 y 0 a cada punto dentro del espacio, 1 si están dentro del objeto y 0 si pertenecen al fondo; la operación de adelgazamiento iterativamente remueve puntos del objeto y los convierte en puntos del fondo, esto hasta que alguna condición es satisfecha. Es importante recalcar que bajo ninguna circunstancia se debe convertir un punto del fondo en punto del objeto. Al proceso de obtención del esqueleto también se le conoce como *Transformación del Eje Medio*.

2.1.1. Conceptos Básicos

Un **voxel** es la unidad cúbica que compone una imagen u objeto tridimensional, por lo tanto se podría considerar que un voxel es el equivalente tridimensional del pixel bidimensional.

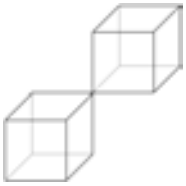
Existen tres formas en las que los voxeles pueden estar conectados entre sí:

1. **Conectados por la Cara**: Cuando dos voxeles tiene una cara en común.



2. **Conectados por la Arista**: Cuando dos voxeles tienen una arista en común.

3. **Conectados por el Vértice**: Cuando dos voxeles tienen un vértice en común.



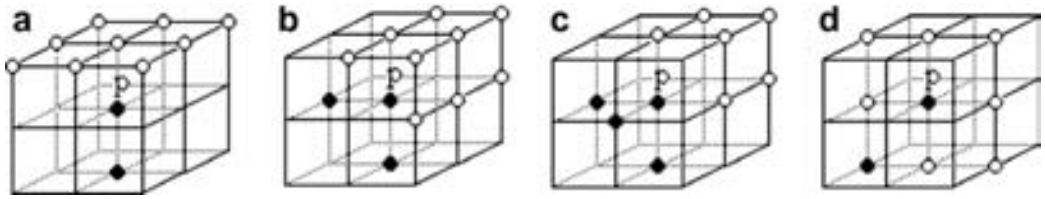


Figura 2.1: Cuatro plantillas base para el algoritmo. Un • representa un punto del objeto y un ∘ representa un punto del fondo. Aquellos que no están marcados son puntos que no importan. *Imagen tomada de [20]*

Si en una imagen tridimensional sólo se permitiera conexiones por cara, desde un elemento p cualquiera sólo se podrían alcanzar a 6 elementos vecinos, por lo tanto a los elementos conectados por cara se les conoce como **6-conectado**. Si bien, ahora se permitiése además de conexiones por cara, conexiones por arista, desde el mismo punto p se podría llegar a 18 elementos vecinos, quedando así un objeto **18-conectado**. Por último, si ahora también permitieramos conexiones por vértice se podría llegar a 26 elementos vecinos, con lo que el objeto quedaría **26-conectado**.

Una característica importante que se busca en el algoritmo de adelgazamiento es que éste preserve la conectividad para objetos tridimensionales. Si esta condición no se cumple provocaría que el proceso de codificación de los esqueletos en códigos cadena no sea posible.

2.1.2. Descripción del Algoritmo

Ma y Sonka[7] propusieron una serie de plantillas predefinidas de eliminación, divididas en 4 clases, A, B, C y D. Si los vecinos de un punto del objeto empatan con alguna de las plantillas, dicho elemento es eliminado (cambiado a punto del fondo). La figura 2.1 muestra las 4 plantillas base del algoritmo. Los puntos marcados con • representan un punto del objeto y aquellos denotados con un ∘ representan a los puntos del fondo. Si algún punto no está marcado significa que no importa y puede representar tanto a un punto del fondo como del objeto.

Por si solas estas plantillas base no son plantillas de eliminación. Ma y Sonka aplican una serie de translaciones a cada plantilla para generar las plantillas de eliminación. Hay 6 plantillas en la Clase A (figura 2.2), 12 en la Clase B (figura 2.3), 8 en la Clase C (figura 2.4) y 12 en la Clase D (figura 2.5). En la Clase D hay que notar que al menos uno de los puntos marcados como □ deben ser puntos del objeto.

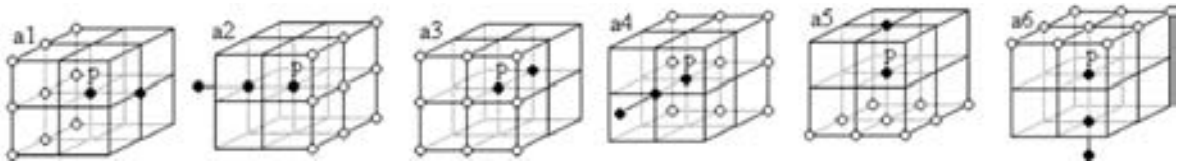


Figura 2.2: Seis plantillas en la Clase A. *Imagen tomada de [20]*

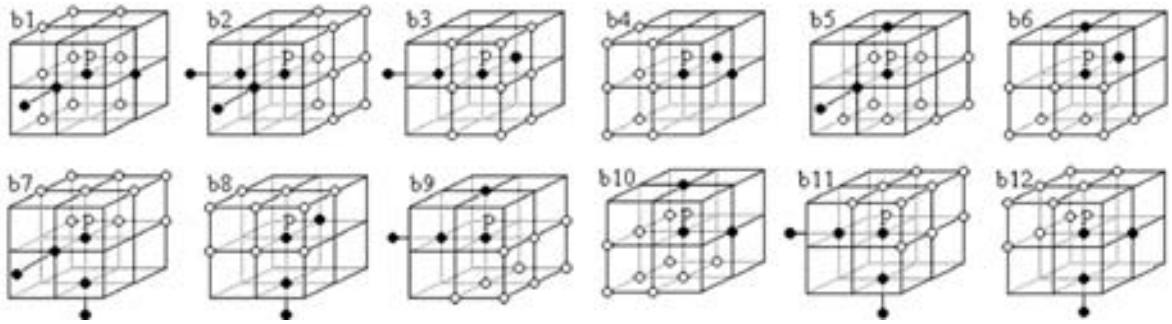


Figura 2.3: Doce plantillas en la Clase B. *Imagen tomada de [20]*

A su vez, Ma y Sonka definieron las siguiente condiciones de preservación:

Sea p un punto del objeto de alguna imagen tridimensional, entonces:

1. p es llamado un punto *final de línea* si p está 26-conectado a exactamente un punto del objeto.

2. p es llamado un punto *casi final de línea* si p está 26-conectado a exactamente dos puntos del objeto, donde dichos puntos son:

   $a)$ ya sea $s(p)$ y $e(p)$, o $s(p)$ y $u(p)$ pero no ambos.

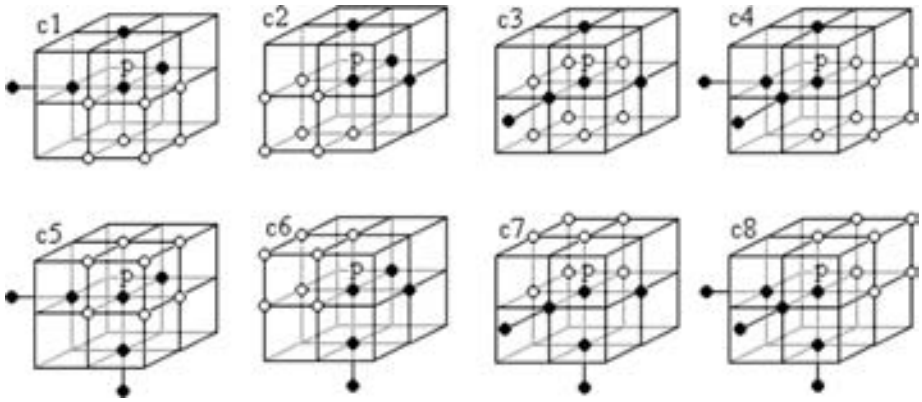


Figura 2.4: Ocho plantillas en la Clase C. *Imagen tomada de [20]*

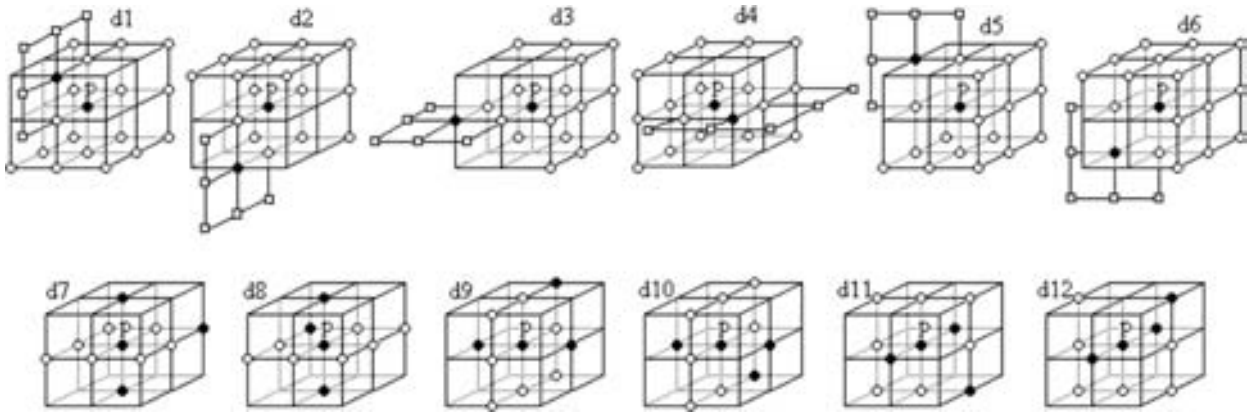


Figura 2.5: Doce plantillas en la Clase D. Donde al menos un punto marcado con □ es un punto del objeto. *Imagen tomada de [20]*

b) ya sea $n(p)$ y $w(p)$, o $u(p)$ y $w(p)$ pero no ambos.

c) ya sea $n(p)$ y $d(p)$, o $e(p)$ y $d(p)$ pero no ambos.

3. p es llamado un punto *cola* si p es un punto final de línea o casi final de línea, de otro modo es llamada un punto *no cola*.

Donde $e(p)$, $w(p)$, $n(p)$, $s(p)$, $u(p)$ y $d(p)$ son los vecinos del este, del oeste, del norte, del sur, de arriba y de abajo de p respectivamente.

En cada iteración, todos los puntos *no cola* que satisfagan al menos una de las plantilla de eliminación en las Clases A, B, C o D, será borrado por el algoritmo 1:

Algoritmo 1 Algoritmo de Ma y Sonka

```
procedure SONKA
    repeat
        Marcar cada punto del objeto que esté 26-conectado a algún punto del fondo.
        repeat
            Simultáneamente eliminar cada punto no cola que satisfaga
            al menos una plantilla de eliminación de las Clases A, B, C o D.
        until ningún punto más pueda ser borrado
        Desmarcar todos los puntos aún marcados pero que no se borraron
    until ningún punto más pueda ser borrado
end procedure
```

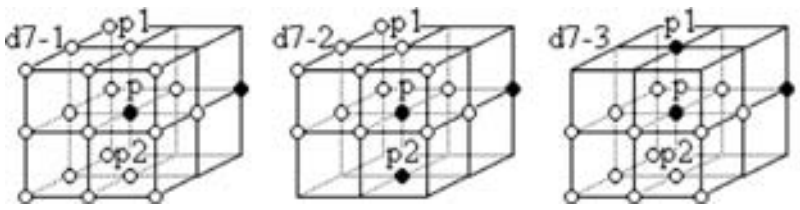


Figura 2.6: Plantillas d7-1, d7-2 y d7-3. *Imagen tomada de [20]*

2.1.3. Problemas de Conectividad y Solución

Wang y Basu encontraron que el método propuesto por Ma y Sonka no siempre preservaba la conectividad en los esqueletos resultantes [20]. Específicamente encontraron que la Clase D de patrones llevaba a problemas de conectividad (Figurar 2.8b).

Para solucionar este problema propusieron modificar cada plantilla de la Clase D y extenderlas en tres nuevas plantillas cada una. Por ejemplo, la plantilla d7 se cambio a 3 plantillas diferentes como se puede ver la figura 2.6.

Del mismo modo las 12 plantillas de la Clase D fueron cambiadas a 36 diferentes plantillas como se muestra en la figura 2.7. Los resultados de está modificación se pueden ver en la figura comparativa 2.8.

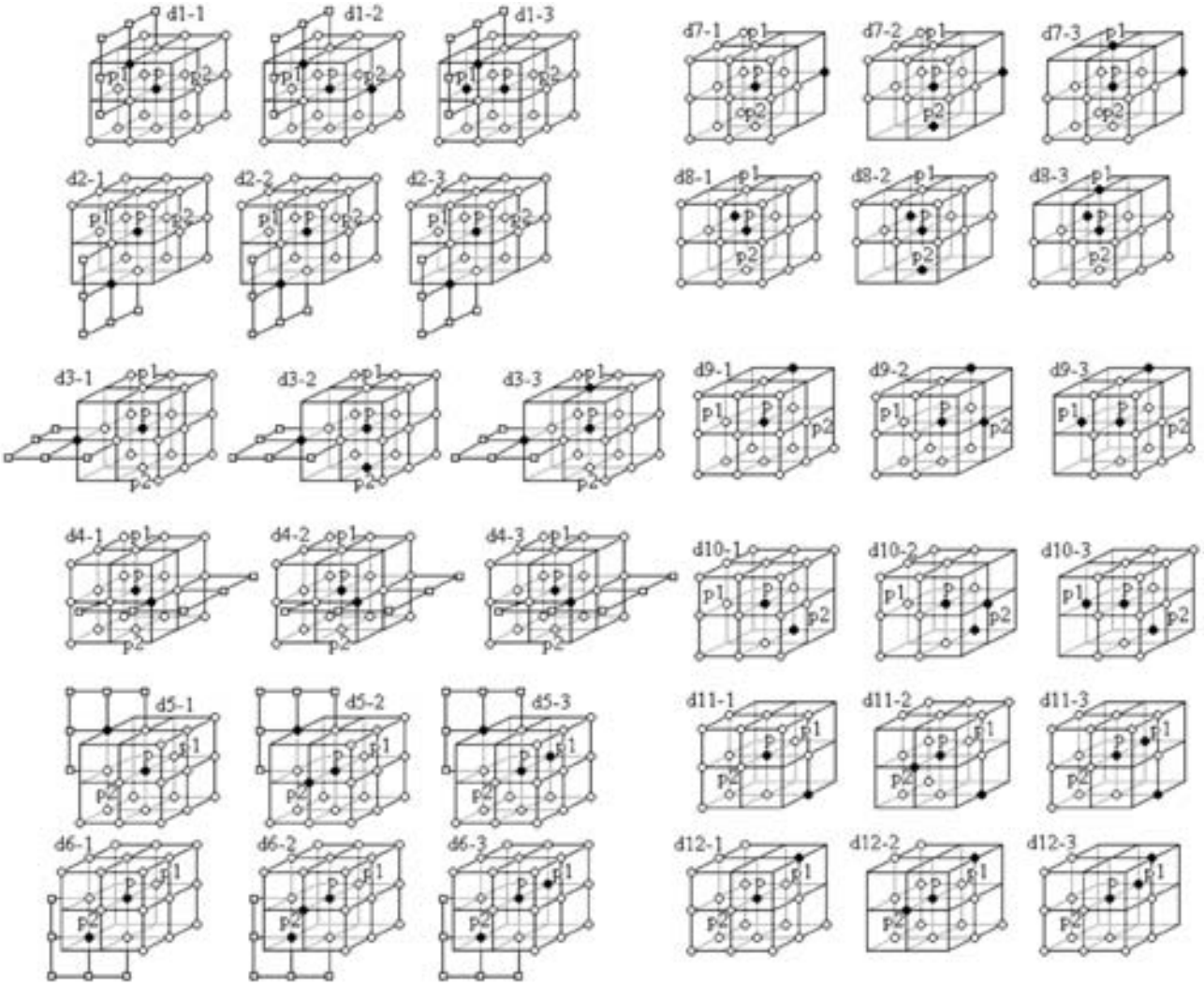


Figura 2.7: Plantillas modificadas de la Clase D. Donde al menos un punto marcado con □ es un punto del objeto. *Imagen tomada de [20]*

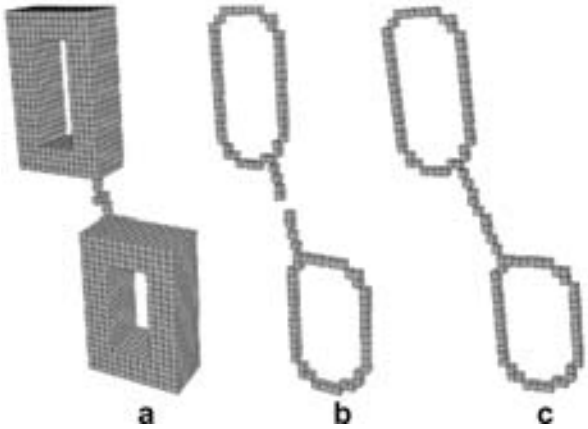


Figura 2.8: (a) Objeto Original, (b) Resultado del algoritmo original y (c) Resultado con las plantillas modificadas. *Imagen tomada de [20]*

2.2. Código Cadena para Árboles

El esqueleto obtenido en el capítulo anterior genera de forma natural una estructura de árbol. E. Bribiesca propuso un código cadena que puede representar estructuras de este tipo [8]. De este modo, al combinar el esqueleto junto al código cadena mencionado, será posible obtener un descriptor del objeto tridimensional original.

El código cadena para árboles tiene las ventajas de conservar tanto la información geométrica como topológica del árbol que representa, características sumamente deseadas para la mejor descripción de objetos tridimensionales. Más aún, los códigos cadena han sido ampliamente estudiados y son usados en diferentes algoritmos de análisis de forma [12].

2.2.1. Elementos del Código Cadena

Una gráfica está compuesta por un conjunto de puntos llamados *vértices* v unidos por *aristas* a. Una gráfica está conectada si existe un camino entre cualquiera dos vértices. Un *árbol* es una gráfica conectada que no contiene ciclos. En un árbol, cualesquiera dos vértices están conectados por un único camino y el número de aristas (ramas) está dado por:

$$a = v - 1$$

El *grado* de un vértice dentro de un árbol, es el número de aristas que hacia él inciden. El método de representación de árboles aquí usado representa árboles de hasta grado 6, dado que representa solamente árboles ortogonales tridimensionales compuestos por segmentos de línea.

Una cadena a es una secuencia ordenada de n elementos y está definida como:

$$a = a_1 a_2 ... a_n = \{a_i : 1 \leq i \leq n\}$$

Las ramas (aristas) de los árboles están compuestos por segmentos de línea. Dos segmentos contiguos de línea definen un cambio de dirección y dos cambios de dirección definen un elemento de la cadena. Por lo tanto, un elemento a_i de la cadena indica el cambio de dirección ortogonal de dos segmentos de línea contiguos de la rama discreta en la posición del elemento.

Figura 2.9: Elementos de la cadena. *Imagen tomada parcialmente de [8]*

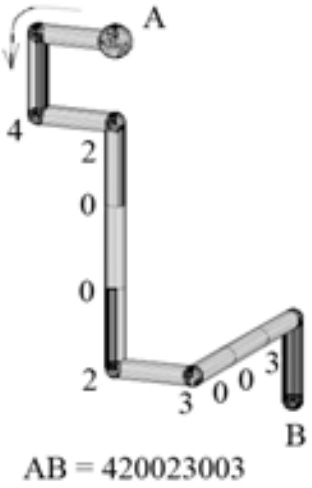


Figura 2.10: Árbol de dos vértices junto con su descriptor. *Imagen tomada parcialmente de [8]*

Los elementos de la cadena para una rama discreta se obtienen al calcular los cambios relativos de dirección ortogonal de los segmentos de línea contiguos que componen a dicha rama. Existem solamente 5 posibles cambios de dirección ortogonal para cualquier rama.

Cada elemento de la cadena etiqueta a un vértice de la rama discreta e indica el cambio de dirección ortogonal sobre dicho vértice. La figura 2.9 muestra los posibles elementos de la cadena.

Si las direcciones b y c (ver el elemento 0 en la figura 2.9) forman un ángulo de referencia y el siguiente elemento sobre el vértice a etiquetar tiene dirección d, la definición formal del elemento con el que se ha de etiquetar está dada por:

$$chain\ element(b,c,d) = \begin{cases} 0 & \text{si } d = c \\ 1 & \text{si } d = b \times c \\ 2 & \text{si } d = b \\ 3 & \text{si } d = -(b \times c) \\ 4 & \text{si } d = -b \end{cases}$$

donde $\times$ denota el producto cruz vectorial en $\Re^3$

La figura 2.10 muestra un ejemplo de un árbol de dos vértices junto con su código cadena computado.

2.2.2. La Inversa de la Cadena

Dentro del campo de programación funcional, las secuencias (como en un código cadena) son llamadas *listas*. Existen dos funciones bien conocidas sobre listas finitas: la concatenación, denotada como $++$; y la *reversa*, que invierte el orden de los elementos de una lista. También, se tienen las propiedades básicas de estas funciones, tales como la asociación, además de:

$$reverse(reverse\ a) = a$$
$$reverse(a++b) = reverse\ b++reverse\ a$$

Una cadena es invertible si el último de sus elementos no es cero. Esto, debido a que para poder viajar en sentido inverso a través del camino descrito por una cadena, ésta debe terminar en un ángulo recto, el cual sería distinto de 0. De manera formal, tenemos:

Dada una cada a, su cadena inversa, $inv\ a$, se define como:

- Si no existe ningún elemento 0 en a, entonces $inv\ a = reverse(a)$
- Si todos los elementos en a, exceptuando el último, son iguales a 0, entonces $inv\ a = a$
- De otro modo, a se puede expresar como una concatenación de dos o más cadenas, $a = a_1 ++ a_2 ++ ... ++ a_n$, donde cada $a_k (k = 1, ..., n)$ pertenece a alguna de las formas de las claúsulas anteriores. Entonces $inv\ a = inv\ a_n ++ ... ++ inv\ a_2 ++ inv\ a_1$

Por ejemplo, la inversa de la cadena mostrada en la figura 2.10 se calcularía del siguiente modo:

$$
\begin{aligned}
inv\ 420023003 &= inv(42 + +002 + +3 + +003) \\
&= inv\ 003 + +inv\ 3 + +inv\ 002 + +inv\ 02 \\
&= 003 + +3 + +002 + +24 \\
&= 003300224
\end{aligned}
$$

2.2.3. Independencia al Vértice de Inicio

Para poder obtener un descriptor que realmente sea único se debe garantizar que siempre se comienza a codificar desde el mismo vértice. Para esto, es necesario *normalizar* el vértice de inicio de codificación.

Si consideramos que cada cadena a tiene un valor base cinco asociado a ella, entonces decimos que el descriptor para la rama que describe a está dada por:

$$max(a, inv\ a)$$

De este modo, tenemos un descriptor *normalizado al vértice de inicio* para las ramas. Por ejemplo, para la cadena mostrada en la figura 2.10, su versión normalizada estaría dada por:

$$
\begin{aligned}
max(420023003, inv\ 420023003) &= max(420023003, 003300224) \\
&= 420023003
\end{aligned}
$$

Para extender esta normalización al árbol completo es necesario encontrar lo que llamaremos el *camino único* dentro del árbol. Para ello consideramos todas las posibles permutaciones entre nodos terminales (caminos hoja a hoja), calculamos sus cadenas (sólo aquellas que sean invertibles) y seleccionamos aquella que tenga el valor base cinco más grande asociado.

La figura 2.11 muestra un árbol compuesto de tres ramas con nodos terminales A, B, y C. El nodo U es el nodo unión donde todas las ramas se encuentran. Todos los posibles caminos hoja a hoja son los siguientes:

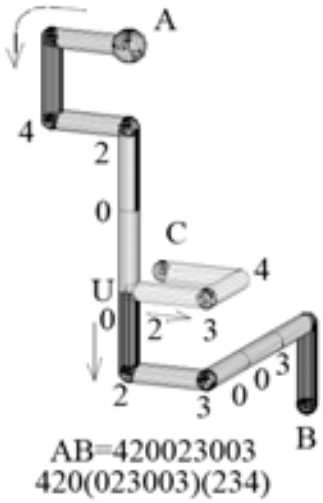


Figura 2.11: Árbol junto con su descriptor en notación de paréntesis. *Imagen tomada parcialmente de [8]*

$AB = 420023003$

$AC = 420234$

$BA = 003300224$

$BC = 0033414$

$CA = 430224$

$CB = 4143003$

Donde el valor base cinco mayor corresponde a la cadena formada por los vértices A y B. Es por esto, que esta cadena corresponde al camino único dentro del árbol y el vértice A es llamado el *origen* o *vértice único de inicio* del árbol.

El número m de todos los posibles caminos de hoja a hoja para cualquier árbol compuesto de v_l vértices hoja está definido por:

$$m = \frac{v_l!}{(v_l!-2)}$$

De donde se puede ver que: $m = v_l(v_l - 1)$. Por lo tanto, el cómputo total del código se realiza en tiempo cuadrático con respecto al tamaño del árbol de entrada.

2.2.4. Manejo de Ciclos y Ortogonalización del Esqueleto

Como ya se ha mencionado, el código aquí descrito funciona para árboles ortogonales, es decir, aquellos donde sus elementos están 6-conectado. Sin embargo, el método de obtención del esqueleto produce esqueletos con elementos 26-conectado. Es por est, que en el prsente trabajo se propone un proceso de adaptación del esqueleto 26-conectado hacia uno 6-conectado.

Para explicar el método aquí propuesto supondremos un código cadena bidimensional cuyos elementos están mostrado en la figura 2.12b. Ahora, supongamos que la figura 2.12a representa una sección de algún esqueleto cualquiera. Las figuras 2.12c y 2.12d representan las dos posibilidades que hay para ortogonalizar el esqueleto. Para el procesamiento de ramas *siempre se toma aquella cadena que tenga el valor base cinco* ***mínimo***. Por lo que, para el ejemplo presentado, la cadena quedaría como 0012.

Ahora, supongamos que tenemos el esqueleto mostrado en la figura 2.13a. Esta gráfica contiene un ciclo, mismo que no es posible procesar con el código cadena. Para ello, proponemos lo siguiente: codificar de forma normal (aplicando la idea del párrafo anterior) y, en el momento que se encuentre una bifurcación (figura 2.13b), continuar el procesamiento por la rama que forme el valor base cinco ***más grande***, al terminar de procesar esta rama, continuar con aquellas ramas que aún no han sido procesadas en orden descendente.

Por ejemplo, el esqueleto de la figura 2.13a fue procesado como se muestra en la figura 2.13d, quedando como cadena resultante 021010101. Es importante notar que *no se trata de romper* los ciclos sino que se propone una forma de *manejarlos* para poder codificarlos mediante el código cadena.

La extensión del método que proponemos a tres dimensiones se hace de forma inmediata. Por lo que es posible trabajar con los esqueletos producidos en el capítulo 2.1.

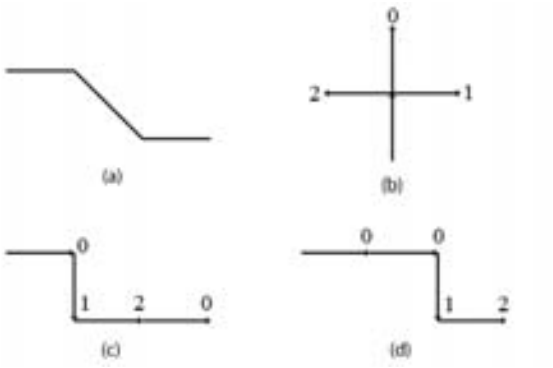


Figura 2.12: (a) Esqueleto Original. (b) Elementos de referencia de la cadena. (c) Primera posibilidad de ortogonalización y (d) Segunda posibilidad de ortogonalización

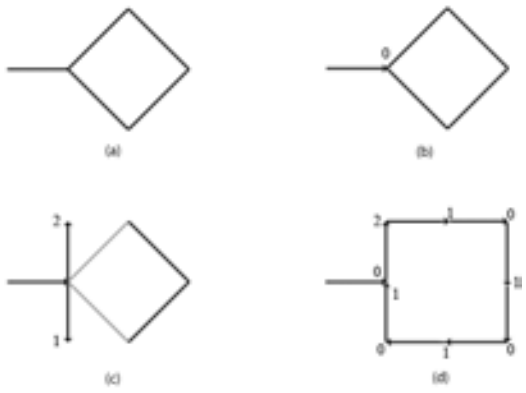


Figura 2.13: (a) Esqueleto original con un ciclo. (b) Bifurcación encontrada. (c) Opciones para continuar el procesamiento y (d) Ciclo procesado y cadena obtenida 021010101

2.2.5. Descriptor de Objetos Tridimensionales

Para el descriptor final usaremos la notación de paréntesis [6], mediante la cual existe una correspondencia entre los árboles y los paréntesis anidados. Es por esto que el descriptor final estará dado en estos términos.

Tomando el árbol mostrado en la figura 2.11, tenemos que su vértice único de inicio es el etiquetado con la letra A. Comenzando a codificar normalmente a partir del vértice A obtenemos los tres primeros elementos de la cadena, 420. Al llegar a la unión en el vértice U tenemos que existen dos posibles caminos representados por los elementos 0 y 2. Siempre seleccionamos las direcciones de acuerdo a su *orden númerico*. Por lo que el camino cuyo primer elemento está representado por 0 es escogido, obteniendo 420(023003). Después regresamos al vértice U y calculamos la cadena de la siguiente rama, quedando como cadena final 420(023003)(234).

En terminos generales decimos que, dado el vértice único de inicio A obtenido mediante el camino único a de un árbol T que representa el esqueleto de un volúmen (objeto tridimensional) O. El ***descriptor único del volúmen*** está dado por el cálculo de los elementos de la cadena (tomando en cuenta las consideraciones presentadas en la sección 2.2.4) usando la notación de paréntesis.

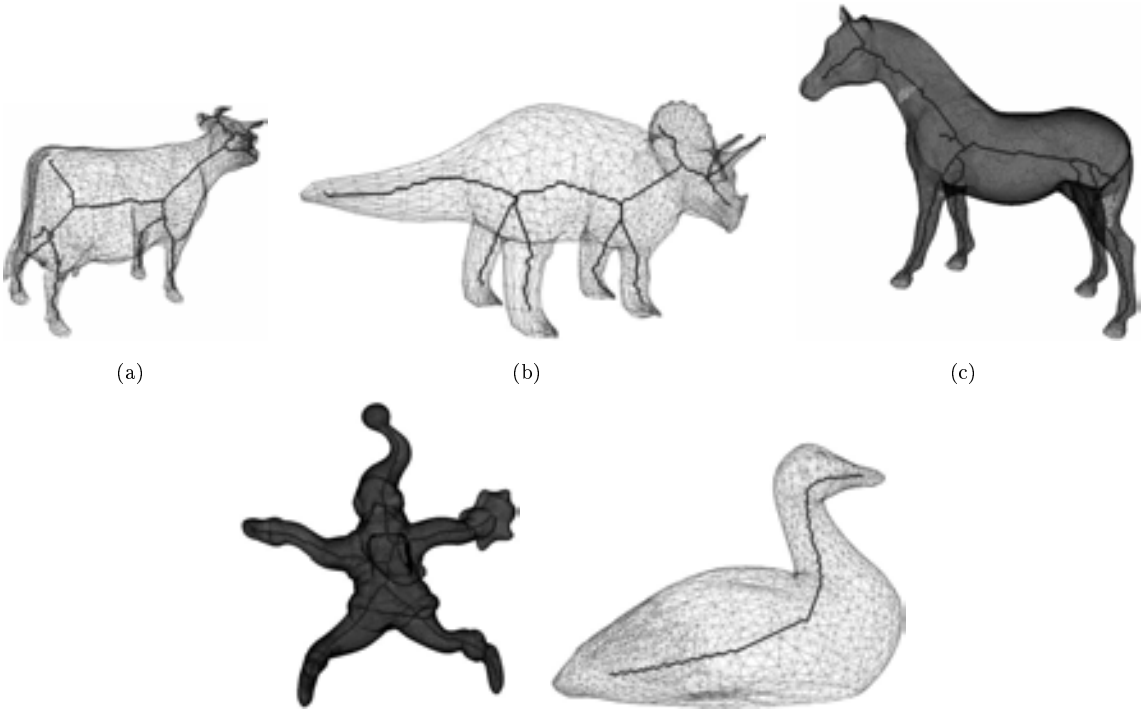
2.3. Resultados

Para probar el método aquí propuesto usamos inicialmente 5 modelos mostrados en la figura 2.14. Es importante notar que estos modelos están en una estructura de mallado por lo que un preprocesamiento de voxelización fue requerido previo a la obtención del esqueleto.

En las figuras 2.15, 2.16, 2.17, 2.18 y 2.19 se muestran los esqueletos de los modelos, la representación ortogonalizada de los mismos y el código cadena que describe a cada modelo en particular.

Posteriormente obtuvimos los descriptores de varios modelos de libre acceso en internet, proporcionados por la Universidad de Princeton[2]. Algunos de los resultados obtenidos se muestran de la figura 2.20 a la 2.27.

Otro resultado importante es la publicación de una parte de esta investigación en la Siggraph Asia 08 como poster [16]. En dicha publicación se presentan las ideas base para esta tesis así como ideas para su ampliación como trabajo futuro.



(a) (b) (c)

(d) (e)

Figura 2.14: Modelos con sus respectivos esqueletos superpuestos. *Imagenes tomadas de [16]*

(a) (b)

212332200022000000000022022(42022022022022022022022222222222220222222202222222222(2(24(4)(200000000000220022000004200000000000000042000000(42222222222222222(22(4(44233333222212333320233322223300333000220323202200221 20)(22222234333323333 3))(24232222222222222233303332232012003200420223334242 0)))(22202200222222222222222222222222242)))(00420022(32333333233322223303233300220001200012032320120321 2)(233333233333333433222222123200000320000022002212032))))(2043333312(32233304232434332 2(43322233302200)(243332(4222)(2333233)))(123332424232233120323 33(33(443(444)(333033322(4222)(3332333)))(33321232320))))

Figura 2.15: Vaca junto con su descriptor. (a) Esqueleto de la vaca y (b) esqueleto ortogonalizado

(a) (b)

22202232042322333223332233333(44333333333333223332220220222232)(2(420022002222(42220220220220220220220220022(2(24(4)(22202200220022002200220042002200000022(2(333322222222222222222220120321203212320223 20)(233222222222222222202222433120333003330220120)(24(4)(22222002200220002200000000420002200022000000000000000042)))))(000(32(4(4)(222222222222222220333022320423242322 3222))(22222222222222202201232023304202203333333)))))(22222120233322002202222232))(24(4)(22321223332222222)))

Figura 2.16: Triceratops junto con su descriptor. (a) Esqueleto del triceratops y (b) esqueleto ortogonalizado

(a) (b)

022002200000042022002200333124330022022012320220004331222200233430323212322223333333 3(433333333333332222233322333333333033332233343333333333001223332222424222222220220023332002202232120022002200220000042022022)(33333333333022002200420000042120000000000000000000032(32200222222222222222222222222332 2(2(423)(2(44)(233322333333333223312433330333333333332233322222233433333(34303333232420)(03222123223303333212)))))(122000000(233(41(3)(2434333332222222123200002200320120320002202330320221200032000423242 32))(22223332233333322000220000000032000000012003203212120320022022023 3))))

Figura 2.17: Caballo junto con su descriptor. (a) Esqueleto del caballo y (b) esqueleto ortogonalizado

(a) (b)

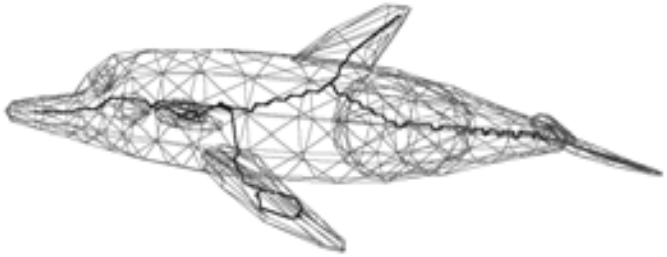
202222223332233300321203330320022220220222202222222222223332333222233333333333333333333333333322333333330333322
33333333333333(34333332233322222(33200121232333222222220220000002232123212333220223332233342422233322222321222022
022002203200023303 2)(2333333333333333(33232423233300000000001200220124203212003330033302232122233322333223332222
2222333)(12222232220220022002200220022012032002232012433324233333332002202201232002332222023303333333333223333 3))
)(012023333333303333333333333322223332222223332123201222320123330223332232123332222222222222222222221232023303333213
23330032(12(44)(333333)))

Figura 2.18: Santa junto con su descriptor. (a) Esqueleto de Santa y (b) esqueleto ortogonalizado

(a) (b)

200220124330023304201201203203201200002200022000220002200000000000000000000022422222222222333222223331203200
220000002330022321203200333022320122233033322223332222222222222202222023 3

Figura 2.19: Pato junto con su descriptor. (a) Esqueleto del pato y (b) esqueleto ortogonalizado



22222222222220222202222(22(4(4)(22222222222022220))(200042433042323203243304200042420042420042420000000
042042000000420420022002200220022002222(42220022002200220033312 00(42(4(424(42(44232333222202233312042 4
2333(22(4(4)(22222233122222204222222433322222))))(24212222222023332002232 22(42220323222222(42(44)(22223
432222222))))(000000000124200000422200233000420))))))(24202202222222222222222)))

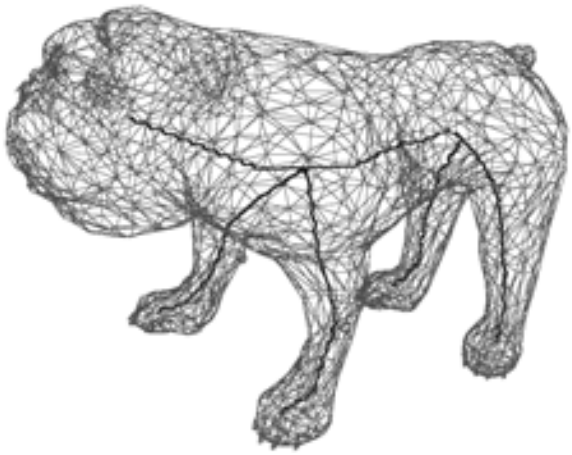
Figura 2.20: Delfin junto con su descriptor

3333(432222243)(2434433333333333322333343322222322333333333333330(42(41(441(4440(44233332333333333333333
332233(3223332233223300333)(232222333223322233320233))(343242000120042320233000(42(42(444(444232422222333
32222333222222222223333333333333322333222233320123212022320120000000)(2))(023332000000220220222 2(2(24(4)
(22202222333223333330(333333332233322333333333333333333(33(443(444)(0423200000))(233322222222222202333200
)))(233333333333332233333333332233333333333322333322033322333220)))(220222222022000420022(32(4(4)(22233322
22333223330333223330000032000042000000000000000000000000042000042000042012))(233322223332023303333330033 3
042120012000042000000000000000000000000042000042000042042000000000)))))))(04232))))))

Figura 2.21: Conejo junto con su descriptor

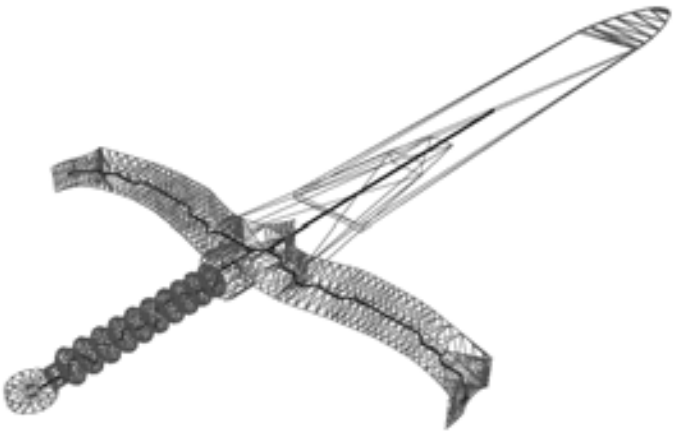
333222120320233003212000004200022000220033302233322222000120220120022000(22(42(444(444200000000000000000
03243300032(32222222220220022002200000220000022000002212003200000000002200000000000023300433323332333333
30022)(12222222220220022002200000012433000022000003330000000032000121200003200000000320042333223303200 2
2))(42(4440(44200000000001202222002200022033(43322(333333)(233333))(332223333))(00420022000320012433 0
32222333223201202200022324200320420003202330333033303 2(23(443(444)(2201223212))(2232233)))))(2))))

Figura 2.22: Figura humana junto con su descriptor



100220022222333022321200220424232003212022222222222223332222333222233322222333222(333(3000000000022002200
220022002200220022002200220022002200220)(243(423(4(4122233322223332222333222222222222222222220022000124330120022033302232122220221203 2)(2344100000000000002200220220022002200221242004200032002200224200233(43333333333333223
33321222220032120032120022000333003201203200000120233022333222)(333333333333332222333233323322022032012032002200120321200022000320033303212322 33))(0)))))

Figura 2.23: Perro junto con su descriptor

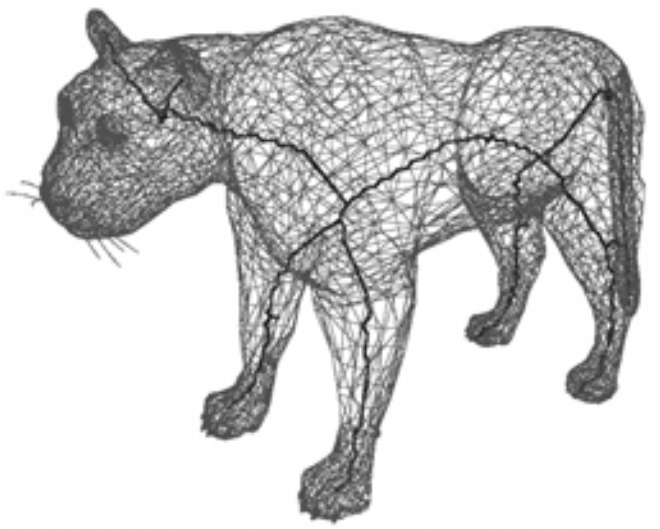


22220220022000000004200022042002 20(22(4(4)(22(2(24(4)(2000221242032000433420324204243342324334200))(00
0022420002200000000420002202222 2))))(200000000000000000000000000000000000000000000000))

Figura 2.24: Daga junto con su descriptor

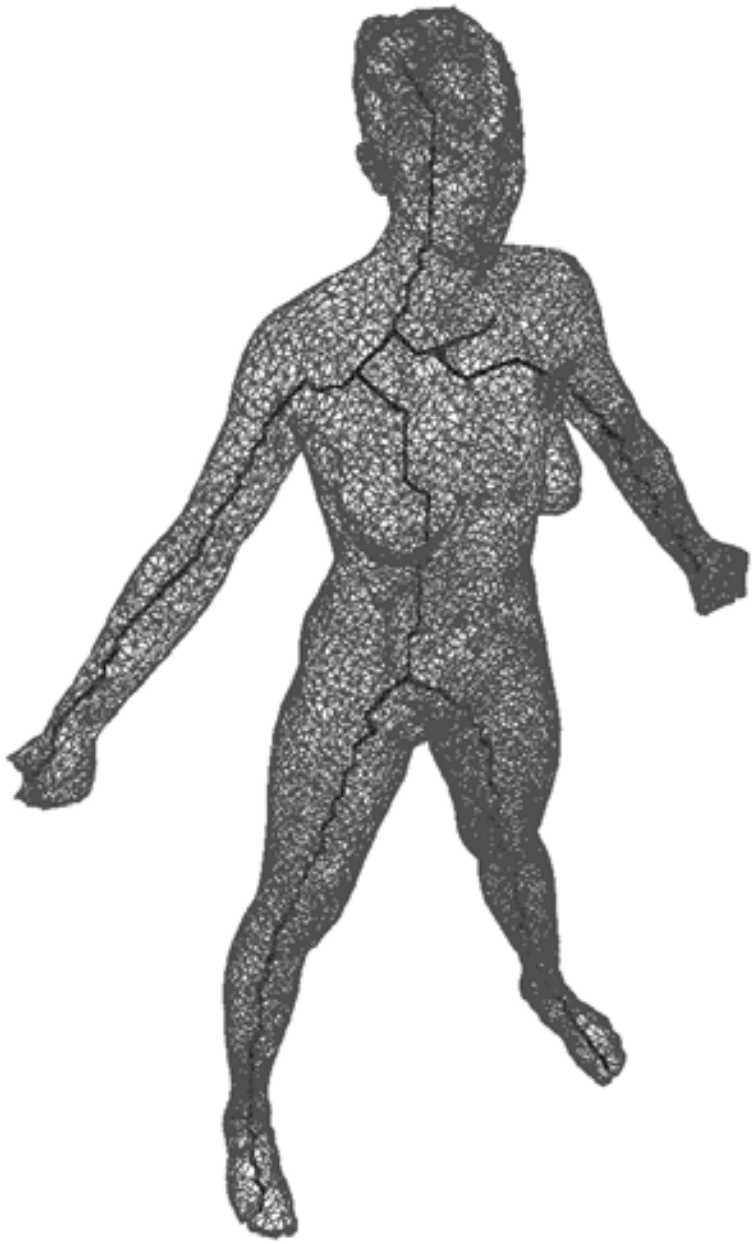
1233332(33304242333333332220120)(2422123242333323(22(444(444)(333333))(33321212(42220120320420000032000000000001200320000000000220000000032002242423212333222222330333333124333333330333333220223330223332233322333222222222(22(44)(3332220(22(44)(333022022(22(44)(33301243332042042003200233032002212320222222222222222022002202201232000220000001200000120003212022002200220220333022022222033322222220333222221203200022120002200000000012000022000022001232022002202222222202222330222222022022002200000000120000000000003200003200000000000000220000000000000000004204334232000320012000000000000000320000000))))))(22333))))

Figura 2.25: Cobra junto con su descriptor



023332232001200000000003200022120320222222222222223332(24233322222222222220023300320002232(42120042004232320123212222)(33))(012(42222222222222222222220022002200022002202200333O(423(14(3)(233322333223332222))(1233322333222222)))(22222002200220022002200220000000004242420022002200220233333(423333(343(41(3)(23022222222222222222042(4242(2442)(022222220022002202202200220022022022022022022022022002202200220220))))(33222222223333212322212322202333202333(3(24(4)(20042002200220012424201202222200222330))(12)))))(333333332233333223212223333222222(433120333000220022002200000042023332200222220)(242433))))

Figura 2.26: Puma junto con su descriptor



123333223333322333033303333332233322333220333033333303330333033333333324232333(33332233(33330033312033
300000042002233333333)(122232(22(41(444(444232222333)(31422333322222222333333333333332233322330(32(44
)(233003330322330321203223303333032220))))))))(12222233320000002223304200042023300320420003200000(42233
33332222330033300333000220002232000423233302202200223202330022000003200022000000223330(22(44)(223322223
33223332233)))(22222333323342220123201232002330003202203212120433002202212003200022000012003330000220
22022(22(4(4)(33333333))(23330333032120))))

Figura 2.27: Mujer junto con su descriptor

Capítulo 3

Medida de Disimilitud entre Curvas Simples

Ya teniendo el descriptor de dos objetos tridimensionales cualesquiera, un importante reto y pregunta inmediata, es saber que tanto se parece el uno al otro. Es por esto que la medida aquí expuesta es un primer acercamiento a saber que tan desiguales son dos objetos en base a sus respectivos códigos cadena. Está medida sólo funciona para curvas simples, es decir, sin ramificaciones.

Si es posible transformar una curva en otra entonces sería posible saber que tan similar son entre ellas obteniendo una medida de la energía requerida en la transformación. La idea principal de nuestro método es ir recorriendo ambas curvas a la misma velocidad al tiempo que se hacen operaciones de *alargar* o *torcer* cuando se encuentran diferencias entre ellas. Al final del recorrido tendríamos dos curvas exactamente iguales y mediante la medición de los alargamientos y torsiones podremos dar una medida de similitud.

Para cumplir este objetivo es necesario definir las operaciones de *alargar* y *torcer* de forma precisa cuando uno se refiere a códigos cadena, así como también el proceso para contabilizar el número necesario de cada operación para transformar las curvas.

3.1. Operaciones de Torsión

Torcer es la operación mediante la cual un elemento de la cadena es intercambiado por otro. Esto es posible debido a que cada elemento de la cadena tiene codificada información de dirección sobre un punto determinado, lo que significa que cambiando solamente un elemento el resto de la cadena permanecerá inalterada. Es decir que estamos *torciendo* la curva sobre un punto.

Si queremos calcular el número de *torsiones* es necesario buscar elementos comunes en ambas cadenas con lo que será posible cambiar aquellos que son diferentes. Por ejemplo, sea S y P dos códigos cadena definidos como:

$$S = 21\mathbf{23}\mathbf{4}3$$
$$P = 213123$$

En este caso, 3 *torsiones* son necesarias (elementos en negrita en S) para convertir a P en S, lo que significa que hay que cambiar los elementos en negritas por aquellos que están en la misma posición en P. En general, elementos comunes son aquellos que aparecen en la misma posición en ambas curvas. Así que para este caso en particular:

$$b(S, P) = 3$$

Donde $b(S, P)$ denota el número de torsiones necesarias para convertir la curva más pequeña en la mas grande[1].

3.2. Operaciones de Alargar

Alargar es la operación mediante la cual un elemento de la cadena más larga es insertado en la más pequeña. La inserción se puede hacer en cualquier punto de la cadena. Por ejemplo, sean S y P dos códigos cadena definidos como sigue:

$$S = 21122$$
$$P = \underline{21}\mathbf{22}\mathbf{1}\underline{122}$$

[1]Se dice que la curva más pequeña es la que tiene el número asociado base 5 más pequeño

Aquí tenemos que insertar 3 elementos en S (números en negrita en P) para convertirla en P. Elementos subrayados en P indican aquellos elementos que son iguales en ambas curvas y por lo tanto no llevan ningún tipo de procesamiento.

Es fácil ver que se necesitan insertar tantos elementos como la diferencia en longitudes entre las dos curvas ya que queremos que al final sean iguales. Por eso, el número de alargamientos necesarios se define como:

$$s(S, P) = |L(S) - L(P)|$$

3.3. Convirtiendo Curvas

El método es ilustrado en la figura 3.1. Se desea convertir la curva mostrada en 3.1a en la curva mostrada en 3.1b dado que la primera es la curva más pequeña. Números en cursiva representan los elementos que están siendo actualmente procesados. Elementos subrayados representan el elemento resultante de la operación descrita en la información de la figura.

El procedimiento en pseudocódigo está descrito en algoritmo 2. En la línea 2.2 $N(S)$ y $N(P)$ representa los valores base 5 asociados a S y P respectivamente. Es en esta línea donde se garantiza que la cadena más pequeña se convierta en la mas grande. De las líneas 2.9 a 2.19 es donde se recorrer todos los elementos de la cadena mas grande.

En caso de que se encuentre una diferencia (línea 2.10) se alarga o se tuerce la cadena S. Dado que las operaciones de alargamientos tienen prioridad sobre las operaciones de torsión, primero se verifica que las longitudes de las cadenas sean diferentes para poder hacer un alargamiento en S (línea 2.11) y, si no es posible, se procede a hacer una torsión (línea 2.14). Los contadores s y b sirven para saber el número de operaciones realizadas de alargamiento y torsión respectivamente.

La operación `Insertar` referida la línea 2.12, inserta el elemento $P[i]$ dentro de la cadena S en la posición i. Esta operación no es definida ya que, como se vera más adelante, puede ser *simulada* agregando un contador más que itere sobre la cadena S.

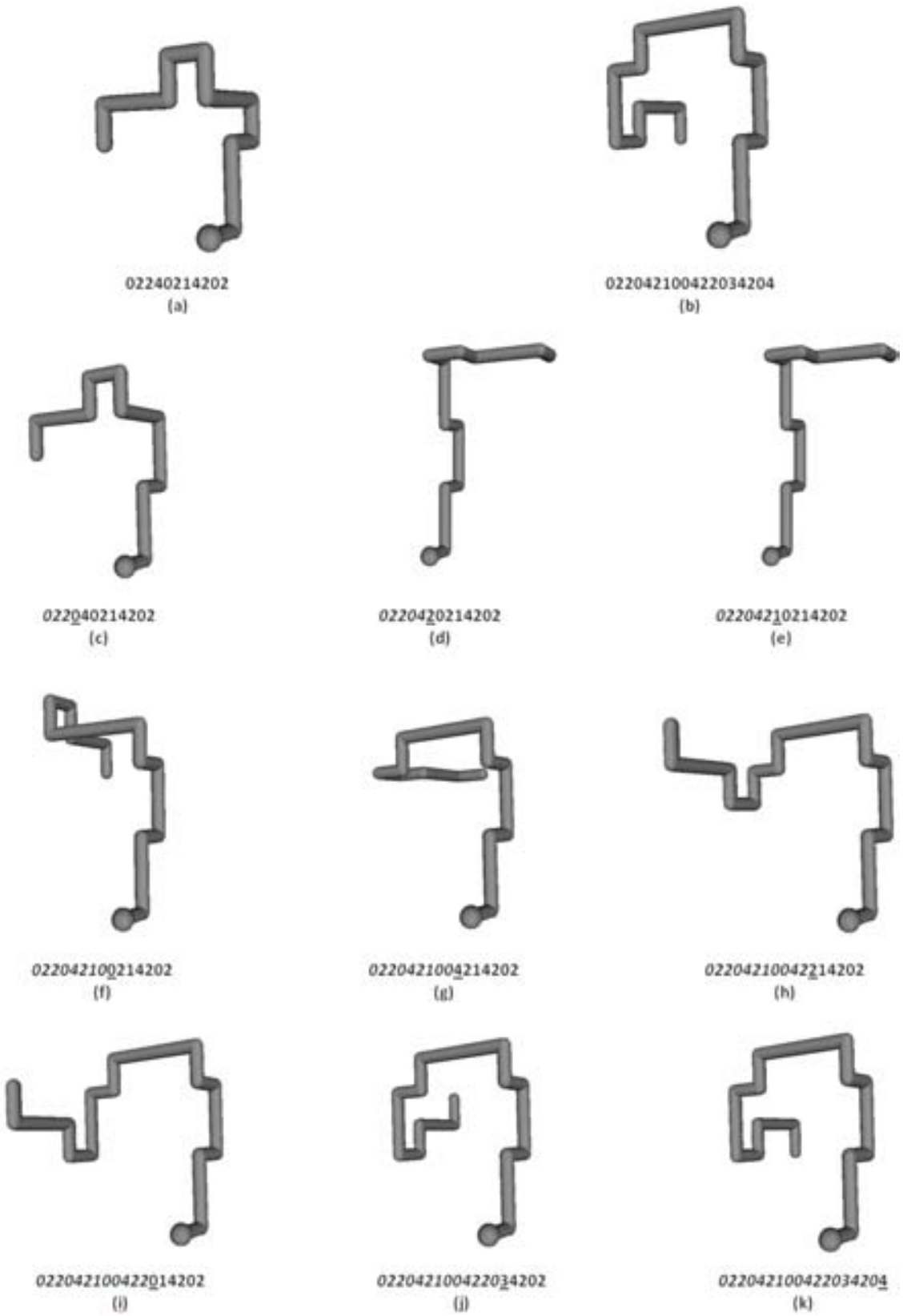


Figura 3.1: Convirtiendo una curva en otra, paso por paso. (a) y (b) son las curvas originales. (c), (d), (e), (f), (g) e (h) Operaciones de Alargar. (i), (j) y (k) Operaciones de Torsión. (k) Curva completamente convertida.

Algoritmo 2 Convertiendo una Curva en Otra

```
1:  procedure CONVERTIR(S, P)
2:      if N(S) > N(P) then
3:          T ← S
4:          S ← P
5:          P ← T
6:      end if
7:      s ← 0
8:      b ← 0
9:      for i ← 1 to L(P) do
10:         if S[i] ≠ P[i] then
11:             if L(S) ≠ L(P) then
12:                 INSERTAR(P[i], S, i)
13:                 s ← s + 1
14:             else
15:                 S[i] ← P[i]
16:                 b ← b + 1
17:             end if
18:         end if
19:     end for
20: end procedure
```

3.4. Cuantificando Diferencias

Si se suma el número de operaciones de torsión y de alargamiento, entonces se tendría una primera idea de que tanto se parecen ambas curva. Esto se representa como sigue:

$$d(S, P) = s(S, P) + b(S, P)$$

Donde S y P son dos cadenas, $s(S, P)$ denota el número de alargamientos necesario y $b(S, P)$ denota el número de operaciones de torsión.

3.5. Medida de Disimilitud

Dado que las medidas de disimilitud son más intuitivas cuando están se encuentran en un rango de 0 y 1, es necesario proponer una forma de normalizar nuestra cuantificación dentro de este rango.

Sean $L(S)$ y $L(P)$ las longitudes de las cadenas S y P respectivamente y supóngase qué P es la cadena más pequeña. En el peor caso posible se tendría que $L(P) = 0$ en cuyo caso

teóricamente se tendría que hacer $L(S)$ alargamientos y $L(P)$ torsiones, esto es:

$$d(S,P) = L(S) + L(P)$$

Y, dado que el número de alargamientos es la diferencia entre las longitudes de las cadenas entonces tenemos que:

$$d(S,P) = (L(S) - L(P)) + L(P)$$
$$d(S,P) = L(S)$$

Con lo que se puede ver que a lo más se necesitan $L(S)$ operaciones (ya sea de alargamiento o torsión) para transformar P en S. Luego entonces, la medida final acotada en el rango de 0 a 1 queda de la siguiente forma:

$$D(S,P) = \frac{s(S,P)+b(S,P)}{L(S)}$$

Donde $D(S,P)$ es nuestra medida final de disimilitud.

3.6. Algoritmo de Cálculo de la Medida de Disimilitud

El algoritmo 2 descrito en la sección 3.3 puede ser modificado para calcular la medida de similitud. El algoritmo 3 muestra dicha modificación.

Es importante notar que la operación `Insertar` ha sido eliminada. Esta operación puede ser *simulada* con el uso de dos contadores que iteren sobre los elementos de S y P separadamente. Cuando se identifica que una operación de inserción es necesaria, el contador que itera sobre S se detiene mientrás qué el que itera sobre P se incrementa. De este modo logramos recorrer del mismo modo ambas curvas que si hubieramos hecho uso de la operación `Insertar`.

Viendo el ciclo principal de conversición de curvas (líneas 3.12 a 3.28) es posible notar que a lo más se necesitan $L(P)$ comparaciones para realizar todo el proceso. Por este motivo el algoritmo se ejecuta en tiempo **lineal**.

Algoritmo 3 Calculando la Medida de Disimilitud entre Curvas

```
1: procedure CALCULAR(result, S, P)
2:     if N(S) > N(P) then
3:         T ← S
4:         S ← P
5:         P ← T
6:     end if
7:     s ← L(P) − L(S)
8:     b ← 0
9:     c ← 0
10:    i ← 1
11:    j ← 1
12:    while i ≤ L(P) do                    ▷ Proceso de conversión modificado
13:        if S[j] = P[i] then
14:            if j ≤ L(S) then
15:                j ← j + 1
16:            else
17:                exit while
18:            end if
19:        else
20:            if c = s then
21:                b ← b + 1
22:                j ← j + 1
23:            else
24:                c ← c + 1                ▷ Simulamos la inserción
25:            end if
26:        end if
27:        i ← i + 1
28:    end while
29:    result ← (s + b) ÷ L(S)
30: end procedure
```

3.7. Resultados

Una medida de disimilitud para curvas simples ya ha sido propuesta con anterioridad[9], así que para validar los resultados del método aquí propuesto, se ha decidido comparar ambos métodos con las curvas descritas en [9] (figura 3.2 y sus versiones digitalizadas en 3.3):

Cuadro 3.1: Resultados obtenidos en [9]

	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

Cuadro 3.2: Resultados obtenidos usando el método descrito en las curvas presentadas en [9]

	A	B	C	D	E	F	G	H	I	J
A	0	0.3846	0.519	0.8684	0.8852	0.9474	0.3091	0.5263	0.5424	0.4359
B		0	0.5063	0.8718	0.8525	1	0.5636	0.6316	0.661	0.2308
C			0	0.9367	0.8101	0.9494	0.5063	0.6835	0.6835	0.519
D				0	0.918	0.8095	0.9091	0.9123	0.9153	0.8718
E					0	0.9344	0.8525	0.8525	0.8033	0.8525
F						0	0.7091	0.9298	0.9322	0.9744
G							0	0.7544	0.7797	0.6
H								0	0.0508	0.7018
I									0	0.7288
J										0

El método aquí propuesto es más rápido de calcular que el anteriormente propuesto[9] y, como se puede observar en las tablas, produce resultados que guardan coherencia comparativa. Esto se puede ver con mayor claridad en las gráficas comparativas de las curvas mostradas en la figura 3.4.

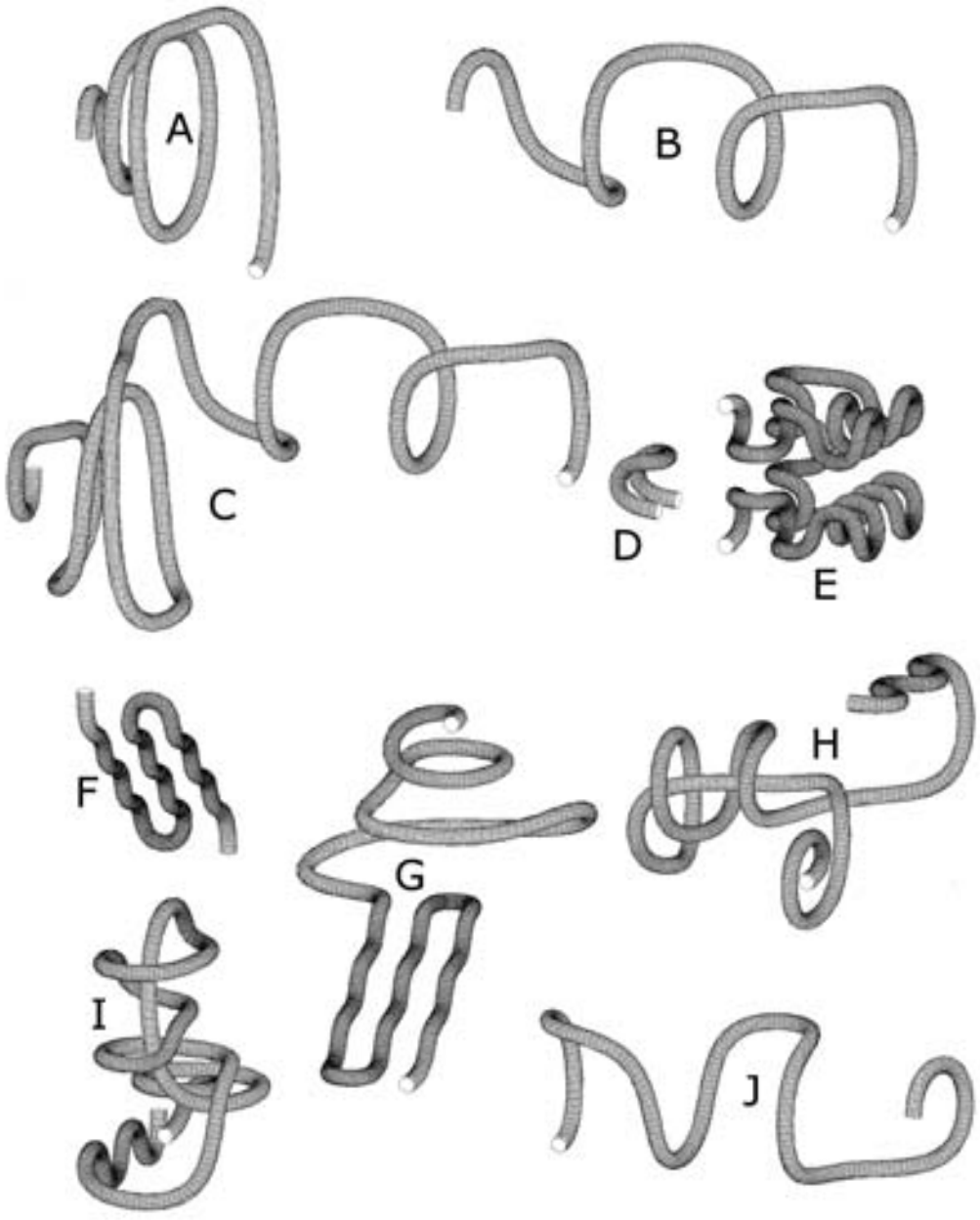


Figura 3.2: Curvas ejemplo presentadas en [9]. A-J

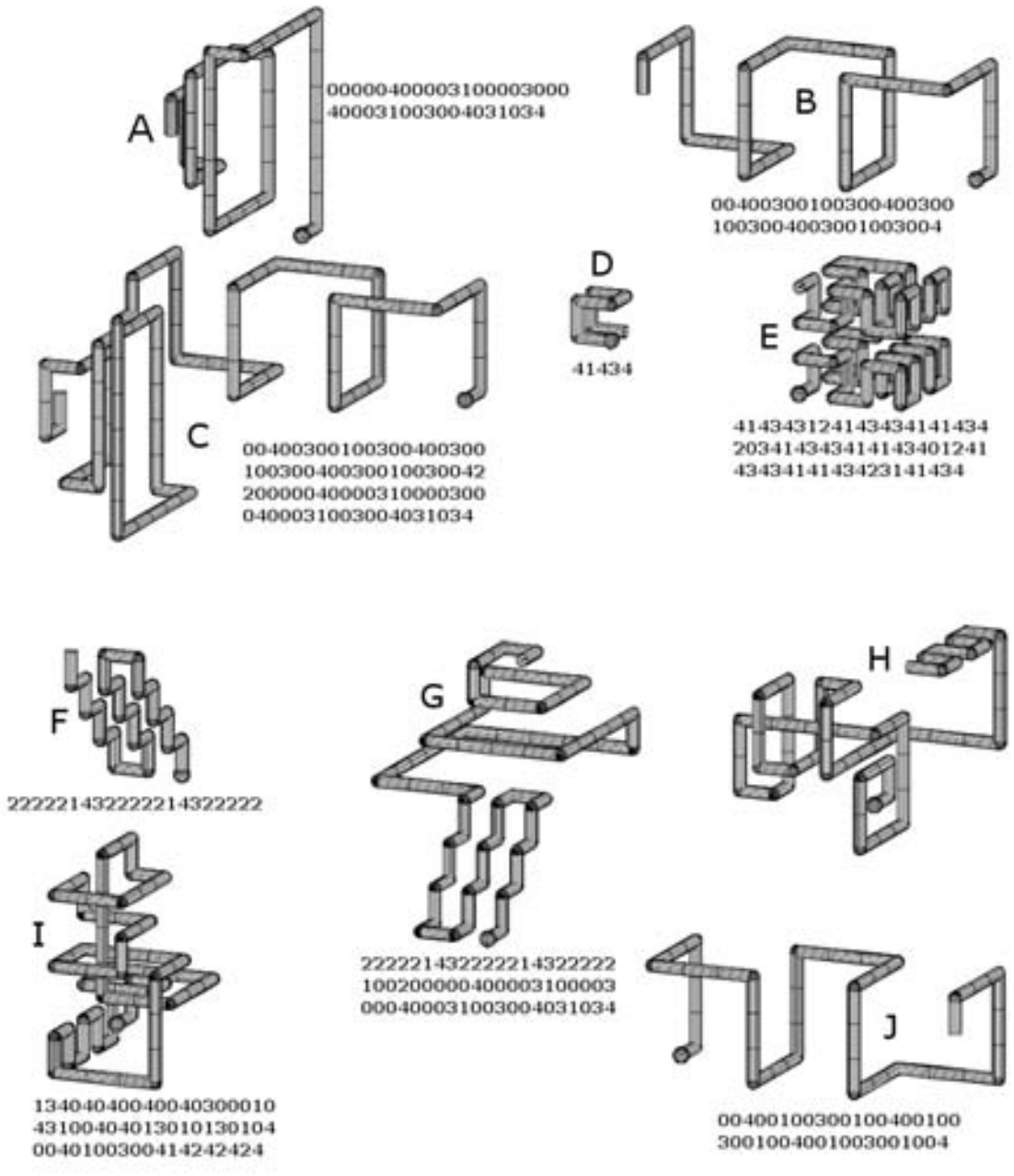


Figura 3.3: Curvas digitalizadas ejemplo presentadas en [9]. A-J

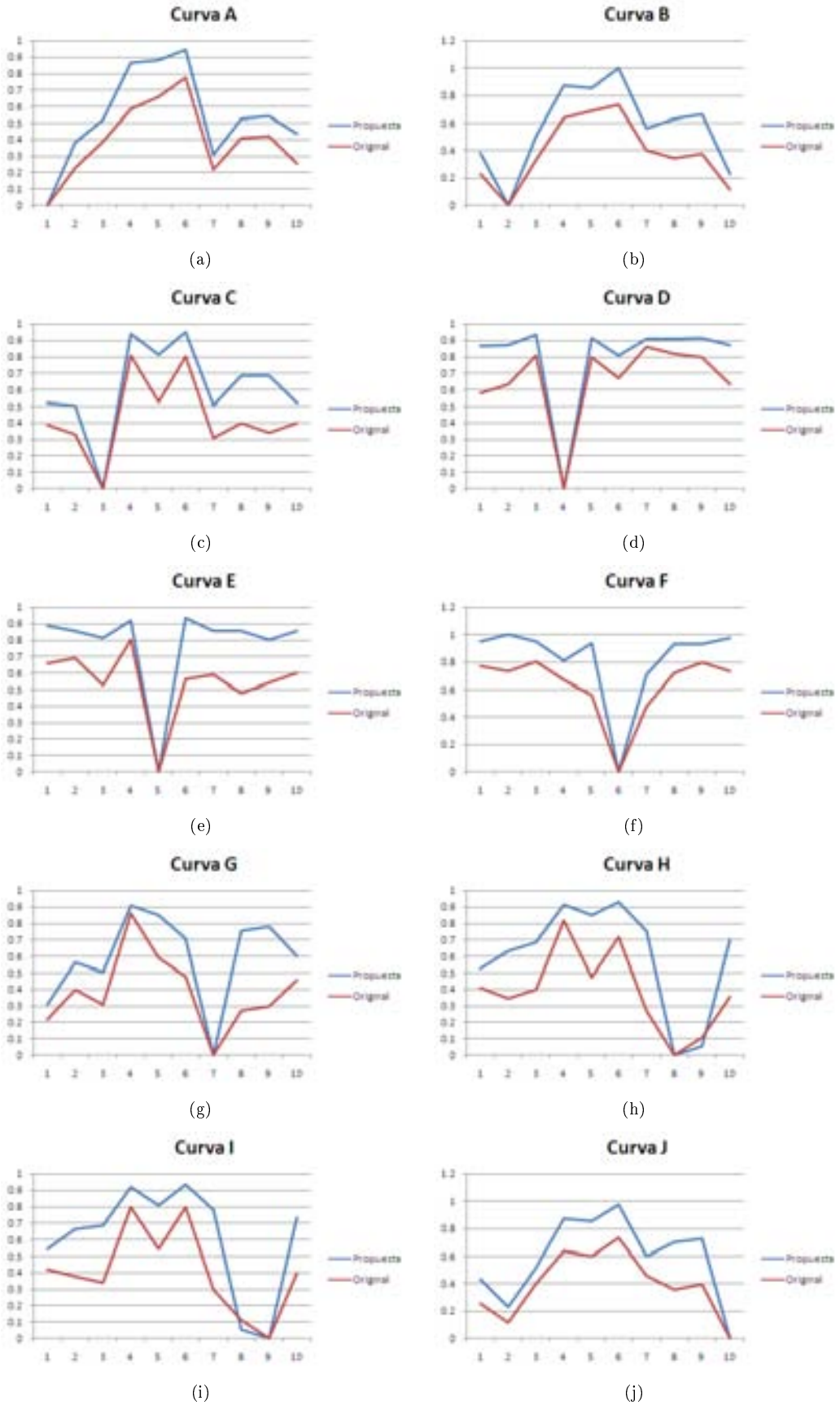


Figura 3.4: Gráficas comparativas entre los resultados obtenidos por el método propuesto y el descrito en [9]. A-J

Capítulo 4

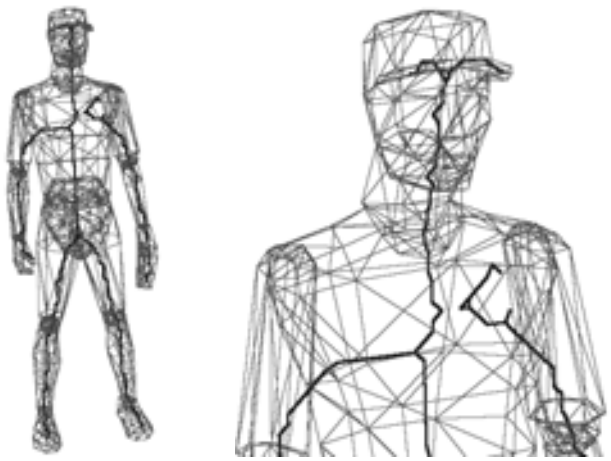
Conclusiones

4.1. Descriptor de Volúmenes

Como se ha mostrado el método aquí descrito puede obtener el descriptor de volúmenes tridimensionales mediante la obtención de su esqueleto y posterior codificación del mismo. Ademas, dado que está basado en un código cadena previo [8], hereda algunas de las bondades del mismo, tales como:

- **Invariante a la traslación** - El uso de direcciones relativas y no posiciones en el espacio hace posible que esta propiedad también se cumpla.
- **Invariante al Vértice de Inicio de Codificación** - Mediante el uso *camino único* dentro del árbol se puede asegurar que la codificación del esqueleto siempre se iniciará en el mismo vértice dando como resultado un descriptor **único** para el objeto.

Desafortunademente, la propiedad de **invariancia bajo la rotación** no es soportada. El procedimiento de obtención del esqueleto requiere de una imagen binaria tridimensional, esto es, un modelo hecho a base de vóxeles. Es por esta razón que un proceso de voxelización (digitalización) es requerido previo a la obtención del esqueleto. Para dicho proceso de voxelización no se tiene un procedimiento que asegure que la imagen binaria resultante sea siempre la misma sin importar el ángulo de rotación del volúmen original, por lo que esta debilidad se extiende a toda la cadena de procedimientos para la obtención final del descriptor. Como trabajo futuro, se piensa que un procedimiento de normalización similar al expuesto en [5], extendido a tres dimensiones sería idoneo para agregar la invariancia a la rotación al descriptor. Más aún, un procedimiento similar también haría al descriptor **invariante a la**



(a) (b)

Figura 4.1: (a) Modelo con esqueleto no conectado y (b) acercamiento al área problemática.

escala lo cual elevaría aún mas la robustez del descriptor.

Otro importante reto a resolver es la conectividad de los esqueletos. Aún cuando el método aquí implementado es una extensión a un método previo resolviendo algunos problemas de conectividad, ésta no se resuelve del todo. En la figura 4.1 se puede observar un modelo con un esqueleto no conectado. Si se observa detenidamente el área del hombro izquierdo en el acercamiento, se puede observar una discontinuidad. Este esqueleto no se puede considerar representativo del modelo original por lo que el método actual se debe refinar considerando casos como el mencionado.

También, hay que notar que a partir de un esqueleto no es posible regresar a la figura original por lo que tampoco es imposible conocer el modelo original a partir de su código cadena.

Tomando en consideración estos problemas se puede concluir que un descriptor único de código cadena basado en la información interna de los modelos es posible. Sin embargo, para agregar la robustez necesaria y un desempeño óptimo del mismo es necesario resolver los importantes retos que la investigación ha mostrado.

4.2. Medida de similitud

La medida de similitud anteriormente propuesta [9] se basa en una búsqueda exhaustiva de patrones comunes dentro de la cadenas, esto hace que la complejidad del computo se eleve incluso a tiempo cuadrático. La principal ventaja de la medida aquí propuesta es su rápidez de computo, ya que solo se necesita recorrer una sola vez la cadena más grande, es decir, se calcula en tiempo lineal.

Los resultados también demuestran que, si bien las disimilitudes obtenidas por la medida propuesta guardan coherencia con las obtenidas por el otro método, se hayan algunas discrepancias. Esto es debido a que en términos prácticos la medida propuesta busca similitudes a nivel local, fallando en detectar pequeños detalles que la medida anteriormente propuesta sí detecta.

Ahora bien, tomemos como ejemplo a las siguiente 2 cadenas:

$$P = 02240\mathit{2142032222}$$
$$S = 02204\mathit{2142032222}0342$$

Se puede ver como a partir de la posición 6 de la cadena existen varios elementos iguales (elementos en cursiva), sin embargo, al ser procesados por el algoritmo que calcula la medida, estos serán recorridos una posición hacia la derecha gracias a la operación de alargamiento que se llevará a cabo en la posición 4, dando como resultado una medida muy alta de disimilitud. Esta falla puede ocasionar que curvas semejantes sean catalogadas como muy diferentes. Una posible solución sería implementar un preprocesamiento en búsqueda subcadenas comunes.

Otra mejora para la medida sería ser el usada junto con algún método de comparación de gráficas para hacer viable la comparación de objetos basados únicamente en sus cadenas. Dicha idea ya ha sido presentada en [16]. Sin embargo, un método basado únicamente en la comparación de elementos de las cadenas es deseado y se propone como un interesante reto a resolver.

Bibliografía

[1] 3d model retrieval system. `http://3d.csie.ntu.edu.tw/~dynamic/`.

[2] 3d model search engine. `http://shape.cs.princeton.edu`.

[3] E. Bribiesca. A chain code for representing 3d curves. *Pattern Recognition*, 33(5):755–765, May 2000.

[4] E. Bribiesca. 3d-curve representation by means of a binary chain code. *Mathematical and Computer Modeling*, 40(3-4):285–295, August 2004.

[5] E. Bribiesca and A. Guzman. How to describe pure form and how to measure differences in shapes using shape numbers. *Pattern Recognition*, 12(2):101–112, 1980.

[6] A. Cayley. A theorem on trees. *Quarterly Journal of Mathematics*, 23:376–378, 1889.

[7] Ma C.M. and Sonka M. A fully parallel 3d thinning algorithm and its applications. *Computer Vision and Image Understanding*, 64(3):420–433, 1996.

[8] Bribiesca E. A method for representing 3d tree objects using chain coding. *Journal of Visual Communication and Image Representation*, 19(3):184–198, 2008.

[9] Bribiesca E. and Aguilar W. A measure of shape dissimilarity for 3d curves. *Int. J. Contemp. Math. Sciences*, 1(15):727–751, 2006.

[10] Paquet E, Murching A, Naveen T, Tabatabai A, and Rioux M. Description of shape information for 2-d and 3-d objects. *Signal Process Image Commun*, 16:103–122, 2000.

[11] H Freeman. On the encoding of arbitrary geometric configurations. *IRE Trans. Electron. Comput*, (10), 1961.

[12] R.M. Haralick and L.G. Shapiro. *Computer and Robot Vision*. Addison-Wesley, 1992.

[13] N. Iyer, Y. Kalyanaraman, K. Lou, S. Janyanti, and K. Ramani. A reconfigurable 3d engineering shape search system part i: Shape representation. In *DETC'03*, 2003.

[14] N. Iyer, Y. Kalyanaraman, K. Lou, S. Janyanti, and K. Ramani. A reconfigurable 3d engineering shape search system part ii: Database indexing, retrieval and clustering. In *DETC'03*, 2003.

[15] Remco C. Veltkamp Johan W. H. Tangelder. A survey of content based 3d shape retrieval methods. *Multimedia Tools Applications*, 39:441–447.

[16] V. Lopez, I. Cheng, E. Bribiesca, T. Wang, and A. Basu. Twist-and-stretch: A shape dissimilarity measure based on 3d chain codes. In *Siggraph Asia*, 2008.

[17] Patrick Min, Michael Kazhdan, and Thomas Funkhouser. A comparison of text and shape matching for retrieval of online 3d models. In *In Proc. European Conference on Digital Libraries*, 2004.

[18] H. Sundar, D. Silver, N. Gagvani, and S. Dickenson. Skeleton based shape and retrieval. In *SMI 2003*, pages 130–139, 2003.

[19] Remco C. Veltkamp and Michiel Hagedoorn. State-of-the-art in shape matching. In M. Lew, editor, *Principles of Visual Information Retrieval*, pages 87–119. Springer, 2001.

[20] Tao Wang and Anup Basu. A note on 'a fully parallel 3d thinning algorithm and its applications'. *Pattern Recognition Letters*, 28:501–506, 2007.