UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

REPRESENTACIÓN Y COMPRESIÓN DE SUPERFICIES VOXELIZADAS USANDO ÁRBOLES Y CÓDIGOS DE CADENA.

T E S I S

QUE PARA OPTAR POR EL GRADO DE:
DOCTOR EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:
M. en C. JOSÉ MIGUEL SALAZAR MONTIEL

Director de Tesis:
Dr. ERNESTO BRIBIESCA CORREA
Instituto de Investigaciones
en Matemáticas Aplicadas y en Sistemas, UNAM.

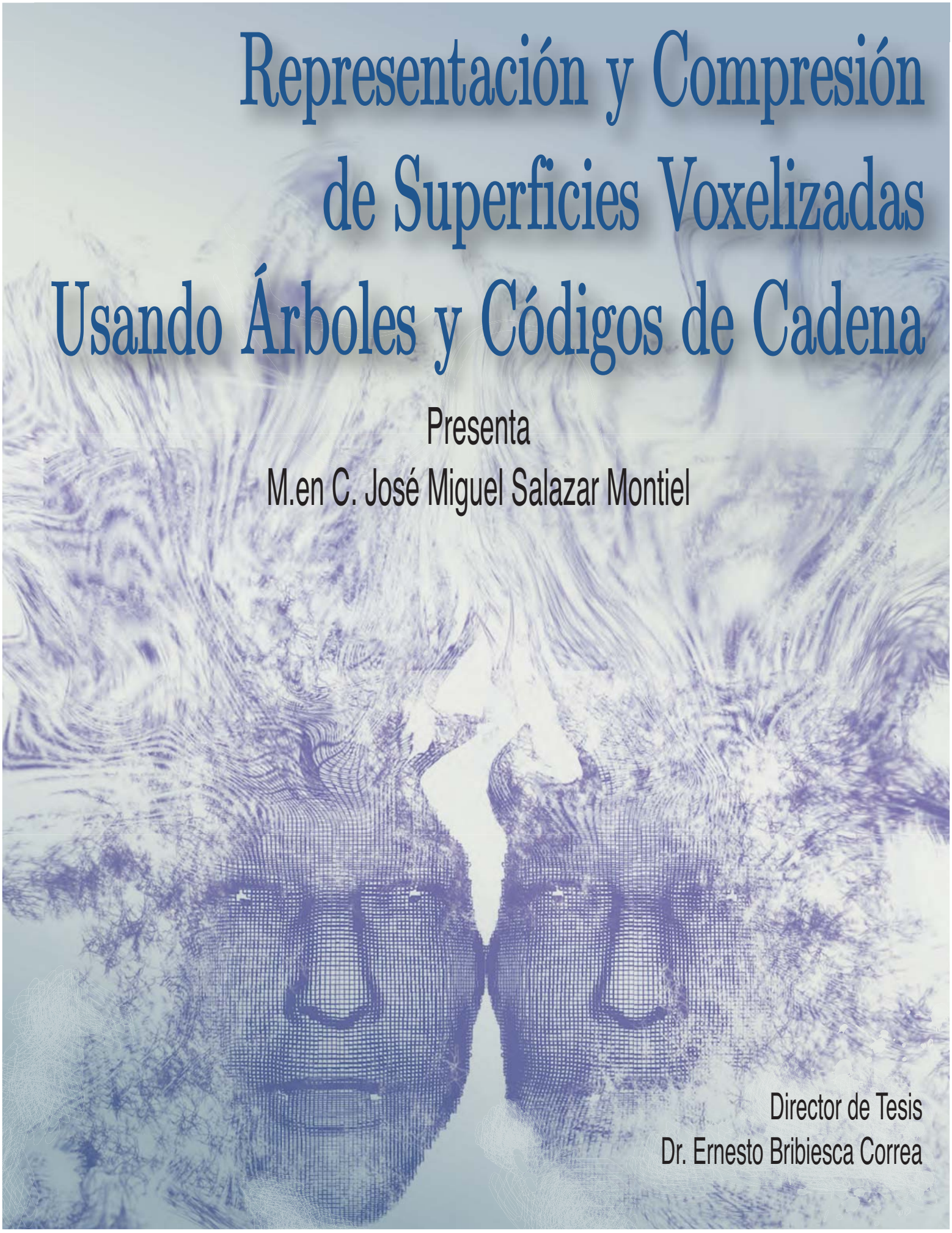
Ciudad Universitaria, CD. MX. Junio, 2017

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Representación y Compresión de Superficies Voxelizadas Usando Árboles y Códigos de Cadena

Presenta

M.en C. José Miguel Salazar Montiel



Director de Tesis

Dr. Ernesto Bribiesca Correa

A la persona que me entiende y admiró
y sin la cual no sería lo que soy gracias a ti Diana.
A mis Jefes y Hermanos.

Agradecimientos

Me gustaría agradecer a todas las personas que de alguna forma me han apoyado a lo largo de toda mi vida, en especial aquellas que estuvieron conmigo durante la realización de este proyecto. A todos ustedes sólo me queda darles las gracias.

A Diana por tu apoyo y ayuda en cada parte de este proyecto. Tu eres la persona que me presiona y hace que esto sea posible.

A mis padres por su apoyo incondicional durante toda mi vida, nunca estaré cerca de poder expresar mi agradecimiento hacia ustedes. A mis hermanos Francisco y el Monstruo por que la vida no sería interesante si no puedo pelear con ustedes. Y a toda mi familia que ha ido agregándose.

Al Dr. Ernesto Bribiesca por su apoyo, compresión y guía durante todo el doctorado. Al Dr. Boris Escalante por su visión, recomendaciones y comentarios. A la Dra. Katya Rodrígez por su ayuda e interés en mi tema original de investigación. Al Dr. Edgar Garduño por sus correcciones y apoyo en los temas de graficación. Y al Dr. Fernando Arámbula por su apoyo como coordinador del posgrado, por sus aportaciones y comentarios al trabajo.

A todas las personas que laboran en el posgrado de ciencias e ingeniería en especial a Lulú y Amalia.

A mis primos Ana y Jacobo por su ayuda.

A todos los compañeros del doctorado Cinthya, Ricardo, Hector,Pedro, Montse, Tzolkin, Rosario, Arturo y los que no son compañeros pero sus conversaciones me ayudaron Ivan, Gibran, Alfonso, Wendy y Caleb.

Amaya y Pablo por ser tan buenos amigos.

Y todos aquellos a pesar de ustedes terminé José, Enano, María, Condor, Aranda, Sandra, Choco, Quet, Martí, Manuel, Betsabe, Chop, Fa, Sus, Aramara, Alux, Ayami, Chut, Renata, Carlos, Fede, Pamela, Gaby, Analí, Ingrid, Roco, y a todos los que me faltan, gracias.

A Cesar, amigo que algún día nos encontraremos.

A todos muchas gracias.

Finalmente quisiera agradecer al posgrado de ciencias e ingeniería de la computación de la Universidad Nacional Autónoma de México por la oportunidad de hacer este trabajo, así como al Consejo Nacional de Ciencias y Tecnología por la beca recibida a través del programa de nacional de posgrados de calidad.

Índice general

Índice de figuras

Índice de tablas

Resumen

El presente trabajo establece la comparación en la cantidad de datos utilizados para almacenar distintas representaciones de superficies representadas a partir de vóxeles. El interés del trabajo se enfoca en representaciones que son invariantes ante rotaciones y traslaciones. Se utiliza la representación de los centros de vóxeles como base para las comparaciones.

Todas las representaciones son construidas dentro del espacio digital $\mathbb{Z}^3$, el trabajo presenta las estructuras del *árbol envolvente* y del *árbol de bordes* para representar a las superficies que se encuentran representadas con vóxeles. El análisis se enfoca primordialmente en la cantidad de datos para representar las estructuras de árbol. Como se utilizan superficies y no objetos con volumen, se busca que las representaciones usen este hecho para reducir la cantidad de datos para ser almacenadas.

Al utilizar estructuras de árbol dentro del espacio $\mathbb{Z}^3$, éstas se pueden representar a partir del *código de cambio de dirección ortogonal de* 5 *direcciones*. Dicha codificación es considerada como una forma eficiente para almacenar estructuras de árbol. Esta codificación nos permite hacer comparaciones en la cantidad de datos a utilizar para almacenar las estructuras del *árbol envolvente* y *árbol de bordes*.

Las comparaciones utilizadas se hacen a partir de superficies de rostros humanos representadas por vóxeles y superficies representadas por vóxeles generadas a partir de modelos digitales de elevación. Usando estas superficies se compara la cantidad de datos necesaria para ser almacenada por las representaciones de *gráfica de adyacencia*, *curva de barrido*, *árbol envolvente* y *árbol de bordes*.

Los resultados obtenidos muestran que en la mayoría de los casos la representación de *árbol envolvente* presenta una compresión en comparación con la cantidad de datos necesaria para almacenar la representación a partir de los centros de los vóxeles.

La codificación de las estructuras de árbol utilizando el *código de cambio de dirección ortogonal de* 5 *direcciones* genera una cadena de símbolos, lo que nos permite hacer una compresión de la cadena utilizando métodos convencionales como la compresión de Huffman.

La investigación detalla los algoritmos utilizados para la obtención de todas las estructuras empleadas, así como los resultados obtenidos. Parte de estos se encuentran publicados en [34].

Capítulo 1

Introducción

La representación de objetos $3D$ ya sea para su modelado, simulación, modificación, reconocimiento, análisis, entre otros. Es uno de los temas más estudiados entre la comunidad de las ciencias de la computación dentro de la rama de graficación por computadora.

Existen distintos tipos de representaciones, las cuales tienen características diferentes, dependiendo de su uso, dichas características son determinantes para evaluar qué representación es la más conveniente utilizar.

Para determinar qué representación es conveniente usar, es común utilizar distintos criterios que evalúen las características de las representaciones. Los criterios varían dependiendo de la aplicación, dentro de los criterios a considerar se encuentran: la facilidad para modificar la representación, la utilización de información adicional dentro del modelo, la cantidad de información que utiliza el modelo para su almacenamiento, la sensibilidad del modelo ante los errores de transmisión, elementos invariantes bajo distintas condiciones, por mencionar algunos.

La representación también depende del tipo de objeto que se desea representar, pues no es lo mismo representar un objeto sólido, una superficie $3D$, un conjunto de puntos, o una estructura topológica (como los nudos), entre otros.

Por ejemplo, los sistemas LIDAR (*Light Detection and Ranging*) utilizan un láser pulsante para obtener las distancias del origen del láser a los objetos en el mundo real. A partir de dicha distancia, la información es recolectada y se genera un conjunto de puntos en $3D$ los cuales son conocidos como nube de puntos (*Point cloud*).

Para representar objetos sólidos los cuales a veces están formados por materiales de distintas densidades, como por ejemplo los órganos en animales o las distintas estructuras dentro del cuerpo humano, es común utilizar la representación de vóxeles (ver Def. 2.27), las cuales representan a los objetos como una colección de cubos a los cuales se les puede asignar distintos valores asociados a la densidad del objeto en esa región. Esta representación es comúnmente utilizada en aplicaciones médicas como las tomografías computarizadas o las resonancias magnéticas o bien en simuladores de coaliciones de partículas.

Otra forma de representar un objeto sólido, es hacer la representación de éste como si fuese un objeto hueco, representándolo únicamente a través de las superficies visibles. Cambiando así el problema a cómo representar superficies en lugar de representar sólidos de forma indirecta.

La forma más común para representar a una superficie es la conocida como "*polygon mesh*" o "*mesh*" presentada en [3]. Esta representación utiliza un conjunto de vértices en $\mathbb{R}^3$ y adyacencias entre éstos para así caracterizar dicha superficie, formando polígonos los cuales representan a una superficie $3D$. La mayoría de las representaciones *mesh* utilizan triangulaciones para formar a los polígonos, esta forma es la más común entre las aplicaciones para mostrar formas complejas, y es la forma más utilizada en aplicaciones como los vídeo juegos, o en los modelos $3D$ que simulen situaciones realistas.

Si se desea comparar representaciones de tipo *mesh*, a partir de los puntos que generan a los polígonos, sería necesario "alinear" las superficies para que las diferencias entre sus puntos fuesen las menos posibles. Al "alinear" se hace referencia a modificar las superficies para que sean lo más cercanas posibles. Dichas modificaciones generalmente son transformaciones geométricas del espacio como son rotaciones, traslaciones, ampliación o reducción. Encontrar dichas transformaciones puede resultar muy complejo, esto debido a que para calcular dichas transformaciones lo más usual es buscar características geométricas (bordes, esquinas, elementos rectos, planos, u otra característica) y generar las transformaciones que alineen mejor dichas características y así poder comparar todo el objeto (ver [24]). Si las características para encontrar las transformaciones son complicadas de obtener o el objeto no cuenta con tales, las transformaciones no pueden ser obtenidas.

Las comparaciones entonces dependen de las transformaciones. Es deseable que para comparar las representaciones de los objetos, éstos no dependan de las transformaciones.

El presente trabajo se enfoca en distintos tipos de representaciones de superficies $3D$ evaluando la cantidad de datos necesarios para almacenar a éstos objetos. El estudio es el resultado de buscar una representación en la cual las formas geométricas se mantengan bajo simples transformaciones afines dentro de $3D$ como son los casos de rotación y traslación. En particular el interés es representar superficies $3D$ generadas a partir de rostros humanos y utilizar la representación de dichas superficies como los elementos para poder hacer una eventualmente una clasificación de los rostros.

Durante la investigación no se realiza ninguna clasificación, pues se consideró que tener para obtener resultados los sistemas de clasificación en las representaciones estudiadas no son robustos como para obtener resultados.

Reconocer objetos es un tema muy estudiado, en el caso particular del reconocimiento de rostros en $2D$ generalmente se utilizan herramientas estadísticas para su identificación (a partir del artículo publicado por M.A Turk y A.P Pentland, ver [36]). Como consecuencia, la utilización de dichas herramientas se traslado a $3D$, y la mayoría de los sistemas de reconocimiento facial en $3D$ basan sus resultados en el uso de éstas herramientas (ver [1] y [39]).

Las herramientas estadísticas evitan tener que encontrar características geométricas y las transformaciones a partir de las cuales se generan las comparaciones, esto hace que el reconocimiento se vea limitado en condiciones donde los rostros no se parecen estadísticamente. Como cuando el rostro se encuentra rotado o bajo distintas condiciones de luz. Estas limitantes se han corregido usando las herramientas estadísticas bajo distintas poses y condiciones de luz o generando un modelo más robusto lo que implica un mayor número de datos.

El uso de características geométricas para el reconocimiento de rostros en $3D$ ha sido poco estudiada (ver [16]). Para hacer un estudio de reconocimiento de rostros usando elementos geométricos se estableció que era necesario encontrar una representación de las superficies $3D$ que mantuviese las propiedades geométricas del rostro y que fuese compacta. Dentro de la literatura especializada no se encontró una investigación en la cual se busquen dichas características en las representaciones, por lo que se considera que esta investigación es novedosa y aporta una contribución dentro del área especializada.

1.1. Definición de la investigación

La investigación se enfoca en analizar representaciones que sean invariantes ante rotaciones y traslaciones, dentro de las cuales la información geométrica del objeto pueda ser extraída. El análisis se basa en la cantidad de datos necesarios para almacenar dichas representaciones.

En la investigación se utilizan como objeto de estudio principalmente superficies de rostros, estas superficies son superficies $3D$ dentro de la base de datos utilizadas en [26] y [30]. Como resultado el estudio utiliza superficies con frontera, orientables y conexas.

Para tener un marco de referencia contra el cual evaluar, en la investigación se utilizó la representación en vóxeles pues es una de las formas más conocidas para representar objetos $3D$. Se utiliza esta representación como base para hacer las comparaciones contra las representaciones que cumpliesen con la propiedad de ser invariantes ante traslaciones y rotaciones. A partir de la representación en vóxeles se generaran representaciones distintas y con base en la cantidad de datos utilizados para su almacenamiento se pueden clasificar y determinar las ventajas de las representaciones.

En la investigación, no sólo se consideran superficies $3D$ voxelizadas a partir de rostros, se incluyen otras superficies $3D$ voxelizadas, como son las superficies generadas a partir de los modelos digitales de elevación (o *DEM* pos sus siglas en inglés).

Se estudiaron representaciones ya establecidas, por lo que se pensó en la representación del *árbol envolvente* como principal objeto de estudio (ver [9]). Esta representación en la forma presentada en [9], representa la superficie externa de los objetos voxelizados, la cual es una superficie cerrada, codificando el *árbol envolvente*, y usando el *código de cambio de dirección ortogonal de 5 direcciones* (presentado en [6]) con la modificación para representar las estructuras de árbol dentro de los espacios digitales (ver [8]).

Esta codificación es una representación unidimensional de una superficie $3D$ la cual contiene información geométrica de la superficie. Se modifica la forma de obtener dicha representación para que represente superficies voxelizadas simples y se estudió la cantidad de información necesaria para almacenar dicha representación.

Aunque la representación para las curvas $3D$ representada usando el *código de cambio de dirección ortogonal de 5 direcciones* presenta una compresión (ver [35]), el caso para una superficie no había sido estudiado, bajo ejemplos simples se observa que al representar la superficie usando la representación del *código de cambio de dirección ortogonal de 5 direcciones del árbol envolvente* no presenta una disminución en la cantidad de datos para su almacenamiento. No se contaba con evidencia para suponer que al representar la superficie usando el *código de cambio de dirección ortogonal de 5 direcciones del árbol envolvente* se utilizará una cantidad menor de datos que la caracterización de vóxeles. Por lo que se estudia las características para determinar si ésta presenta una compresión con respecto a la representación con vóxeles de la superficie y bajo qué condiciones se da.

Se estudian otras representaciones como las *curvas de barrido*, la *gráfica de adyacencia*, el *árbol de bordes*, haciendo un análisis similar para cada una de éstas.

Durante la investigación se busca determinar si las representaciones de árbol envolvente y árbol de bordes presentan una compresión con respecto a la representación de vóxeles y bajo que condiciones. Para tal motivo se detalla las construcciones de estas estructuras y se generan las representaciones para las superficies antes mencionadas. Se calcula la cantidad de datos necesarios para almacenar las representaciones. A partir de los datos obtenidos se comparan y se determina si existe una compresión con respecto a la representación de vóxeles.

1.2. Contenidos

El trabajo presentado se divide en distintos capítulos, a continuación se hace un pequeño resumen de cada uno de éstos.

En el Capítulo 2 se definen las estructuras básicas con las cuales se trabajará, debido a que la representaciones que se discuten en este trabajo son gráficas (o grafos) se da un conjunto de definiciones básicas sobre gráficas. Las estructuras se generan en el espacio donde los vóxeles son los elementos base, se da una breve introducción a los conceptos en este espacio conocido como espacio digital.

En el Capítulo 3 se definen los conceptos básicos de los códigos de cadena, se introducen los que son considerados más relevantes, y se describe como se utilizan para representar ciertas estructuras. En este capítulo se presentan las estructuras de *Curvas de barrido*, *Árbol envolvente* y *Árbol de bordes* que son las representaciones centrales del trabajo.

El Capítulo 4 se establecen los criterios para evaluar la cantidad de datos usados para cada representación y se presenta la representación conocida como *Gráfica de adyacencia*.

El análisis de los resultados obtenidos y discusiones sobre las ventajas y desventajas sobre los distintos métodos de representación de las superficies se encuentran en el Capítulo 5.

En el capítulo final se discuten las representaciones, se hace un sumario de los capítulos anteriores, las contribuciones del trabajo y los posibles trabajos a futuro.

Capítulo 2

Conceptos Básicos

A continuación se expondrán los conceptos básicos esenciales para hacer una exposición concisa del trabajo realizado. Dependiendo de la importancia del concepto en el trabajo se profundizará para una mejor comprensión del mismo.

2.1. Gráficas

Se iniciará con enlistar las definiciones básicas de teoría de gráficas que serán usadas a lo largo del trabajo presentado. El lector interesado en profundizar puede consultar los siguientes libros especializados [4] y [37].

Def. 2.1 (Gráfica)**.** Una *gráfica* G se define como la pareja $G = (V, A)$ donde V es un conjunto finito que se denomina como *vértices* o *nodos* de G y el subconjunto $A \subset V \times V$ las aristas de G. Si $(u, v) \in A$ se dice que hay una arista entre el vértice u y el vértice v. Una gráfica es simple si para toda $(u, v) \in A$ se tiene que $(v, u) \in A$ o bien $(u, v) = (v, u)$. De forma similar una gráfica es *dirigida* si para un $(u, v) \in A$ se tiene que $(u, v) \neq (v, u)$. Se dice que la gráfica tiene tamaño $|V|$, donde $|V|$ es el número de elementos en el conjunto V.

Def. 2.2 (Camino)**.** Un *camino* $\mathcal{C}$ en una gráfica $G = (V, A)$ es un subconjunto de vértices $\mathcal{C} = \{v_1, v_2, \ldots v_k\} \subset V$ donde $(v_i, v_{i+1}) \in A$. Se dice que el camino tiene longitud $|\mathcal{C}| - 1$ y que el vértice v_1 es el vértice inicial y v_k el vértice final.

Def. 2.3 (Trayectoria)**.** Una *trayectoria* en una gráfica $G = (V, A)$ es un camino (Def. 2.2) que no repite vértices. Es decir, si $\{v_1, v_2, \ldots v_k\} \subset V$ es una trayectoria entonces si $v_i, v_j \in \mathcal{C}$ con $i \neq j$ implica que $v_i \neq v_j$.

Def. 2.4 (Camino cerrado)**.** Un *camino cerrado* en una gráfica $G = (V, A)$ es un camino $\{v_1, v_2, \dots v_k\} \subset V$ donde $v_1 = v_k$.

Def. 2.5 (Ciclo)**.** Un *ciclo* $\{v_1, v_2, \dots v_k\} \subset V$ en $G = (V, A)$ es una trayectoria en los vértices $\{v_1, \dots, v_{k-1}\}$ y el vértice inicial y final son iguales (i.e., $v_1 = v_k$). Se dice que el ciclo tiene longitud $k - 1$.

Def. 2.6 (Conexa)**.** Una gráfica $G = (V, A)$ es *conexa* si $\forall u, w \in V$ existe un camino $\{v_1, v_2 \dots v_k\} \subset V$ en G, donde $u = v_1$ y $w = v_k$. La *componente conexa* de un vértice $u \in V$ en $G = (V, A)$ son todos los vértices w en V tales que hay una camino de u a w en G.

Def. 2.7 (Árbol)**.** Una gráfica $G = (V, A)$ es un *árbol* si es conexa y sin ciclos. Un *bosque* es una gráfica cuyas componentes conexas son árboles.

Def. 2.8 (Raíz)**.** A todo árbol se le puede asignar un vértice el cual se denomina *raíz*, del cual hay una única trayectoria del vértice raíz hacia todos los vértices del árbol.

Def. 2.9 (Arbol enraizado)**.** Cuando a un árbol se le asigna un vértice raíz r, se dice que el árbol esta enraizado en r. De esta forma se le puede inducir un orden al árbol, el cual esta definido a partir de la raíz de este.

Def. 2.10 (Vértice hoja)**.** Sea G un árbol, se dice que u es un vértice hoja si sólo tiene una arista y no es el vértice raíz.

Def. 2.11 (Vértice padre)**.** Sea u un vértice del árbol T, el *padre* de u es el primer vértice distinto de u en la trayectoria de u a la raíz de T.

Def. 2.12 (Ancestros)**.** Sea T un árbol, se dice que un vértice u es ancestro de un vértice v, si u es un vértice en la trayectoria de u a la raíz de T.

2.2. Espacios digitales

Las siguientes definiciones son las bases de las estructuras con las que se trabaja primordialmente durante la investigación doctoral. En la sección se presentan los conceptos relacionados con lo que se denomina *geometría digital*, estos primordialmente se basan en los trabajos presentados por Rosenfeld (1931–2004) y otros, los cuales se encuentran presentes en [12] y en [23].

La geometría discreta utiliza conjuntos discretos $\mathcal{A} \subsetneq \mathbb{Z}$ con $|A| < \infty$ para definir los objetos dentro de ésta. Los objetos son de la forma $\mathcal{O} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_m$ donde $\mathcal{A}_i \subsetneq \mathbb{Z}$ y $|\mathcal{A}_i| < \infty$.

La investigación utilizará la topología de la cuadrícula en estos objetos, como se están tomando subconjuntos de $\mathbb{Z}$, y éste es un conjunto bien ordenado, se induce el orden heredado de $\mathbb{Z}$ en el conjunto $\mathcal{O}$, y por lo tanto se puede pensar como una retícula.

Una retícula es un conjunto parcialmente ordenado $(S, \leqslant)$, con S un conjunto y $\leqslant$ el orden en el conjunto S, donde cualquier subconjunto $S' \subset S$ tiene un supremo bajo el orden $\leqslant$.

Por simpleza en la notación se utilizará como objeto de estudio al espacio $\mathbb{Z}^m$ pues siempre se puede insertar a los objetos en este espacio, aunque los conjuntos a considerar son de la forma $\mathcal{O} = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_m$ con $|\mathcal{A}_i| < \infty \; \forall i \in \{1, \ldots, m\}$.

Las definiciones presentadas a continuación pueden ser modificadas para cualquier dimensión finita, en este trabajo sólo se considera los casos $\mathbb{Z}^2$ y $\mathbb{Z}^3$ pues son los espacios donde se trabaja principalmente, enfocando el estudio en $\mathbb{Z}^3$. Los espacios digitales no se limitan únicamente a éstos.

Para tener una definición formal de lo que es un espacio digital, se utiliza una definición similar a la presentada en [12].

Def. 2.13 (Espacio Digital)**.** Un *espacio digital* es un espacio discreto en donde cada punto está definido como un vector de elementos enteros.

Para estudiar el espacio digital $\mathbb{Z}^m$ primero se tiene que entender la relación entre los elementos del mismo espacio. Al elemento más simple se le llama *vértice*, *punto* o 0-celda.

Se representará al espacio digital de forma gráfica usando el plano Cartesiano. Los puntos del espacio digital se representaran como los elementos en el plano Cartesiano con coordenadas enteras (ver Fig. 2.1).

En el espacio digital $\mathbb{Z}^2$ dos vértices $u = (u_1, u_2)$ y $v = (v_1, v_2)$ son adyacentes si $u_i = v_i$ y $|u_j - v_j| = 1$ con $i \neq j$.

A partir de relacionar elementos básicos se generan más estructuras en el espacio digital. Para relacionar los puntos básicos del espacio $\mathbb{Z}^2$, se toma al segmento de recta definido entre dos vértices adyacentes. Al segmento de recta junto con los vértices se le denomina arista o 1-celda (ver Fig. 2.2) y a la arista entre los vértices adyacentes $u, v \in \mathbb{Z}^2$ la denotaremos como $[u, v]$.

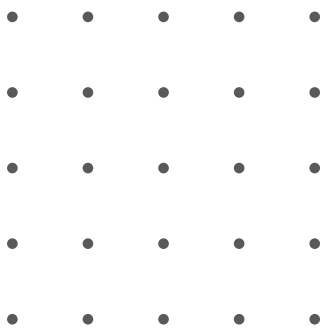


Fig. 2.1: Un subconjunto del espacio $\mathbb{Z}^2$. Donde cada punto es el elemento base del espacio $\mathbb{Z}^2$.

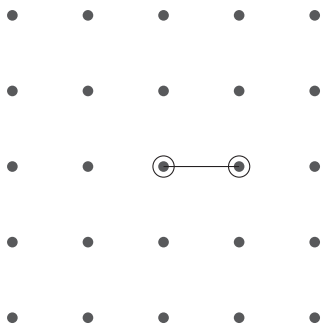


Fig. 2.2: Arista en $\mathbb{Z}^2$.

Para relacionar a las aristas entre si, se define la adyacencia entre las aristas. Estas serán adyacentes entre ellas, si comparten vértices (ver Fig. 2.3). Para el trabajo la definición formal de esta será la siguiente:

Def. 2.14. (Adyacencia aristas) Sean $a = [u, w]$ y $b = [z, x]$ aristas en $\mathbb{Z}^2$ con $u, w, z, x \in \mathbb{Z}^2$ y $a \neq b$. Las aristas a y b son adyacentes si cualquiera de las siguientes es cierta $u = z$, $u = x$, $w = z$ o $w = x$.

De esta forma se define si un vértice es adyacente a una arista, se considera que un vértice v es adyacente a la arista $a = [w, u]$ si v no pertenece a la arista a y v es adyacente a w o u (ver Fig. 2.4).

A partir de las dos estructuras creadas (aristas y vértices) se define la siguiente estructura, en este caso una 2-celda.

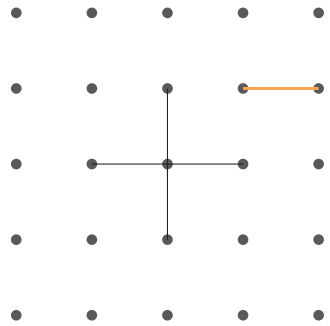


Fig. 2.3: Las aristas en negro son adyacentes entre si, la naranja no es adyacente.

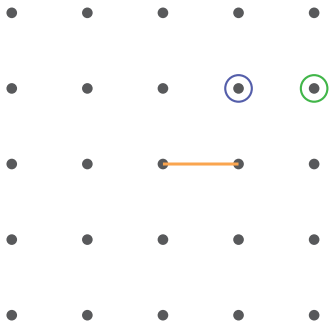


Fig. 2.4: La arista naranja es adyacente al vértice en el círculo azul, pero no es adyacente al vértice en el círculo verde.

Def. 2.15 (píxel)**.** Una *2-celda* es el plano que esta encerrado por 4 aristas y 4 vértices, tal que cumple con las siguientes propiedades. Sean u_0, u_1, u_2, u_3 vértices donde u_1 es adyacente a u_0 y u_2, u_2 es adyacente a u_1 y u_3, u_3 es adyacente a u_2 y u_0 y u_0 es adyacente a u_3 y u_1. Las aristas serían $a_0 = [u_0, u_1]$, $a_1 = [u_1, u_2]$, $a_2 = [u_2, u_3]$ y $a_3 = [u_3, u_0]$. Para denotar la región que encierra las aristas utilizaremos una notación similar para referirnos a las aristas, usando $[,]$ en este caso $[a_0, a_1, a_2, a_3]$ (ver Fig. 2.5). La 2-celda será la región que encierran las aristas, las aristas y los vértices de las aristas. $\mathcal{P} = \{[a_0, a_1, a_2, a_3], a_0, a_1, a_2, a_3, u_0, u_1, u_2, u_3\}$.

La *geometría digital* no tiene muchos años como tal, pues se empezó a formalizar su estudio a partir de la segunda mitad del siglo 20 como resultado de la investigación en los campos de graficación por computadoras y análisis digital de imágenes (ver [12] y [23]). Muchas de las definiciones y algoritmos se utilizaron y desarrollaron antes de la definición formal del concepto de geometría digital. A partir de los años 60s y 70s con los estudios de A. Rosenfeld sobre curvas digitales y sus propiedades topológicas, se empieza una definición formal del concepto de geometría digital. En [21] se da una definición de 2-celda o celda de resolución, aunque su uso se generalizó desde muchos años antes. A continuación se presenta la definición formal descrita en [21] sólo como un antecedente.

Def. 2.16 (Celda de resolucion o Celda)**.** Una celda de resolución es la más pequeña de área primaria que constituye la imagen de intensidad asociada a una imagen digital. Para hacer referencia a una celda de resolución se utilizan sus coordenadas espaciales que son las coordenadas del centro de su superficie. La celda de resolución o formaciones espaciales de celda de resolución, constituyen la unidad básica para el procesamiento de bajo nivel de los datos de imágenes digitales. Las celdas de resolución por lo general tienen áreas que son cuadradas, rectangulares o hexagonales delimitadas por cuadrados, rectángulos o hexágonos.

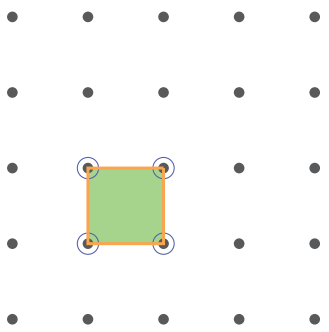


Fig. 2.5: Un píxel con sus vértices en azul y aristas en naranja.

A diferencia de la Def. 2.14 se puede observar que las 2-celdas pueden ser adyacentes sí comparten algunas de las estructuras antes descritas (i.e., aristas o vértices). Es decir, se considera que dos 2-celdas son adyacentes sí en su intersección se encuentra un vértice o una arista. Por tal motivo se tienen dos tipos de adyacencia en el caso de las 2-celdas.

Def. 2.17 (Adyacencia aristas)**.** Se dice que $\mathcal{P}_1 = \{C = [a_0, a_1, a_2, a_3], a_0, a_1, a_2, a_3, u_0, u_1, u_2, u_3\}$ y $\mathcal{P}_2 = \{C' = [b_0, b_1, b_2, b_3], b_0, b_1, b_2, b_3, w_0, w_1, w_2, w_3\}$ son *adyacentes por aristas* sí $\mathcal{P}_1 \cap \mathcal{P}_2 \neq \emptyset$ y $a_i = b_j$ para alguna $a_i \in \{a_0, a_1, a_2, a_3\}$ y $b_j \in \{b_0, b_1, b_2, b_3\}$.

Def. 2.18 (Adyacencia vértice)**.** Se dice que $\mathcal{P}_1 = \{C = [a_0, a_1, a_2, a_3], a_0, a_1, a_2, a_3, u_0, u_1, u_2, u_3\}$ y $\mathcal{P}_2 = \{C' = [b_0, b_1, b_2, b_3], b_0, b_1, b_2, b_3, w_0, w_1, w_2, w_3\}$ son *adyacentes por vértice* sí $\mathcal{P}_1 \cap \mathcal{P}_2 \neq \emptyset$ y $u_i = w_j$ para algún $u_i \in \{u_0, u_1, u_2, u_3\}$ y $w_j \in \{w_0, w_1, w_2, w_3\}$.

Es fácil observar que los elementos que son adyacentes por aristas también lo son por vértice, pero las 2-celdas que son adyacentes por vértice no necesariamente son adyacentes por arista como se puede observar en la Fig. 2.6. La figura muestra tres 2-celdas, la 2-celda azul es adyacente a la 2-celda verde por el vértice en el circulo azul, mientras que la 2-celdas naranja es adyacente al verde tanto por vértices como por aristas puesto que ambas 2-celdas comparten la arista naranja.

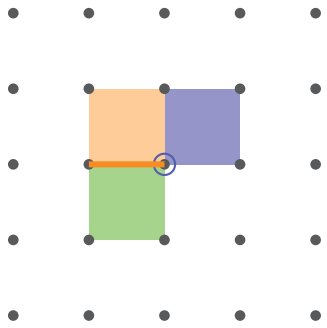


Fig. 2.6: La 2-celda en verde es adyacente a la azul por vértice y a la naranja por arista.

Debido a las definiciones de adyacencia, se puede definir lo que es una vecindad. Se considera una vecindad de un elemento digital u (vértice, arista o 2-celda) a todos los elementos que son adyacentes a u. En el caso de las 2-celdas la adyacencia de un elemento digital depende del tipo de adyacencia que se considera (vértice o arista), entonces la vecindad cambia dependiendo del tipo de adyacencia.

Def. 2.19 (Vecindad)**.** Sea un elemento digital $u \in \mathbb{Z}^2$ su *vecindad* denotada por $\mathcal{N}(u)$ son todos aquellos elementos digitales w que sean adyacentes a u.

Como la vecindad considera todos los tipos de elementos digitales, es necesario especificar el tipo de elemento de interés, por lo que se denotará como $\mathcal{N}_0(u)$ cuando se toman los elementos en $\mathcal{N}(u)$ que sean únicamente vértices, a $\mathcal{N}_1(u)$ cuando se toman los elementos en $\mathcal{N}(u)$ que sean únicamente aristas y $\mathcal{N}_2(u)$ cuando se toman los elementos en $\mathcal{N}(u)$ que sean únicamente 2-celdas.

En la Fig. 2.7 se observa a los elementos dentro de la vecindad de la 2-celda coloreada en negro. La vecindad consta de los vértices dentro de los círculos azules, las aristas en naranja, y las 2-celdas en verde, todos estos elementos están siempre dentro de la vecindad, mientras que los azules sólo cuando se considera la adyacencia por vértices.

La $\mathcal{N}_0(u)$ adyacencia de la 2-celda en negro en la Fig. 2.7 son sólo los vértices en azul, la $\mathcal{N}_1(u)$ son las aristas en naranja. Para denotar la vecindad de las 2-celdas adyacentes por vértices a una 2-celda u se utiliza $\mathcal{N}_2^V(u)$ en la figura las 2-celdas azules y verdes están incluidas. El caso de la vecindad de 2-celdas por aristas de la 2-celdas u se denota como $\mathcal{N}_2^A(u)$ donde en la figura se encuentra únicamente las 2-celdas verdes.

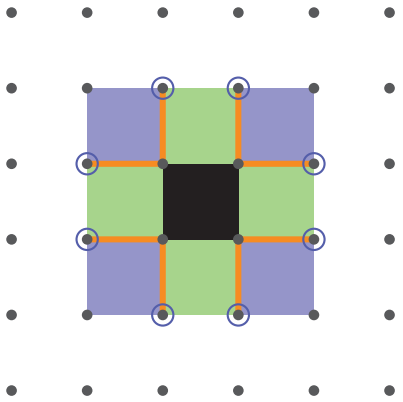


Fig. 2.7: La vecindad de una 2-celda.

2.3. Más conceptos en $\mathbb{Z}^2$

En la Def. 2.16 se habla del concepto de *imagen digital*. Una imagen digital se considera como una función en donde a cada pareja en $M \times N$ donde $M, N \subsetneq \mathbb{Z}$ se le asigna un valor, de esta forma se puede representar una imagen del mundo real. Un ejemplo muy conocido de lo anterior son las fotos digitales donde a cada elemento en una cuadricula se le asigna un color de esta forma, generando así una imagen digital.

Para la geometría digital, una imagen digital será una función del espacio digital a un espacio de intensidad. El espacio de intensidad comúnmente es un conjunto finito con una operación de suma, un espacio vectorial, u otros.

Def. 2.20 (Imagen Digital)**.** Es una función $\mathcal{I}\colon M \times N \to I$ del espacio digital $M \times N$ donde $M, N \subsetneq \mathbb{Z}$ a un espacio de intensidad I.

Las parejas de la forma $(u, f(u))$ son las que constituyen una imagen digital, estas parejas son las que se definen en [21] como píxeles. Por completez del trabajo se introduce la definición de *píxel* presentada en [21] aunque en el trabajo se utiliza la presentada en Def. 2.15.

Def. 2.21 (Píxel)**.** Un *píxel*, (*picture element*) *elemento de imagen* o *pel* es una pareja cuyo primer elemento es una celda de resolución (ver la Def. 2.16) o una pareja (renglón, columna) de una posición espacial y cuyo segundo elemento es el valor en el espacio de intensidad en la imagen o el vector de intensidad de los valores asociados con la posición espacial.

Dependiendo del espacio de intensidad se puede asignar nombres específicos a las imágenes. Cuando el espacio de intensidades es $I = \{0, 1, \ldots 2^n - 1\}$ se dice que la imagen esta en *escala de grises*, en el caso que el espacio de intensidad sea $I = \{0, 1\}$ se dice que la imagen es binaria. El caso donde el espacio de intensidad es $I = \{0, 1, \ldots, 255\} \times \{0, 1, \ldots, 255\} \times \{0, 1, \ldots, 255\}$ y estos representan la intensidades de los colores rojo, verde y azul, respectivamente, la imagen es a color utilizando el sistema RGB (por sus siglas en inglés).

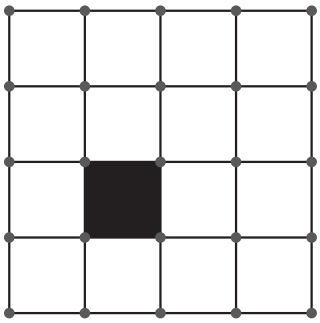


Fig. 2.8: Píxel *encendido* y los demás elementos en la imagen se encuentran *apagados*.

Éste trabajo utiliza sólo imágenes binarias por lo cual cuando en una imagen el valor del píxel es 1 se dice que el píxel esta *encendido* o *prendido* y cuando el píxel tiene valor 0 se encuentra *apagado* (ver la Fig. 2.8).

Dentro de una imagen digital binaria podemos definir lo que es una *figura digital*, la cual esta formada por las 2-celdas encendidas, mientras que los elementos que no pertenece a la figura son las 2-celdas que se encuentran apagadas. El caso de las 1-celdas y de las 0-celdas, la pertenencia a la figura dependerá si éstas se encuentran dentro de una 2-celda encendida.

Otros de los conceptos a definir son las estructuras geométricas que se forman en el espacio digital a partir de las relaciones antes establecidas, una de las estructuras más simples son las curvas. Para definir una curva en el espacio digital se toma la siguiente definición.

Def. 2.22 (Curva digital)**.** Una *curva digital* $\mathcal{C} = \{u_1, u_2, \ldots, u_k\}$ es un conjunto de elementos digitales en donde $u_{i+1} \in \mathcal{N}(u_i)$.

Se observa que la definición de curva digital, ésta puede contener elementos de distintas dimensiones dentro del espacio digital. A lo largo del trabajo se entenderá por una curva digital, únicamente a las curvas cuyos elementos tengan la misma dimensión; por lo tanto, existen curvas de 0-celdas (vértices), curvas de 1-celdas (aristas) o 2-celdas (píxeles) en $\mathbb{Z}^2$.

Def. 2.23 (Longitud Curva)**.** Se define la longitud de una curva como el número de segmentos de recta en la curva, por lo que si una curva tiene k elementos entonces la longitud de ésta será de $k - 1$.

Las curvas tienen distintas características, como regresar al mismo punto de partida, en este caso se dice que la *curva es cerrada* si $\mathcal{C} = \{u_1, u_2, \ldots, u_k\}$ es una curva y $u_1 = u_k$. Otra característica a considerar son los posibles tipos de curvas y la clasificación de éstas. Se dice que una curva es una *curva simple* si no se intersecan consigo misma; es decir, si $i \neq l$ entonces $u_j \neq u_l$ para toda $i, j \in \{0, \ldots, k\}$. Se dice que una *curva es simple cerrada* si cumple ser simple para todos los elementos salvo los extremos de la curva.

A partir de la definición de vecindad se definirán conceptos básicos de topología para los espacios digitales, dentro de los conceptos a destacar para este trabajo, se consideran aquellos relacionados con los elementos de una figura en el espacio digital; es decir a los elementos internos, externos y los elementos frontera.

Def. 2.24 (Interno)**.** Un elemento digital w es *interno* si todos los elementos de $\mathcal{N}(w)$ y w pertenecen a la figura.

Def. 2.25 (Frontera)**.** Se dice que un elemento digital w es *frontera* (o *exterior*) de una figura, si w pertenece a la figura y en $\mathcal{N}(w)$ existe un elemento que no pertenece a la figura. Se denota a la frontera de un conjunto $\mathcal{C}$ como $\partial(\mathcal{C})$

Def. 2.26 (Externo)**.** Un elemento se le denomina *externo* si no pertenece a la figura

2.4. El espacio $\mathbb{Z}^3$

Las definiciones básicas presentadas en la Sección 2.2 para el espacio $\mathbb{Z}^2$ se pueden modificar para definir los conceptos en dimensiones mayores. A continuación se presentan las definiciones básicas para el espacio $\mathbb{Z}^3$.

En el caso del espacio $\mathbb{Z}^3$ se usará la misma forma de construir las distintas estructuras que se utilizó en el espacio $\mathbb{Z}^2$, donde se usan a las 0-celdas (vértice) y a partir de éstas se generan otras estructuras como las 1-celdas (aristas) y las 2-celdas. En el caso de las 2-celdas (píxeles) cuando se encuentran en $\mathbb{Z}^3$ son llamadas *caras* o *surfels*.

De igual forma que en el espacio $\mathbb{Z}^2$ donde la 2-celda es la estructura de dimensión máxima en $\mathbb{Z}^3$ se tiene la 3-celda o *vóxel*.

En [21] definen a un vóxel como: "Un vóxel, abreviatura para elemento volumétrico (en inglés) es un par ordenado cuyo primer componente es una (renglón, columna, capa) ubicación dentro de un volumen de un paralelepípedo rectangular y cuyo segundo componente es el vector de propiedades del volumen del paralelepípedo rectangular".

En este trabajo se tomará a un vóxel de forma similar a las 2-celdas, es decir un vóxel está formado por 8 0-celdas, 12 1-celdas, 6 2-celdas y todas juntas forman a la 3-celda o vóxel.

Def. 2.27 (vóxel)**.** Una 3-celda o *vóxel* es un cuboide definido por 8 0-celdas $V = \{v_0, \ldots, v_7\}$ donde cada una de esta es adyacente a tres del conjunto V. Se toman las 12 posible 1-celdas $A = \{a_{01} = [v_0, v_1],\ a_{12} = [v_1, v_2],\ a_{23} = [v_2, v_3],\ a_{03} = [v_0, v_3],\ a_{04} = [v_0, v_4],\ a_{15} = [v_1, v_5],\ a_{26} = [v_2, v_6],\ a_{37} = [v_3, v_7]\ a_{45} = [v_4, v_5],\ a_{56} = [v_5, v_6],\ a_{67} = [v_6, v_7].\ a_{47} = [v_7, v_4]\}$ formadas por los elementos en V, a partir de estas y los elementos de V se generan las 6 2-celdas o caras $C = \{c_{0123} = [a_{01}, a_{12}, a_{23}, a_{03}],\ c_{0154} = [a_{01}, a_{15}, a_{45}, a_{04}],\ c_{1265} = [a_{12}, a_{26}, a_{56}, a_{15}],\ c_{2376} = [a_{23}, a_{37}, a_{67}, a_{36}],\ c_{0374} = [a_{03}, a_{37}, a_{47}, a_{04}]\}$. Los vértices, las aristas y las caras junto con el volumen delimitado por las caras es lo que se consideran como un *vóxel* $B = \{V, A, C\}$.

Al igual que en el espacio $\mathbb{Z}^2$, la forma para generar distintas estructuras en el espacio $\mathbb{Z}^3$ es a través de la adyacencia de los elementos en el espacio. Dos vóxeles son adyacentes si comparten elementos del espacio digital (vértices, aristas o caras). Por lo que se tienen tres tipos de adyacencias distintas.

Def. 2.28 (Adyacencia)**.** Si $B = \{V, A, C\}$ es un vóxel con vértices V, aristas A y caras C. Y $B' = \{V', A', C'\}$ es un vóxel con vértices V', aristas A' y caras C', entonces se dice que B' es *adyacente por vértices* a B si $V \cap V' \neq \emptyset$. De forma análoga es B' *adyacente por aristas* a B si $A \cap A' \neq \emptyset$. Por último es B' *adyacente por caras* a B si $C \cap C' \neq \emptyset$.

Se puede definir la vecindad de un vóxel $B = \{V, A, C\}$ de forma similar a la Def. 2.19, pero para no aumentar elementos y notaciones que no se usaran en el trabajo, sólo se consideran las vecindades conformadas por vóxeles. Para los demás elementos digitales se dará de forma explicita los elementos a usar. A continuación se consideran las vecindades de vóxeles de un determinado vóxel.

Def. 2.29 (Vecindad vóxel por vértices)**.** Un vóxel B' pertenece a la vecindad de un vóxel B por vértice, denotada por $\mathcal{N}_V(B)$, si B' es adyacente por vértices a B.

Def. 2.30 (Vecindad vóxel por arista)**.** Un vóxel B' pertenece a la vecindad de un vóxel B por arista, denotada por $\mathcal{N}_A(B)$, si B' es adyacente por aristas a B.

Def. 2.31 (Vecindad vóxel por cara)**.** Un vóxel B' pertenece a la vecindad de un vóxel B por caras, denotada $\mathcal{N}_C(B)$, si B' es adyacente por caras a B.

Por las definiciones anteriores se observa que si un vóxel $B = \{V, A, C\}$ es adyacente por caras a $B' = \{V', A', C'\}$, entonces B también es adyacente a B' por aristas.

Al ser adyacentes los vóxeles por caras, implica que $C \cap C' \neq \emptyset$, por lo tanto, existe $c \in C \cap C'$. Sin pérdida de generalidad se puede suponer que $c = \{a_{i_0}, a_{i_1}, a_{i_2}, a_{i_3}\}$ donde a_i son las aristas que determinan la cara c. Por lo tanto, $a_{i_0}, a_{i_1}, a_{i_2}, a_{i_3} \in A \cap A'$. Lo que implica que B y B' son adyacentes por aristas.

La demostración donde la vecindad por aristas esta contenida dentro de la vecindad por vértices es análoga a la anterior, por tal razón se omite.

La vecindad por caras esta contenida en la vecindad por aristas y la vecindad por aristas esta contenida en la vecindad por vértices.

Cada vóxel tiene 6 caras, por lo tanto, el número máximo de vóxeles contenidos en la vecindad por caras es 6, por lo que en la literatura a la vecindad por caras se le llama 6-vecindad. A la vecindad por caras del vóxel B se denota como $\mathcal{N}_6(B)$. En la Fig. 2.9 se muestra a un vóxel en color rojo con los vóxeles que son adyacentes por caras.

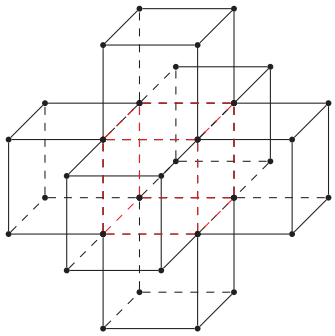


Fig. 2.9: La 6 vecindad de un vóxel.

Los elementos que son adyacentes por aristas al vóxel en rojo se muestra en la Fig. 2.10, pero la vecindad por aristas de un vóxel serian los elementos que se muestran en la Fig. 2.9 más los 12 elementos en la Fig. 2.10. Por esta razón en la literatura a la vecindad por aristas se le denomina 18 vecindad y la se denota como $\mathcal{N}_{18}(B)$.

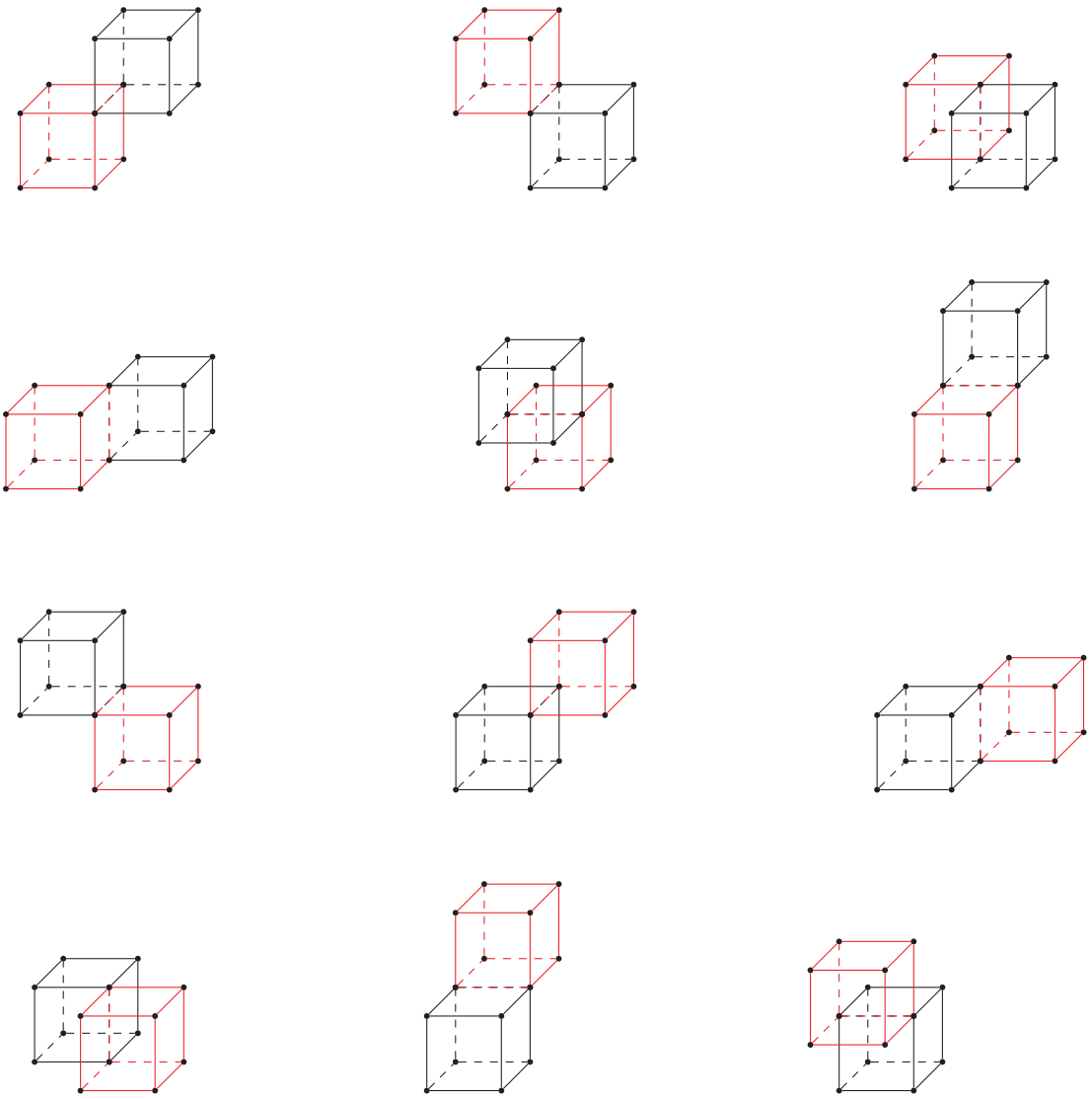


Fig. 2.10: Los elementos adyacentes por aristas de un vóxel.

Los elementos que son únicamente adyacentes por vértice se muestran en la Fig. 2.11 por las afirmaciones anteriores la vecindad por vértices tiene que contener a la vecindad por caras y la vecindad por aristas por lo que a dicha vecindad se le llama 26 vecindad y se le denota como $\mathcal{N}_{26}(B)$.

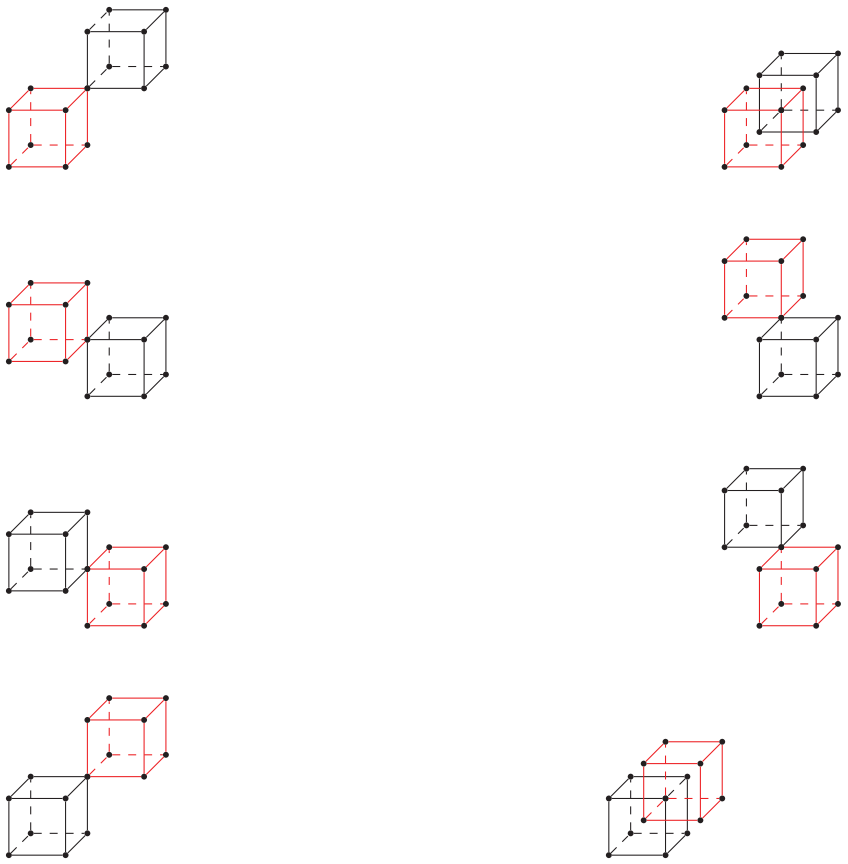


Fig. 2.11: Los elementos adyacentes por vértices de un vóxel.

Dentro de la literatura especializada a las vecindades antes definidas $\mathcal{N}_{26}$, $\mathcal{N}_{18}$ y $\mathcal{N}_6$, también se les denomina como la vecindad 0-dimensional, 1-dimensional y 2-dimensional respectivamente. La razón de esto es que a los vóxeles se les puede dar la estructura de complejo simplicial. Si se toma la intersección de dos vóxeles, éstos tienen que tener en común complejos simpliciales de dimensión menor. Como ejemplo si se toma el caso de que compartan un vértice, entonces el complejo simplicial de dimensión menor es de dimensión 0, o como se denotó anteriormente, una 0-celda. De forma análoga la vecindad 1-dimensional y la 2-dimensional.

Otro tipo de vecindades que se pueden definir son las vecindades de los elementos de un vóxel, es decir para cada vértice, arista o cara perteneciente a un vóxel, se puede definir su vecindad de vóxeles con respecto al vóxel del cual forman parte.

Def. 2.32. Un vóxel $B_i = \{V_i, A_i, C_i\}$ pertenece a la ***vecindad de un vértice*** v con respecto al vóxel $B = \{V, A, C\}$, si el vértice v pertenece a B y B_i ($v \in V$ y $v \in V_i$), y el vóxel B_i se encuentra en la vecindad de B ($B_i \in \mathcal{N}_n(B)$ donde $n \in \{6, 18, 26\}$).

Def. 2.33. Un vóxel $B_i = \{V_i, A_i, C_i\}$ pertenece a la ***vecindad de una arista*** a con respecto a un vóxel $B = \{V, A, C\}$, si la arista a pertenece a B y B_i ($a \in A$ y $a \in A_i$), y el vóxel B_i se encuentra en la vecindad de B ($B_i \in \mathcal{N}_n(B)$ donde $n \in \{6, 18, 26\}$).

Def. 2.34. Un vóxel $B_i = \{V_i, A_i, C_i\}$ pertenece a la ***vecindad de una cara*** c con respecto a un vóxel $B = \{V, A, C\}$, si la cara c pertenece a B y B_i ($c \in C$ y $c \in C_i$), y el vóxel B_i se encuentra en la vecindad de B ($B_i \in \mathcal{N}_n(B)$ donde $n \in \{6, 18, 26\}$).

Con la definición del elemento básico del espacio $\mathbb{Z}^3$ se pueden formar imágenes digitales de la misma forma que se hizo en $\mathbb{Z}^2$, es decir se define una imagen como una función del espacio $\mathbb{Z}^3$ a un espacio de intensidad. En el caso del presente trabajo únicamente se considera al espacio de intensidad como el conjunto $I = \{0, 1\}$.

De forma análoga a lo establecido en la Sección 2.2, un vóxel v en $\mathbb{Z}^3$ se encuentra encendido en la imagen $Im\colon \mathbb{Z}^3 \to \{0, 1\}$ si $Im(v) = 1$. Si $Im(v) = 0$ el vóxel se encuentra apagado .

Para analizar las imágenes digitales el interés se basa en las propiedades y estructuras geométricas dentro de éstas. Como las estructuras se definen a partir de la adyacencia se puede observar que las definiciones como curva digital (ver Def. 2.22), elemento interno (ver Def. 2.24), elemento frontera (ver Def. 2.25) y elemento externo (ver Def. 2.26) se pueden definir de la misma forma en el espacio $\mathbb{Z}^3$.

Def. 2.35. Se denomina a un vértice como ***vértice externo*** si pertenece a un vóxel encendido y dentro de su vecindad existe un vóxel apagado.

Def. 2.36. A una arista se le denomina ***arista externa*** si se encuentra dentro de un vóxel encendido y dentro de su vecindad existe un vóxel apagado.

Def. 2.37. Una cara se le llama ***cara externa*** si se encuentra dentro de un vóxel encendido y dentro de su vecindad existe un vóxel apagado.

Para representar objetos $3D$ en el espacio $\mathbb{Z}^3$ se utilizan distintos tipos de digitalizaciones. Una digitalización se considera como una función que determina que elementos del espacio digital representan a un objeto $3D$. De esta forma se generan imágenes digitales que representan objetos y cuyas propiedades geométricas estarán determinadas por el objeto $3D$ original.

La forma más simple de digitalización en el espacio $\mathbb{Z}^3$ es tomar un encaje del objeto $\mathcal{O} \subset \mathbb{R}^3$ en el espacio digital y asignar los elementos digitales que se intersecan con el objeto. Existen muchos tipos de digitalizaciones como la digitalización de Gauss, externa de Jordan, interna de Jordan, (para ver algunas de éstas se pueden consultar a las siguientes fuentes [12], [23] y [20]).

Para el espacio $\mathbb{Z}^3$ se puede establecer una digitalización usando los elementos representativos del espacio $\mathbb{Z}^3$, es decir los vóxeles.

Def. 2.38 (Voxelización)**.** Una *voxelización* de un objeto $\mathcal{O} \subset \mathbb{R}^3$, es una representación en el espacio $\mathbb{Z}^3$ del objeto $\mathcal{O}$. Es decir, una función del cubo $\mathbb{Z}^3$ en $\{0, 1\}$ donde los elementos encendidos representan al objeto $\mathcal{O}$ en el espacio $\mathbb{Z}^3$.

Existen distintos tipos de objetos que se pueden generar dentro del espacio $\mathbb{Z}^3$, como pueden ser puntos, curvas, superficies, volúmenes, u otros. Aunque existen distintas definiciones para estos objetos, en el presente trabajo se presentan las que se consideran convenientes para que el trabajo se encuentre auto contenido. Las definiciones que se utilizan para definir qué es una superficie digital se toman de [12].

Para definir una *superficie digital* se tomará un conjunto de elementos digitales que sea lo más similar a lo que se considera una superficie en la topología clásica, es decir, que localmente sea similar a un plano bajo transformaciones continuas. En este caso lo más similar a un plano localmente, sería las caras de los vóxeles. Al perder la continuidad en el espacio $\mathbb{Z}^3$ es necesario encontrar otra forma de definir lo que se considera una superficie. Se observa que para que exista una cara de un vóxel, las aristas de éste se tienen que encontrar dentro de la superficie. Por lo tanto, si una arista se encuentra en la superficie es necesario que dentro de la superficie exista otra arista con la cual pueda formar una 2-celda, en consecuencia se define el concepto de *movimiento paralelo* como:

Def. 2.39 (Movimiento paralelo)**.** Sean aristas (1-celdas) $[p, p']$ y $[q, q']$ formadas por $p, p', q, q' \in \mathbb{Z}^3$ respectivamente en el espacio digital. $[q, q']$ es un movimiento paralelo de la arista $[p, p']$, si q y q' son adyacentes a p y p' respectivamente, pero q y p' no son adyacentes, y q' y p tampoco son adyacentes.

A partir de esta definición se puede considerar lo que se denomina como una *superficie digital*.

Def. 2.40 (Superficie digital)**.** Un conjunto conexo S de $\mathbb{Z}^3$ es una superficie digital, si en cualquier punto $p \in S$ está incluido en una 2-celda de S y se cumple que

1. Cualquier 2-celda está conectado por arista en S.

2. Toda 1-celda tiene uno o dos movimientos paralelos en S

3. S no contiene 3-celdas.

A partir de esta definición se puede hacer una clasificación básica de los elementos de la superficie digital a partir del número de movimientos paralelos de las 1-celdas (aristas).

Def. 2.41 (Frontera Superficie)**.** La frontera de una superficie digital, está formada por los elementos digitales que se encuentran en aristas que sólo tienen un movimiento paralelo.

De esta forma se puede clasificar a las superficies digitales a partir de los elementos que se encuentran en su frontera.

Def. 2.42 (Superficie Cerrada)**.** Una superficie digital es cerrada, cuando divide al espacio en dos componentes, una componente que se considera interna y la otra que se considera externa.

Si tomamos en cuenta la Def. 2.41 se puede ver que cuando una superficie es cerrada la frontera de la superficie es vacía.

Def. 2.43 (Superficie simple)**.** Se dice que una superficie digital es simple, si la frontera de la superficie es la unión de curvas digitales cerradas.

No debe confundirse una superficie digital con una superficie voxelizada, una superficie voxelizada no será una superficie digital, pues la superficie voxelizada es la voxelización de una superficie y ésta contiene 3-celdas, mientras que una superficie digital no puede contener 3-celdas.

A partir de una figura conexa en $\mathbb{Z}^3$ se le puede asignar distintas estructuras para describirla o representarla. Una de las estructuras que se puede generar utilizando las definiciones de adyacencia es la que se conoce como gráfica de adyacencia.

Def. 2.44 (Gráfica de Adyacencia)**.** La *gráfica de adyacencia* es la gráfica resultante de tomar los centros de los vóxeles como vértices. Dos vértices en la gráfica de adyacencia son adyacentes, es decir hay una arista entre ellos, si son adyacentes bajo el tipo de adyacencia considerada en $\mathbb{Z}^3$.

Por la definición anterior se tienen tres tipos de gráficas de adyacencia: la gráfica de adyacencia cuando se considera la vecindad 6, 18 o 26. Aunque tienen el mismo número de vértices, el número de aristas es distinto. Otro aspecto a considerar de la gráfica de adyacencia, es que no es una característica única de un objeto en $\mathbb{Z}^3$. Dicho de otro modo, dos objetos pueden tener la misma gráfica de adyacencia, pero esto no implica que los objetos sean iguales.

Otra representación de una figura conexa, es a partir de una curva que pase por todos sus elementos, las curvas que pasan por todos sus elementos y no se interseca en ninguna parte se le conoce como curva de barrido.

Def. 2.45 (Curva de Barrido)**.** Una curva de barrido $\mathcal{C}$ de un objeto $\mathcal{O} \subset \mathbb{Z}^3$ es una curva que pasa por todos los elementos del objeto $\mathcal{O}$ y es simple.

No todos los objetos en $\mathbb{Z}^3$ tienen una curva de barrido, por lo cual dicha representación no se puede considerar como una buena representación para utilizar en distintos tipos de objetos. Pero cuando un objeto tiene una curva de barrido, ésta es una forma simple de describir a un objeto. En [7] se utiliza la curva de barrido para representar modelos de elevación digital, conocidos como *DEM* en la Sección 3.3 se profundiza sobre este tipo de curvas.

Capítulo 3

Códigos de Cadena

En [17] Freeman introduce una forma para describir una figura digital dentro de una imagen digital, describiendo las curvas que forman parte de la figura digital en $\mathbb{Z}^2$. Ésta se basa en describir las curvas utilizando las direcciones de los segmentos de recta que forman la curva. La descripción codifica las direcciones utilizando un elemento en el conjunto $\mathcal{A} = \{0, 1, \ldots, n\}$ donde $n = \{3, 7\}$, a dicho conjunto se le conoce como alfabeto en la teoría de códigos (ver [15]). Asignando las direcciones de segmentos de recta a tomar dentro del plano digital ($\mathbb{Z}^2$) con un símbolo (ver Fig. 3.1 y Fig. 3.2).

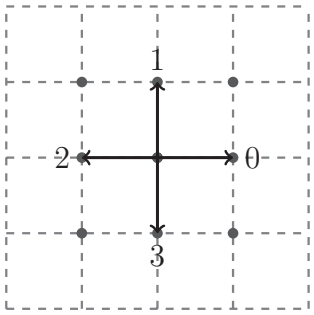


Fig. 3.1: Código de Freeman para vértice central en el espacio digital con $\mathcal{A} = \{0, 1, 2, 3\}$.

Al emplear la definición de curva digital (ver en el Capítulo 2 la Def. 2.22), si $\mathcal{C}u = \{w_0, w_1, \ldots w_k\}$ es una curva digital, entonces el código de Freeman utiliza las direcciones $\mathcal{D} = \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$ para el alfabeto de la Fig 3.1 y $\mathcal{D} = \{(1, 0), (1, 1), (0, 1), (-1, 1), (-1, 0), (-1, -1), (0, -1), (1, -1)\}$ para el alfabeto de la Fig. 3.2, para describir la curva. Sea $\{\vec{d_0}, \vec{d_1} \ldots, \vec{d_{k-1}}\}$ el conjunto de direcciones, donde d_i es la dirección del elemento w_i a w_{i+1}. Se define una función para codificar las direcciones $F : \mathcal{D} \to \mathcal{A}$.

La codificación de la curva queda de la forma $F(\vec{d_0})F(\vec{d_1})\ldots F(\vec{d_{k-1}})$, la cual es un elemento de $\mathcal{A}^*$, donde $\mathcal{A}^* = \bigcup_{i=1}^{\infty} A^n$ con $\mathcal{A}^n = \{\, w_1 \ldots w_n \mid\ w_i \in \mathcal{A}\ \ \forall i \in \{1, \ldots, n\}\,\}$.

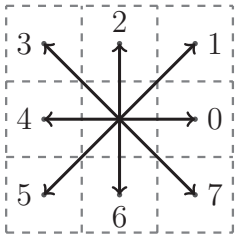


Fig. 3.2: Código de Freeman para píxeles.

En [18] Freeman propone una codificación para las curvas en $3D$ la cual se muestra en la Fig. 3.3. En este código se utiliza el alfabeto $\mathcal{A}_{F3d} = \{00, 01, 02, 03, 04, 05, 06, 07, 10, 11, 12, 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 24, 25, 26, 27, 28\}$ codificando así las 26 posibles direcciones a tomar dentro del espacio digital $\mathbb{Z}^3$.

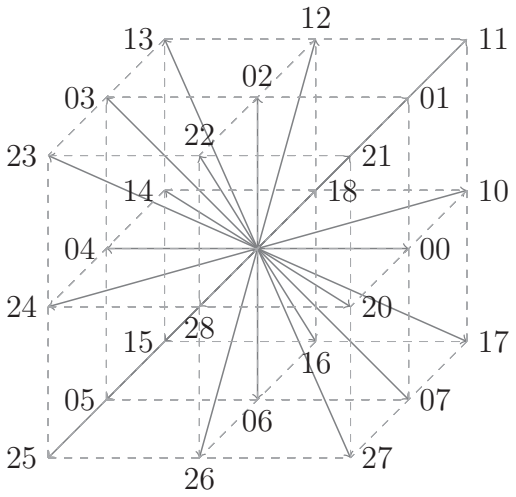


Fig. 3.3: Código de Freeman para dimensión $3D$.

Al utilizar el código de cadena de Freeman, en vez de la descripción completa de los elementos de la curva, se puede observar que la representación es una representación sin pérdida e invariante ante traslaciones. Aunque la representación de una curva utilizando el código de Freeman es una forma simple de representar a una curva, esta representación no es invariante ante rotaciones.

El código de Freeman es capaz de representar a un objeto a través de la curva de su contorno, pero para describir a éste en distintas situaciones es preferible que sea invariante bajo transformaciones básicas como lo son las rotaciones o escalamientos, cosas que el código de Freeman es susceptibles ante dichos cambios.

3.1. Código de cadena para vértices

En [5] Bribiesca presenta un código de cadena distinto al presentado por Freeman. El código presentado se basa en utilizar los cambios de dirección en los vértices de una curva de perímetro de contacto de una figura en $\mathbb{Z}^2$, este código se le conoce como el *código de cadena de vértices* o *VCC* por sus siglas en inglés y cuya representación se encuentra en la Fig. 3.4.

Usando el código *VCC*, la curva en la Fig. 3.5 de 0-celdas, o vértices, es codificada. Para representar dicha curva, se utiliza el número de 2-celdas encendidas que están en contacto con cada vértice. El alfabeto del código utilizado representa el número de vértices de los píxeles encendidos en ese punto del contorno. Como sólo se utilizan píxeles cuadrados, se puede observar que sólo hay tres posibilidades (ver la Fig. 3.4), por lo que el alfabeto del código es $\mathcal{A}_{VCC} = \{1, 2, 3\}$.

Para normalizar el vértice de inicio, cuando se tienen curvas cerradas se selecciona al vértice que genere la cadena que represente el menor número entero, tomando siempre la dirección en sentido de las manecillas de reloj. De esta forma se evita tener múltiples vértices iniciales y distintas cadenas que representen a la misma figura.

Esta forma de codificar las curvas de vértices en $\mathbb{Z}^2$ presenta diferencias con respecto al código de Freeman, entre las que se resaltan las de ser invariante con respecto a la translación, rotación y la transformación espejo, además presenta una reducción en la cantidad de los datos para guardar la curva (ver [35]).

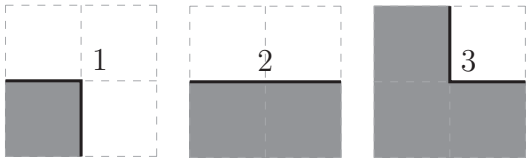


Fig. 3.4: El código de cadena *VCC* donde $\mathcal{A}_{\mathcal{VCC}} = \{1, 2, 3\}$.

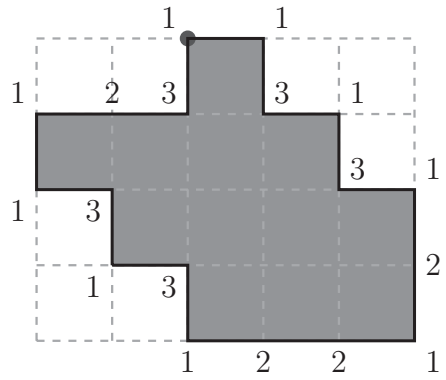


Fig. 3.5: Fig. con el *VCC* en cada uno de sus vértices.

3.2. Códigos de cadena y estructuras en $\mathbb{Z}^3$

Como se ve en el inicio del Capítulo 3, en la Fig. 3.3, Freeman propone un código de cadena en el espacio $\mathbb{Z}^3$, presentado en la Fig. 3.3, utilizando un alfabeto de 26 elementos. En [6] Bribiesca presentan un código de cadena, el cual utiliza únicamente 5 símbolos. Y al igual que el *VCC*, presentado en la Sección 3.1, es invariante ante traslaciones y rotaciones.

Similar al *VCC*, donde los cambios de dirección de una curva se pueden utilizar para describir a ésta, el código presentado en [6] utiliza los cambios de dirección en el espacio $\mathbb{Z}^3$ para hacer una descripción de la curva. Dentro del espacio $\mathbb{Z}^3$ los cambios de dirección son ortogonales en los casos de una curva de 0-celdas (vértices) de 1-celdas (aristas) y de 2-celdas (caras o *surfels*). Para el caso de 3-celdas (vóxeles) es necesario que la curva a considerar sea una curva 6-conexa lo cual implica que todos los cambios de dirección sean ortogonales.

Al código presentado en [6] se le llama *código de cambio de dirección ortogonal de 5 direcciones* o *5ODCCC* (por sus siglas en inglés) y codifica una dirección utilizando las dos direcciones anteriores; denotemos a $\vec{u}$ como una dirección entre elementos digitales. Si $\vec{v}$,$\vec{u}$,$\vec{w}$ son tres direcciones consecutivas se utiliza la regla que se muestra en la Fig. 3.6.

Por lo que si $\mathcal{C} = \{z_o \dots z_k\}$ es una curva digital y $\{\vec{v}_0, \dots, \vec{v}_{k-1}\}$ representa las direcciones, donde $\vec{v}_i$ es la dirección de z_i a z_{i+1}. Para codificar la dirección $\vec{v}_i$ se utilizan las direcciones $\vec{v}_{i-1}$ y $\vec{v}_{i-2}$ usando la siguiente fórmula:

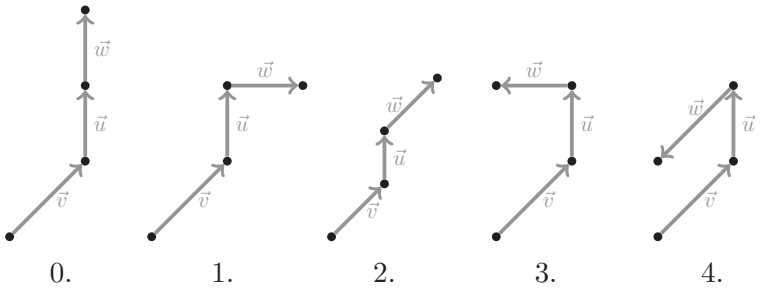


Fig. 3.6: Código de cadena *5ODCCC*.

$$5ODCCC(\vec{v}_i, \vec{v}_{i-1}, \vec{v}_{i-2}) \quad \begin{cases} 0 & \text{si } \vec{v}_i = \vec{v}_{i-1}. \\ 1 & \text{si } \vec{v}_i = \vec{v}_{i-2} \times \vec{v}_{i-1}. \\ 2 & \text{si } \vec{v}_i = \vec{v}_{i-2}. \\ 3 & \text{si } \vec{v}_i = \vec{v}_{i-1} \times \vec{v}_{i-2}. \\ 4 & \text{si } \vec{v}_i = -\vec{v}_{i-2}. \end{cases} \tag{3.1}$$

Donde $\times$ representa el producto cruz en $\mathbb{R}^3$ aplicado a las direcciones con entradas enteras. Como sólo se están tomando entradas enteras y ortogonales, se garantiza la cerradura de la operación.

El 0 representa que la curva mantiene la última dirección, mientras que los elementos 2 y 4 mantiene a la curva dentro de un mismo plano determinado por los dos cambios de dirección anteriores. En los casos 1 y 3 la curva cambia de plano con respecto a sus últimos dos cambios de dirección.

Para los casos donde $\vec{v}_{i-1} = \vec{v}_{i-2}$ se toman las últimas direcciones que sean distintas y se codifica con base a éstas. De esta forma se pueden codificar las curvas en $\mathbb{Z}^3$.

Es de observar que para obtener el código de cadena de una curva en $\mathbb{Z}^3$ es necesario tener al menos tres direcciones para su codificación, por lo que si la curva es de menor tamaño, no se puede codificar. Por esta razón, una curva no puede ser codificada desde su primer cambio de dirección, para remover esta limitante, se le añaden dos direcciones perpendiculares entre ellas ("manija") antes del inicio de la misma para poder codificar desde el inicio.

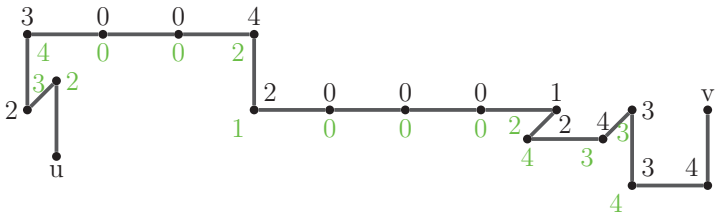
Entre las propiedades del código *5ODCCC* se encuentra la invariancia ante la rotación. Como el código depende de los cambios de dirección de los elementos anteriores, al rotar las curvas, los valores resultantes en (3.1) siguen siendo los mismos. Por lo tanto, el código de cadena de la curva rotada sigue siendo el mismo.

Es claro que la traslación no modifica la codificación de la curva, por lo que también es invariante ante traslaciones.

Cuando se toma el código *5ODCCC* de una curva cerrada el código depende de su vértice inicial, si se toman el código en dos vértices distintos, se obtiene que las cadenas resultantes son iguales bajo un corrimiento o desplazamiento. Por lo que en caso de tener una curva cerrada se utiliza el mismo método que en la Sección 3.1. Es decir, se toma como representante a la cadena que representa un menor entero.

Otra propiedad del *5ODCCC* es que para obtener el código de cadena de la curva en sentido opuesto, se puede obtener recorriendo el código de cadena de la curva original en sentido contrario e intercambiando la posición de los bloques que representan los segmentos rectos de la curva (o bloques de 0) con los elementos del código anteriores distintos a 0, como se observa en la Fig. 3.7.

Si se toma la transformación espejo de una curva, el código de cadena resultante de dicha transformación, es el código de cadena de la curva original intercambiando cada instancia de 1 por 3 y de 3 por 1. Esto se da, pues al tomar las transformaciones espejo de las curvas en la Fig. 3.6, los cambios de dirección que representan a 0, 2 y 4 mantiene la misma codificación, mientras que el cambio que representa 1 se transforma en 3 y viceversa.



u a v 230042000124334
v a u 433420001200432

Fig. 3.7: El código de cadena *5ODCCC* de la curva de u a v y de v a u.

3.3. Curvas de barrido y curvas de llenado de espacio

Tanto en $3D$ como en $2D$ una figura digital puede ser descrita a través de una curva, siempre y cuando se pueda encontrar una curva simple que recorra la figura digital. Este tipo de curvas se les conoce en la bibliografía especializada como curvas que llenan el espacio (ver [33]). En el caso de un plano, se pueden encontrar curvas como la *curva Peano* la cual recorre todo el plano sin intersecarse.

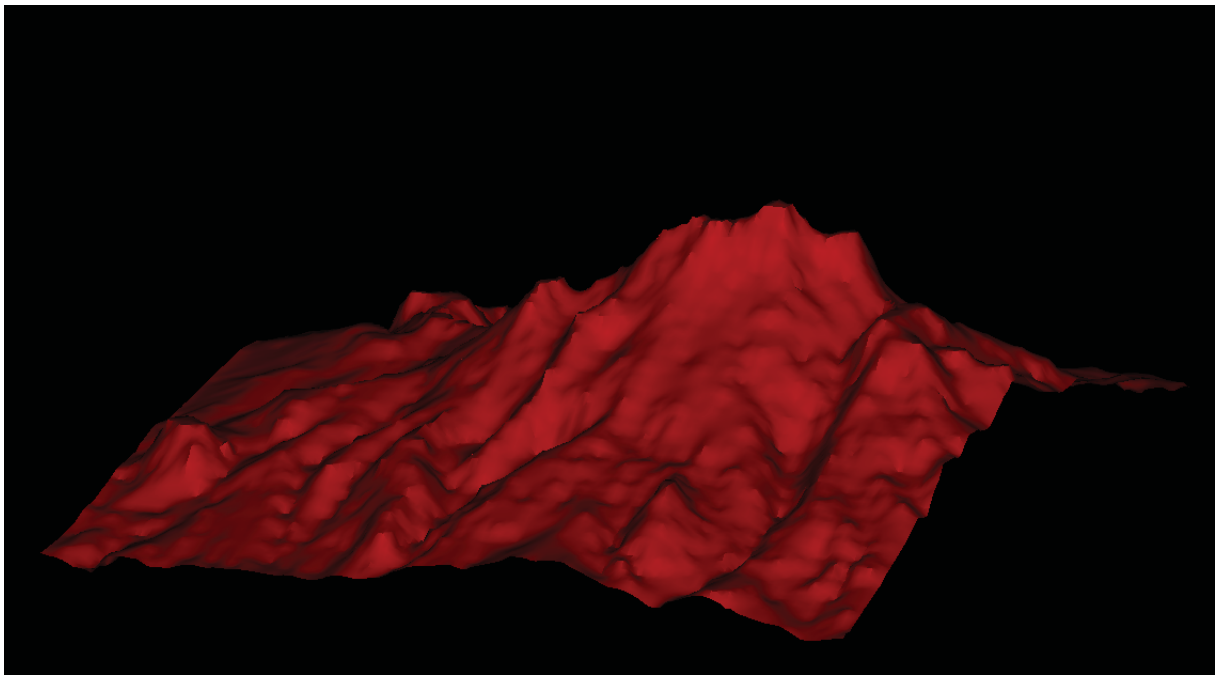


Fig. 3.8: El "*mesh*" de un modelo digital de elevación del Iztacchíuatl.

En [7] se representan superficies discretas utilizando curvas que llenan el espacio y curvas de barrido. De esta forma se puede representar a una superficie a través del código de cadena de la curva de barrido o bien de una curva que llena el espacio.

De igual forma, se puede representar un objeto $3D$ si se encuentra una curva simple que recorra a todos los elementos del objeto. Generando el código de cadena de la curva se obtiene una representación a partir de una cadena de símbolos.

Tanto en el caso de una superficie $3D$ como el de un objeto $3D$, la existencia de la curva no está garantizada. Es decir, en el caso de tener una superficie no isomorfa a un plano, no hay garantía de poder encontrar una curva que recorra toda la superficie. Aunque fuese deseable utilizar una curva de barrido o una curva que llena el espacio $3D$ para poder describir al objeto utilizando el código de cadena, no todos los objetos $3D$ o superficies $3D$ pueden ser descritos a partir de una sola curva.

Algunos de los objetos que si se pueden representar utilizando las curvas de barrido y las curvas que llenan el espacio son todas las superficies que son isomorfas a un plano, como el caso de los modelos de elevación digital (o *DEM* por sus siglas en inglés); como se muestra en [7] en donde utilizan una digitalización del modelo de elevación del "*Iztaccíhuatl*" (ver Fig. 3.8 y Fig. 3.9) y se obtiene el código de cadena de una curva de barrido utilizando el *5ODCCC*.

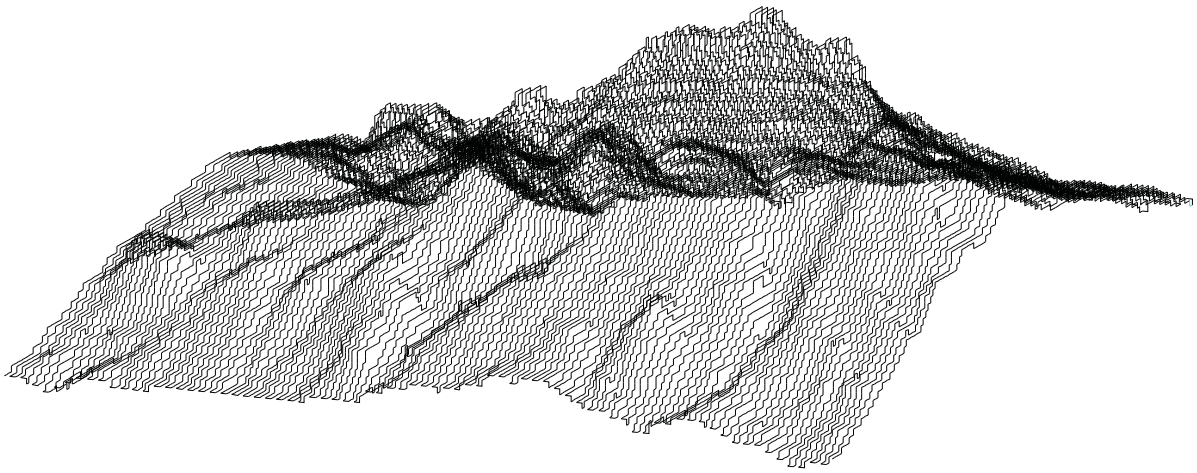


Fig. 3.9: La curva de barrido del *DEM* del Iztacchíuatl en la Fig. 3.8.

Incluso se pueden tomar distintas curvas para la representación de una superficie $3D$ pues se pueden tomar la curva generada por los vértices, la curva de la superficie digital generada a partir de la unión de los centros de las 2-celdas del *DEM*, o bien la curva que une a los centros de los vóxeles. En cualquiera de los casos, si la curva existe, puede ser descrita utilizando el código *5ODCCC*.

3.4. Árboles y el código cambio de dirección ortogonal de 5 direcciones

Una estructura que se puede describir con cierta facilidad utilizando el código *5ODCCC* son los árboles encajados dentro de $\mathbb{Z}^3$. De la Def. 2.7 se observa que la gráfica no tiene ciclos. Por lo que un árbol encajado dentro del espacio $\mathbb{Z}^3$ se puede pensar como la unión de curvas en $\mathbb{Z}^3$.

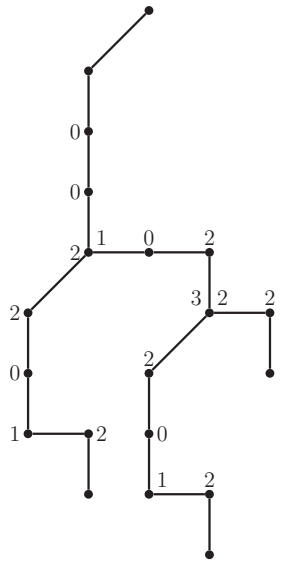
Para describir al árbol usando el código *5ODCCC* se tiene que poder describir las uniones de curvas en $\mathbb{Z}^3$, en [8] se utiliza una forma para poder describir las estructuras de árbol utilizando paréntesis "()" para la descripción de los árboles como uniones de curvas. El uso de paréntesis para representar árboles fue propuesto por Arthur Cayley en [10].

Si se toma a un vértice como raíz del árbol, entonces las curvas que forman al árbol serán curvas que empiezan en el vértice raíz y terminan en un vértice hoja. Es claro que estas curvas se pueden codificar utilizando el código $5ODCCC$; para poder describir la estructura de árbol, el código de cadena debe representar las posibles distintas curvas de la raíz a los vértice hojas. Para hacer tal distinción se introducen paréntesis, si se tiene más de una posibilidad de dirección en la estructura del árbol cada paréntesis contendrá al código de cadena de la curva correspondiente.

Sean $\mathcal{C} = \{u_0, \ldots, u_k\}$ y $\mathcal{C}' = \{v_0, \ldots, v_l\}$ curvas en el árbol de la raíz a las hojas u_k y v_l respectivamente. Supongamos que las curvas son iguales desde su primer elemento hasta el elemento $j \leq k, l$ (esto es que $u_i = v_i \; \forall i \in \{0 \ldots j\}$). El código de cadena debe representar que las curvas son iguales desde la raíz hasta la dirección que son distintas. Por lo que si $a_0 \ldots a_{k-1}$ y $b_0 \ldots b_{l-1}$ son los códigos de cadena que representan a las curvas $\mathcal{C}$ y $\mathcal{C}'$ respectivamente, entonces $a_i = b_i$ con $\forall i \in \{0 \ldots j\}$.

Si $a_{j+1} < b_{j+1}$ el código de cadena que representa a ambas curvas sería:

$$a_1 a_2 \ldots a_j (a_{j+1} a_{j+2} \ldots a_k)(b_{j+1} b_{j+2} \ldots b_l). \tag{3.2}$$



00(102(22)(32012))(22012)
0(0(1(0(2(2(2)))(3(2(0(1(2)))))))(2(2(0(1(2))))))

Fig. 3.10: Un árbol y su código de cadena.

En la Fig. 3.10 se observa el código de cadena que representa al árbol en la figura. En este ejemplo se muestran las dos posibles versiones del código de cadena con la notación de paréntesis. En la primera, cada vez que un vértice tiene más de un hijo se abren paréntesis para colocar dentro de éstos al código de cadena que representa a la curva a partir del vértice hijo en adelante en dirección de las hojas de la estructura de árbol. En la segunda, simplemente se introducen los paréntesis por cada arista que se tiene en el árbol.

Para obtener el código de cadena que representa a un árbol G con raíz r con la representación mínima de paréntesis se realiza el siguiente procedimiento:

Sea $5ODCCC(\vec{u}, \vec{v}, \vec{w})$ la función que asigna el código a las direcciones $\vec{u}, \vec{v}, \vec{w}$ (ver (3.1)). Sea $\delta_G(v)$ la vecindad del vértice v en G y $\mathcal{H}^r_G(v)$ los hijos del vértice v en el árbol G con raíz en r. $\overrightarrow{uv}$ es la dirección del vértice u al vértice v. $\mathcal{P}(u)$ es el padre de u.

1: $\mathcal{V} = \emptyset$
2: $\mathcal{V}' = \emptyset$
3: $Hijos_{cc} = \emptyset$
4: $\mathcal{S} =$ "" (cadena vacía).
5: Opcional se añade a r una manija (ver Sección. 3.2)
6: Se añade r a $\mathcal{V}$.
7: **while** $\mathcal{V} \neq \emptyset$ **do**
8: **for** $u \in \mathcal{V}$ **do**
9: Quitar u de $\mathcal{V}$.
10: **if** $\mathcal{H}^r_G(u) == 1$ **then**
11: $w \in \mathcal{H}^r_G(u)$.
12: $cod = 5ODCCC(\overrightarrow{uw}, \overrightarrow{u\mathcal{P}(u)}, \overrightarrow{\mathcal{P}(u), \mathcal{P}(\mathcal{P}(u))})$.
13: Introducir cod en $\mathcal{S}$ donde corresponde.
14: Insertar w en $\mathcal{V}'$.
15: **else if** $\mathcal{H}^r_G(u) > 1$ **then**
16: **for** $w \in \mathcal{H}^r_G(u)$ **do**
17: $cod = 5ODCCC(\overrightarrow{uw}, \overrightarrow{u\mathcal{P}(u)}, \overrightarrow{\mathcal{P}(u), \mathcal{P}(\mathcal{P}(u))})$.
18: Inserta cod en $Hijos_{cc}$.
19: Inserta w en $\mathcal{V}'$.
20: **end for**
21: Ordena $Hijos_{cc}$.
22: Ordena $\mathcal{V}'$ con el orden inducido en $Hijos_{cc}$.
23: **end if**

24: $\forall s \in Hijos_{cc}$ se introducen en $\mathcal{S}$ como "(s)" donde corresponde.
25: $Hijos_{cc} = \emptyset$.
26: **end for**
27: $\mathcal{V} = \mathcal{V} \cup \mathcal{V}'$
28: **end while**

Como ejemplo, para la Fig. 3.10 el código generado en los distintos pasos es

0
00
00(1)(2)
00(10)(22)
00(102)(220)
00(102(2)(3))(2201)
00(102(22)(32))(22012)
00(102(222)(320))(22012)
00(102(222)(3201))(22012)
00(102(222)(32012))(22012)

Por simplicidad, al implementar el algoritmo que genera el $5ODCCC$ del árbol, cada arista a considerar en cada paso se representa dentro del código como "(c)" con $c \in \{0, 1, 2, 3, 4\}$ dentro del paréntesis correspondiente. Cada arista se encuentra dentro de un paréntesis donde el vértice añadido es hijo del vértice anterior.

Como ejemplo, se presenta el código resultante en cada paso utilizando a "(c)" con $c \in \{0, 1, 2, 3, 4\}$ para representar a cada arista en el árbol de la Fig. 3.10 queda como:

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

No todas las estructuras de árbol en $3D$ pueden ser representadas utilizando el *5ODCCC*, sólo aquellas donde los vértices del árbol tengan a lo más seis aristas incidentes a los vértices. Dichas estructuras de árbol se encuentran dentro de $\mathbb{Z}^3$ cuando las adyacencias son ortogonales, entonces se pueden representar a las estructura de árbol con el *5ODCCC* siempre que sus adyacencias sean ortogonales.

3.5. Árbol Envolvente

Una estructura de árbol que describe un objeto $3D$ de forma precisa es el árbol envolvente. En [9] Bribiesca, Martínez y Guzman presentan el árbol envolvente de un objeto $3D$, la estructura de árbol está formada por los vértice externos (ver Def. 2.35) y algunas de las aristas externas de la figura (ver Def. 2.36).

Para construirlo se toma un vértice externo del objeto a partir del cual se generará un árbol, este vértice es el vértice raíz del árbol envolvente. A partir del vértice raíz se obtienen los hijos del vértice y se toman aquellos que sean vértices externos unidos a través de aristas externas. Esto se repite hasta que todos los vértices externos de la componente conexa por curvas estén incluidos en el árbol.

A continuación se describe de forma explicita la construcción del árbol envolvente. Sea $\mathcal{O} = \{b_1, b_2 \dots b_n\}$ un objeto voxelizado y $\{b_i\}_{i=1}^{n}$ sus vóxeles, r un vértice externo de $\mathcal{O}$. $\mathcal{H}(v)$ una función que obtiene a los hijos de un vértice v a través de aristas externas. θ una función que ordena a los vértices. Sean $\mathcal{L}$ y $\mathcal{V}$ listas vacías de vértices.

El árbol envolvente con raíz r de $\mathcal{O}$ se construye de la siguiente forma:

- Caso inicial: A partir del vértice raíz.

  1: $\mathcal{L} = \emptyset$
  2: $\mathcal{V} = \emptyset$
  3: $r \in \mathcal{V}$
  4: $\mathcal{S} = \mathcal{H}(r)$
  5: $\theta(\mathcal{S})$
  6: **for** $v \in \theta(\mathcal{S})$ **do**
  7: Insertar a v en $\mathcal{L}$ y $\mathcal{V}$
  8: **end for**
  9: $\mathcal{S} = \emptyset$

- Caso general: La lista $\mathcal{L} \neq \emptyset$.

while $\mathcal{L} \neq \emptyset$ **do**
  Quitar al elemento v de $\mathcal{L}$
  $\mathcal{S} = \mathcal{H}(v)$
  Se ordenan $\theta(\mathcal{S})$
  for $v_s \in \theta(\mathcal{S})$ **do**
    if $v_s \notin \mathcal{V}$ **then**
      Insertar a v al final de $\mathcal{L}$ y en $\mathcal{V}$
    end if
  end for
end while

Por la construcción del árbol envolvente se observa que éste esta definido por el vértice raíz y por la función θ, la cual ordena a los vértices que se agregan. Por lo tanto, el árbol envolvente de un objeto es el mismo cuando se toma el mismo vértice raíz y el mismo orden en los vértices añadidos al árbol. Lo que implica que no hay un único árbol envolvente para un objeto.

Para generar el árbol envolvente de una forma única basta con determinar un vértice raíz y el orden a seguir. Para escoger el vértice inicial se le asigna una característica geométrica como puede ser el vértice mayor en la coordenada x, ser el primer vértice en el eje mayor, el primer vértice en el eje menor, u otra. De forma tal que determine al vértice raíz de forma única.

Para determinar el orden para los vértices hijos en los vértices a añadir en el árbol, se va a utilizar el orden inducido por el *5ODCCC*. Para generar el *5ODCCC* de un vértice padre al vértice hijo son necesarias tres direcciones, dichas direcciones se toman utilizando los ancestros de los vértice padre (ver Def. 2.12).

Como se ve en la Sección 3.2 para asignar el código correspondiente a una curva desde su inicio se le añade una manija. En el caso de los árboles en [9] se le añade una manija al vértice raíz de la misma forma que se hace con las curvas. La dirección de la manija determina el orden en que se añaden los hijos del vértice raíz al árbol, por lo que también se determina al árbol (ver Fig. 3.11). De esta forma, se obtiene el código de cadena del árbol envolvente que es de forma similar al ejemplo en la Fig. 3.10

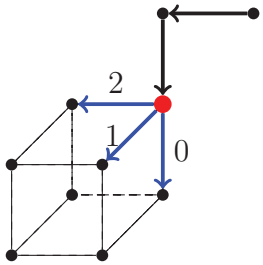


Fig. 3.11: Vértice raíz con direcciones originales.

3.5.1. Disminuir los posibles árboles

Para que la dirección de la manija no sea determinante en la formación del árbol envolvente, ésta es eliminada y así el árbol envolvente estará determinado únicamente por el vértice de inicio. Al determinar el orden de los hijos usando las propiedades geométricas del vértice raíz, se genera una forma canónica para determinar al árbol envolvente.

El orden a tomar para la construcción del árbol es determinado por la geometría del vértice raíz, durante la investigación se determinó que para definir el orden de una forma simple era necesario que el vértice raíz fuese un vértice completamente externo, es decir, que dentro de la vecindad de vóxeles del vértice, únicamente se encuentre un vóxel encendido. Esta propiedad implica que el vértice raíz tenga a lo más tres vértices hijos, los cuales todos pertenecen al mismo vóxel. El código de cadena que se le asigna a cada vértice hijo del vértice raíz es el 1 (ver Fig. 3.12), mientras que el orden para generar el árbol envolvente es inducido por el marco de referencia que da el vértice y el vóxel encendido en sentido levógiro como se muestra en la Fig. 3.13

Las direcciones necesarias para generar el código *5ODCCC* de los vértice que son descendientes de los vértices hijos del vértice raíz estarán dadas por las direcciones para que el primer elemento sea 1 (ver Fig. 3.14) y que se encuentren dentro del vóxel encendido del vértice raíz. Los vértices subsecuentes ya no son influenciados por las direcciones supuestas. Por lo que el árbol se sigue construyendo usando el *5ODCCC*. Tomando el árbol envolvente de esta forma, se limitan los posibles árboles envolventes generados a partir del mismo vértice raíz y la geometría del vértice resulta determinante en la formación del árbol envolvente.

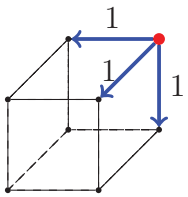


Fig. 3.12: Vértice raíz y direcciones modificadas.

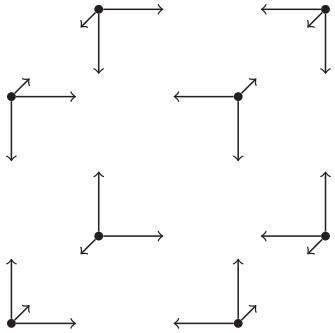


Fig. 3.13: Los distintos vértices en un vóxel con sus marcos de referencia.

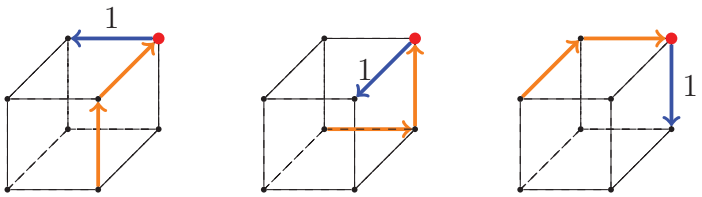


Fig. 3.14: Direcciones a considerar para los códigos de cadena de los elementos subsecuentes.

3.5.2. Objeto o superficie

En la Sección 3.5 se define el árbol envolvente de la superficie externa de un objeto voxelizado, es decir, se toma la superficie externa del objeto; ésta representa al objeto, y al tomar el árbol envolvente, que es una representación de la superficie, éste es una representación del objeto.

La superficie en estos casos es una superficie cerrada (ver Def. 2.42), similar a la superficie de una esfera voxelizada. Al tomar una superficie cerrada, el algoritmo para encontrar el árbol envolvente se termina cuando ya no encuentra vértices externos a través de aristas externas.

Si el objeto a representar no es un objeto con volumen, en este caso una superficie, el árbol envolvente debe representar a la superficie y no al objeto voxelizado. Cuando se tiene una superficie voxelizada se está añadiendo un volumen a la superficie y al tomar el árbol envolvente éste no representaría a la superficie voxelizada, es decir se puede tomar el árbol envolvente de la superficie exterior de una superficie voxelizada pero se estaría añadiendo información artificialmente al objeto.

En este trabajo no se utilizan objetos voxelizados, sino superficies voxelizadas, formalmente bajo la Def. 2.40 una superficie voxelizada no es una superficie digital. Para que el árbol envolvente represente a la superficie voxelizada como una superficie, la condición de paro al formar el árbol envolvente es modificada. La condición se cambia con la intención de evitar añadir vértices extras que pudiesen añadir información ya codificada dentro del árbol.

La condición de paro modificada evita regresar a un vóxel ya recorrido, de esta forma se asegura no añadir elementos al árbol envolvente que puedan formar vóxeles de más, evitando añadir volumen a la superficie. La condición funciona para superficies voxelizadas simples y orientables, pues si se toman los vértices del árbol envolvente y las 2-celdas generadas por éstos, se generan una superficie digital acorde a la Def. 2.40.

3.6. Árbol de Bordes

Otra de las estructuras que se puede representar utilizando el *5ODCCC* es el árbol de bordes. El árbol de bordes se presenta en [25], el cual toma como vértices del árbol a los centros de los vóxeles que se encuentran en la superficie exterior de una figura voxelizada y también se encuentren en un borde.

Para determinar los vóxeles que se encuentran en un borde de un objeto voxelizado es necesario encontrar los planos del objeto voxelizado y tomar la intersecciones de dichos planos. Los vóxeles que se encuentran en más de un plano, son los que se consideran pertenecientes a los bordes del objeto y son los vértices del árbol de bordes.

Para encontrar el árbol de bordes, éste se construye de forma similar al árbol envolvente. Se toma un vóxel inicial perteneciente a un borde y se van recorriendo los bordes a través de los vóxeles. El árbol termina cuando no hay más vóxeles que se puedan recorrer.

El árbol de bordes al ser una estructura en $\mathbb{Z}^3$ y al tomar adyacencia 6 en los vóxeles puede ser descrito con el *5ODCCC* como en la Sección 3.4.

Al tomar el árbol envolvente en vóxeles y el árbol de bordes en vóxeles, es claro que el árbol de bordes tendría menor número de vértices. Si un vóxel es un vóxel interno de un plano, éste no se encuentra dentro del árbol de bordes pero si dentro del árbol envolvente. Todos los vértices del árbol de bordes son vértices del árbol envolvente.

El árbol envolvente es una representación de la superficie voxelizada con la información de la geometría local de la superficie codificada dentro de éste.

3.6.1. Árbol de bordes en superficie

Al árbol de bordes presentado en [25] se le puede adecuar de forma similar al árbol envolvente para representar a una superficie voxelizada. El árbol de bordes tendrá un menor número de vértices que el árbol envolvente pero conservando la geometría de la superficie voxelizada, pues los elementos que se eliminan son elementos internos de planos digitales eliminando así vértices que no aportan a la complexidad de la geometría local de la superficie.

Construcción de árbol de bordes en superficie

Para construir el árbol de bordes en una superficie voxelizada es necesario determinar que elementos pertenecen a los bordes. A diferencia del árbol presentado en [25], el árbol que se construye utiliza los vértices de los vóxeles. Para determinar que vértices pertenecen a los bordes no es suficiente con determinar a los vóxeles que pertenecen a los bordes, es necesario determinar a las 2-celdas que generan la superficie digital de la superficie voxelizada y después encontrar los vértices que se encuentran en los bordes de la figura.

Para determinar cuales 2-celdas se encuentran dentro de la superficie digital generada por la superficie voxelizada, se genera un algoritmo a partir de la gráfica de 6-adyacencia de la superficie voxelizada.

El algoritmo empieza en una cara de la superficie digital y va generando la superficie digital trasladando la 2-celda a partir de las aristas de la gráfica de adyacencia. Es posible trasladarse a través de la arista de la gráfica de adyacencia si ésta no representa una adyacencia interna o bien una adyacencia contraria a la 2-celda que se desea mover. De esta forma, se garantiza que todos los vértices que se toman como parte de la superficie digital, se encuentren en una 2-celda y que todas las aristas tengan al menos un movimiento paralelo y se evita añadir 2-celdas que formen un vóxel. Las superficies generadas de esta forma son las superficies evaluadas en el presente trabajo, éstas se muestran en las figuras en Fig. 3.15 y Fig. 3.16.

Teniendo la superficie digital generada de esta forma, se puede clasificar a los vértices de ésta para determinar si se encuentran dentro de un borde.

A cada vóxel dentro de la superficie voxelizada se le encuentra el plano digital al cual pertenece, generando una partición de los elementos de la superficie voxelizada. La partición anterior induce una partición en las 2-celda de la superficie digital. Para cada 2-celda se selecciona su clase a partir del vóxel dentro de la superficie voxelizada, la asignación es con el mismo plano digital al cual pertenece el vóxel. Se encuentran todas las posibles 1-celdas de la superficie hecha a partir de las 2-celdas y se les asigna el plano digital al cual pertenecen.

Como el interés son todas las posibles 1-celdas de la 2-celdas de los bordes, se toman únicamente aquellas que se encuentran en más de un plano, generando así una clasificación para las 1-celdas. Para las 1-celdas clasificadas dentro de un borde se toman las 0-celdas que serán los vértices del árbol de bordes y se construye el árbol correspondiente.

El árbol resultante tendrá un menor número de vértices que el árbol envolvente y será un descriptor de la superficie digital de la superficie voxelizada. Al tomar los vértices de los bordes se eliminan los vértices simples de la superficie digital, es decir, aquellos que se consideran como vértices internos de la segmentación en planos de la superficie digital.

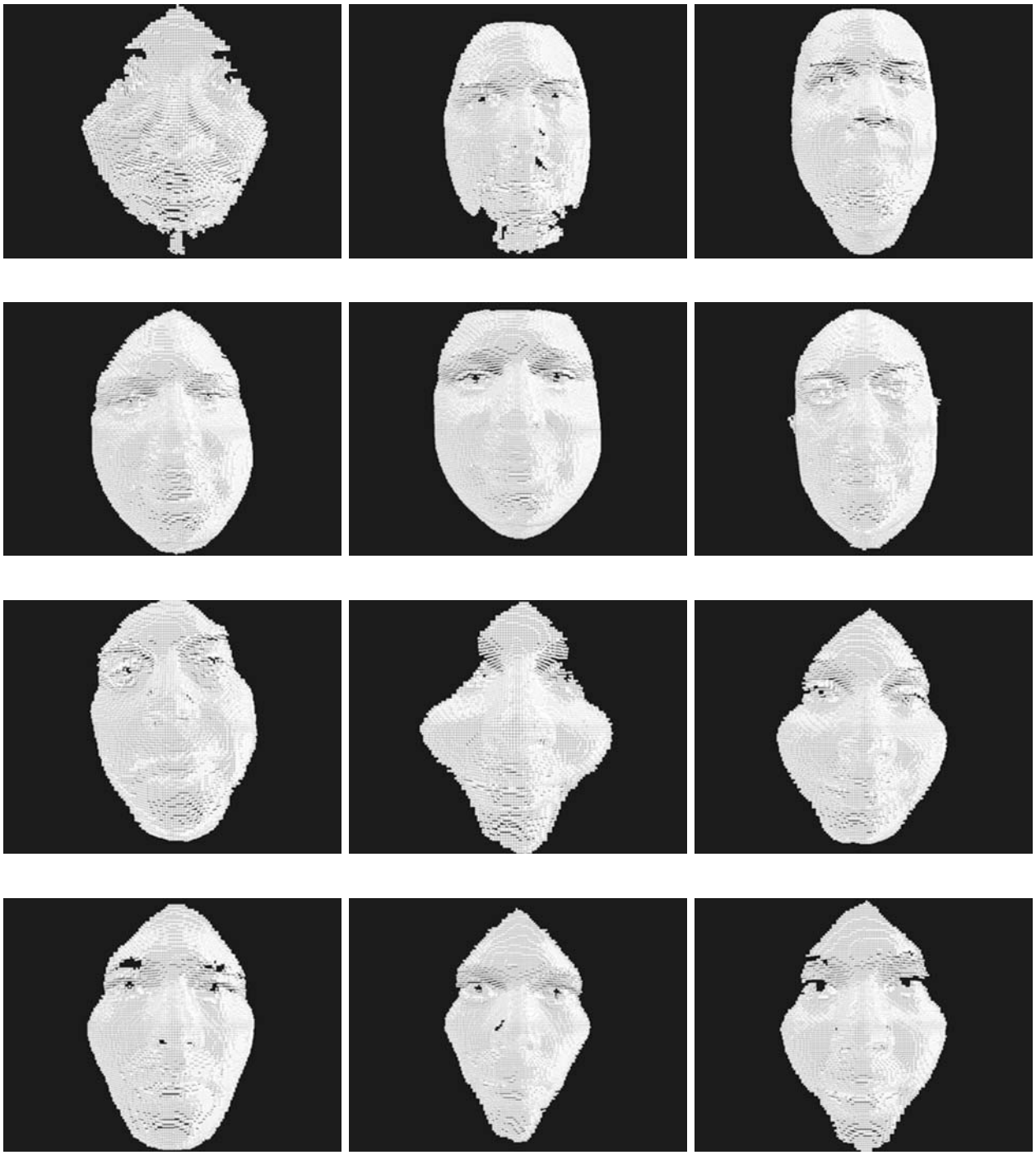


Fig. 3.15: Superficies digitales extraídas a partir de superficies voxelizadas.

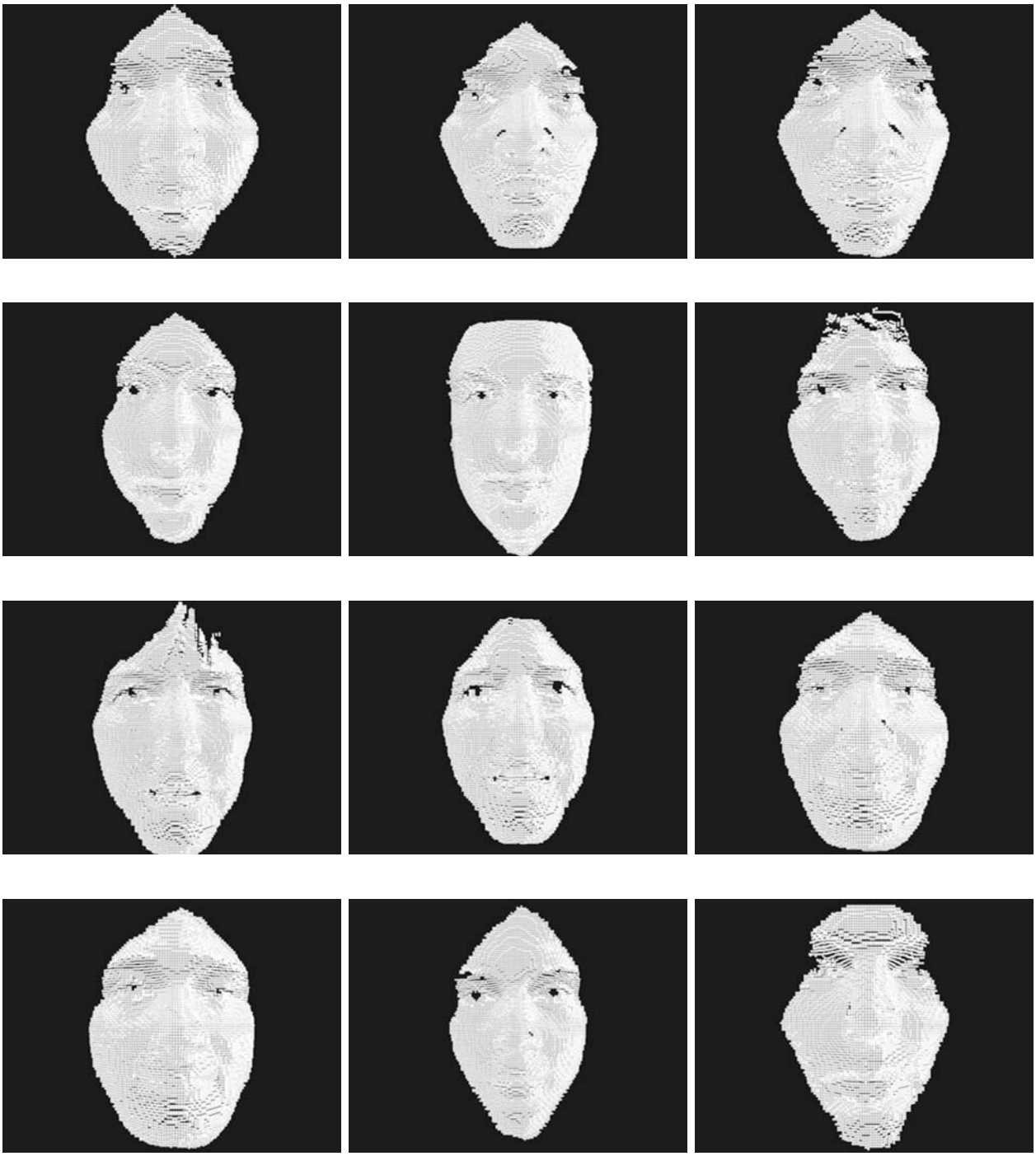


Fig. 3.16: Superficies digitales extraídas a partir de superficies voxelizadas. 2.

Capítulo 4

Compresión

Para almacenar una superficie en $3D$ existen distintos métodos, entre los cuales el más utilizado es el conocido como "*mesh*" o mallas, la superficie se triangula o poligoniza, y se almacenan las coordenadas de los vértices que generan la triangulación o poligonalización además de las adyacencias entre los vértices para que generen los triángulos o polígonos.

Existen métodos para comprimir superficies almacenadas como las mallas trianguladas o mallas poligonales, en [2] y [31] se hacen estudios sobre los algoritmos de compresión de superficies, principalmente enfocados a mallas triangulares o poligonales. La revisión más actual sobre la compresión de superficies $3D$ se encuentra en [24].

La forma más común para comprimir superficies almacenadas como mallas poligonales es generar una triangulación de los polígonos de la malla y comprimir la triangulación a partir de métodos de compresión para mallas triangulares. Este tipo de compresión aumenta el número de vértices y es una compresión con pérdida, pues no se puede regresar a la forma original de la malla poligonal (ver [2]).

Los métodos de compresión más comunes para mallas triangulares regulares utilizan las adyacencias locales para el almacenamiento de la malla, una forma común dentro de los algoritmos de compresión es utilizar árboles generadores de la gráfica de adyacencia de la malla y eliminar las coordenadas especificas de cada vértice (ver [24]).

El caso para superficies voxelizadas no es común dentro de la literatura especializada (ver [24]), por lo que el presente trabajo se considera como un trabajo novedosos y aporta al desarrollo del área.

Para las superficies voxelizadas, la forma más simple de guardar la superficie sería hacer su almacenamiento como una imagen digital dentro del espacio $\mathbb{Z}^3$. Para almacenar la imagen digital, sin pérdida de generalidad, lo más simple sería considerar la imagen como una matriz de $l \times m \times n$ como representación del espacio $\mathbb{Z}^3$, donde l, m, n son las dimensiones de la imagen binaria donde se encuentra la superficie voxealizada. Al ser una imagen binaria para cada elemento de la matriz le correspondería 1 si el vóxel se encuentra encendido y 0 si se encuentra apagado. Es claro que al almacenar la superficie voxelizada de esta manera la cantidad de bits necesarios para su almacenamiento sería $l \cdot m \cdot n$, sin importar el tamaño de la superficie voxelizada. Lo que se considera como una forma poco eficiente para almacenar una superficie voxelizada.

Al considerar superficies voxelizadas y no objetos $3D$ voxelizados, en la mayoría de los casos, la cantidad de información necesaria para almacenar la superficie es menor si sólo se almacenan las coordenadas de los centros de los vóxeles de la figura digital. Al trabajar dentro del espacio $\mathbb{Z}^3$ las entradas de las coordenadas serán elementos enteros y se almacenan a éstas como una lista de vectores de tamaño tres. Al ser la forma más común para almacenar un objeto dentro del espacio $\mathbb{Z}^3$ la representación de centros de los vóxeles será la base de la comparación con respecto a otras representaciones.

Para hacer las comparaciones de la cantidad de datos necesarios para almacenar cada representación, este capítulo detalla la forma de calcular la cantidad de datos para almacenar cada una de las representaciones presentadas con anterioridad y que a su vez son utilizadas en el presente trabajo.

En [34] se hace el análisis de algunas formas de almacenar los objetos $3D$ las cuales se explicaran en este capítulo.

4.1. Tamaño de los datos

En la siguiente sección se muestran como se calcula la cantidad de datos necesarios para almacenar los distintos métodos para representar una superficie voxelizada. Para cada una de las distintas representaciones se hace un análisis de como se almacena la representación y de como se obtienen los datos necesarios para su almacenamiento, usando dicha información en el Capítulo 5 se dan los resultados de las comparaciones entre los distintos métodos.

4.1.1. Tamaño para el centro de vóxeles

Como se vio en la Sección 2.4, un vóxel tiene 8 vértices, 12 aristas y 6 caras. Para hacer referencia a un vóxel es más simple tomar el centro de éste, y tomar las coordenadas en $3D$ que lo representan. La representación de los centros de los vóxeles en sí es una forma de compresión. Para tomar la representación de éstos, se toman las coordenadas enteras que representan a los centros de los vóxeles.

Para almacenar un objeto utilizando los centros de los vóxeles que pertenecen al objeto, la cantidad de información a utilizar dependerá de la resolución del objeto. Para representar un vóxel a partir de las coordenadas de los centros de los vóxeles en una imagen de tamaño $l \times m \times n$ se utiliza un vector de coordenadas (a, b, c) donde $a \in \{0, \ldots, l-1\}$, $b \in \{0, \ldots, m-1\}$ y $c \in \{0, \ldots, n-1\}$. El número de bits necesario para almacenar al vector es la suma del número de bits necesario para guardar la terna a, b, c. Para guardar a un elemento de $\{0, \ldots, l-1\}$ el número de bits necesarios es igual al número de bits que se necesitan para representar a l en binario. Por lo tanto, el número de bits para almacenar la terna a, b, c es igual al número de bits necesario para guardar la terna l, m, n . Por lo que si $2^{k_l-1} < l \leq 2^{k_l}$, $2^{k_m-1} < m \leq 2^{k_m}$ y $2^{k_n-1} < n \leq 2^{k_n}$ el número de bits para guardar la terna es $k_l + k_m + k_n$.

Si un objeto tiene N vóxeles encendidos dentro de una imagen con resolución $l \times m \times n$, entonces el número de bits para guardar al objeto sera

$$(k_l + k_m + k_n) \cdot N.$$

Es posible asumir que se puede encajar al objeto dentro de un cubo de tamaño $M \times M \times M$ donde $2^{R-1} < M \leq 2^R$, por el análisis anterior para guardar un centro de vóxel del cubo sería necesario utilizar $3 \cdot R$ bits. Por lo que la fórmula anterior queda como:

$$3 \cdot R \cdot N. \tag{4.1}$$

De esta forma se utilizará (4.1) para calcular el número de bits usados para almacenar un objeto en $\mathbb{Z}^3$ cuando se utilicen los centros de los vóxeles como representación.

Símbolos	Bits
0	000
1	001
2	010
3	011
4	100

Tabla. 4.1: Almacenamiento de *5ODCCC* en bits.

4.1.2. Almacenamiento usando el *5ODCCC*

El número de bits necesarios para guardar una curva usando el *5ODCCC*, claramente depende de la longitud de la curva a guardar. Para guardar cada elemento del código *5ODCCC* de forma simple, el número de bits a utilizar es 3 bits para ser almacenado (como se ve en la Tabla 4.1) por lo que cada cambio de dirección queda almacenado utilizando 3 bits. Si se tiene una curva de longitud $N+1$ elementos, ésta tiene N posibles cambios de dirección y se utilizan $N-2$ elementos para codificar la curva sin añadir una manija al vértice inicial, mientras que se utilizan N en el caso de añadir la manija.

La fórmula para determinar el número de bits a usar en los dos casos distintos queda como:

$$\begin{aligned} \text{Datos usados} &= (N-2)\cdot 3 \\ \text{Datos usados} &= N\cdot 3 \end{aligned} \tag{4.2}$$

Como se muestra en la Sección 3.4 se puede utilizar el *5ODCCC* para representar una estructura de árbol. Para el caso de almacenaje de una estructura de árbol se le añaden dos símbolos más al código como se ve en la Tabla 4.2, también se utilizan 3 bits para almacenar cada elemento en este nuevo código. Al utilizar el código presentados en las tablas anteriores, al codificar la Fig. 3.10 se obtienen dos códigos de cadena que representan al mismo árbol. Los casos son cuando sólo se abren paréntesis, cuando hay más de un hijo, y el caso en que se abran para cada arista dentro del árbol. Para construir un algoritmo que genere el *5ODCCC* expandido de un árbol es más simple tomar el segundo caso. Al comparar la cantidad de datos usados para almacenar un árbol, es claro que la segunda opción sería la más costosa en términos de bits usados. Por tanto, si la segunda opción muestra que la cantidad de datos a usar es menor que la cantidad de datos usados por la representación de centros de los vóxeles, entonces la primera también.

Símbolos	Bits
0	000
1	001
2	010
3	011
4	100
(	101
)	110

Tabla. 4.2: Almacenamiento de *5ODCCC* para estructuras de árbol.

Al tomar el caso en el cual se abren paréntesis para cada una de las aristas del árbol, se observa que para guardar una arista usando el *5ODCCC* expandido, se utilizarán tres símbolos de éste ((a) donde $a \in \{0, \ldots, 4\}$). De manera que para calcular el número de bits necesarios para almacenar una estructura de árbol, es necesario saber el número de aristas del árbol, las cuales están dadas por la fórmula $V - 1$ donde V es el número de vértices del árbol.

Por lo explicado en la Subsección 3.5.1 se codifican todos los cambios de dirección de un árbol. Si T es un árbol y $|V(T)|$ es el número de vértices de éste, la cantidad de bits necesaria para almacenar al árbol T utilizando el *5ODCCC* queda como:

$$Bits_{5ODCCC}(T) = \underbrace{3}_{bits} \cdot \underbrace{3}_{simbolos} \cdot (|V(T)| - 1)\,. \tag{4.3}$$

Usando (4.3) se calculará el número de bits necesario para almacenar un árbol usando el *5ODCCC* como representación de éste.

4.1.3. Curvas de Barrido

Un método para representar una superficie voxelizada es representar a la superficie utilizando una curva que recorra a todos los elementos de la superficie, este tipo de curvas se las conoce como curvas de barrido. En [7] muestran una forma de representar a una superficie voxelizada utilizando una curva de barrido y como codificarla utilizando el *5ODCCC*.

Por la Def. 2.45 una curva de barrido es una curva digital en $\mathbb{Z}^3$, la cual se puede describir usando el *5ODCCC*. Por tanto, si una curva de barrido tiene longitud $N+1$, usando el número de bits mostrado en la Tabla 4.1 dicha curva usará $3 \cdot N$ bits, como se explica en la Subsección 4.1.2. Generando así que una superficie de tamaño $N + 1$ vóxeles sea representada por $3 \cdot N$ bits.

4.1.4. Gráfica de Adyacencia

Otro método para representar una superficie voxelizada consiste en representarla a través de su gráfica de adyacencia (ver Def. 2.44), para almacenar una gráfica es necesario almacenar los nodos y las adyacencias entre ellos. Como en este trabajo únicamente se utilizan superficies conexas simples, no es necesario guardar los nodos de la gráfica.

Para almacenar la gráfica se utilizarán únicamente las adyacencias de ésta. Y para almacenar las adyacencias de la gráfica sólo se requiere almacenar las aristas. Éstas se almacenarán en parejas, por lo tanto, para calcular la cantidad de datos a utilizar, es necesario saber el número de aristas en la gráfica.

Por consiguiente si una gráfica G tiene $|A(G)|$ aristas y $|V(G)|$ vértices, donde $0 \leq |V(G)| < 2^k$ para referirse a cada vértice, es necesario utilizar k bits, por lo tanto, cada arista necesita $2 \cdot k$ bits para ser almacenada. Entonces la cantidad de información necesaria para almacenar la gráfica G queda como:

$$Bits_{Graf}(G) = 2 \cdot \underbrace{k}_{\text{vértices}} |A(G)|. \tag{4.4}$$

Es claro que la cantidad de información necesaria para almacenar la gráfica G depende tanto de su número de vértices, como del número de adyacencias de cada vértice. Para calcular el número de bits será necesario obtener toda la gráfica y ver las aristas de ésta.

4.2. Almacenar una Superficie

Para almacenar una superficie voxelizada existen distintos métodos, este trabajo compara los presentados con anterioridad, con el fin de encontrar las ventajas y desventajas de cada uno. Como se ha mencionado el método de referencia para hacer la comparaciones será el almacenamiento de los centros de lo vóxeles pues es la forma de almacenaje que se considera más simple.

En esta sección se hace la descripción de como influye la geometría local en la cantidad de datos para almacenar el árbol envolvente. Para calcular la cantidad de datos del árbol envolvente se utiliza (4.3).

4.2.1. Almacenamiento de superficies usando el árbol envolvente

Mediante (4.3) podemos calcular el número de datos usados para almacenar una superficie hecha de vóxeles usando el *5ODCCC* que codifica al árbol envolvente de la superficie, sería únicamente necesario saber el número de vértices exteriores del objeto; sin embargo por lo visto en la Subsección 3.5.2 no se puede tomar simplemente los vértices exteriores de la superficie voxelizada y calcular a partir de éstos el número de bits necesario para almacenar el árbol envolvente usando el *5ODCCC* pues se tomarían vértices de más que no corresponden al árbol.

Calcular el número de vértices del árbol no depende directamente del número de vóxeles de la superficie a considerar, es decir, no hay una fórmula para obtener el número de vértices que el árbol envolvente recorre.

Para ejemplificar esta afirmación se toma el árbol envolvente de la superficie voxelizada en la Fig. 4.1a y cuyo árbol envolvente se encuentra en la Fig. 4.1b. Para mostrar que el árbol envolvente no depende del número de vóxeles de la figura de una manera constante a la Fig. 4.1a se le añade un vóxel en distintas partes (Fig. 4.1c, Fig. 4.1e y Fig. 4.1g) y se obtienen los árboles envolventes correspondientes empezando desde el mismo vértice.

Como muestra la Fig. 4.1a la superficie tiene 26 vóxeles, se toma el árbol envolvente de la superficie y éste se muestra en la Fig. 4.1b con el vértice raíz marcado en rojo. El cual tiene 52 vértices y 51 aristas.

A la Fig. 4.1a se le agrega un vóxel en distintos lugares y se obtienen los árboles envolventes de las nuevas superficies. Se observa que el árbol envolvente de la superficie en la Fig. 4.1c, el cual se muestra en la Fig. 4.1d, tiene 56 vértices y 55 aristas. La superficie en la Fig. 4.1e también tiene un vóxel más que la Fig. 4.1a y tiene el árbol envolvente en la Fig.4.1f, el cual tiene 53 vértices y 52 aristas. Por último la superficie en Fig.4.1g con árbol envolvente en la Fig. 4.1h tiene 57 vértices y 56 aristas.

Todos los árboles son construidos a partir del mismo vértice y tienen el mismo número de vóxeles. El número de aristas es distinto en cada árbol, se puede observar que la geometría local de cada superficie afecta al número de elementos de que aparecen en el árbol envolvente. El número de bits usados dependerá de geometría local de la superficie, es decir, dos superficies pueden tener el mismo número de vóxeles, pero los árboles envolventes pueden ser de distinto tamaño. En consecuencia, los códigos de cadena que representan a las superficies son de distintas longitudes.

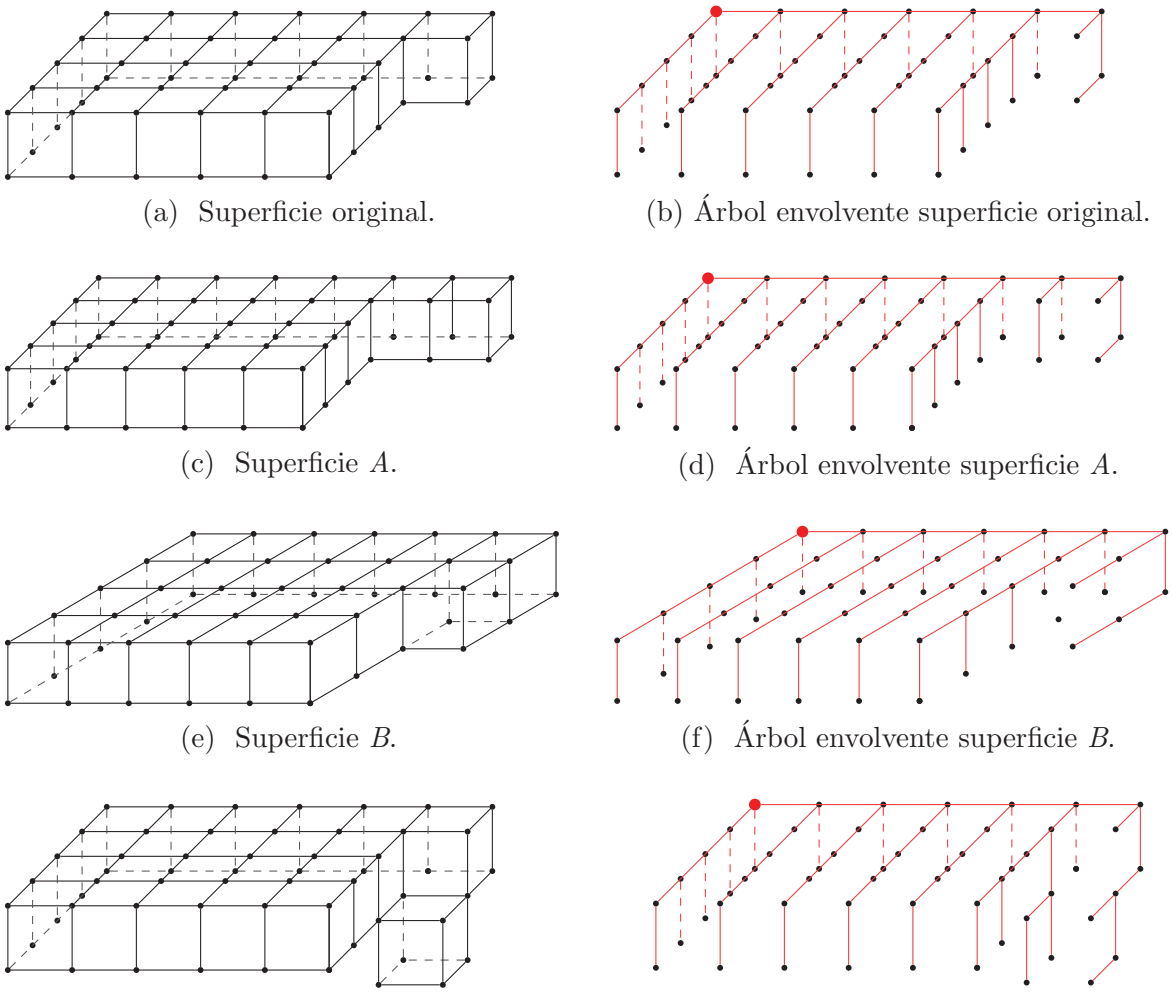
Como no se puede determinar *a priori* el tamaño del árbol envolvente en las superficies estudiadas, para obtener los datos fue necesario generar el árbol envolvente en su totalidad. Y a partir de éste generar el código de cadena, para finalmente calcular el número de bits usados en el almacenamiento del árbol envolvente de la superficie.

4.3. Compresión de datos

El trabajo realizado se basa en buscar sí el árbol envolvente codificado usando el *5ODCCC* sobre una superficie voxelizada presenta una compresión en la cantidad de información necesaria, comparándolo con el almacenamiento de la misma superficie usando las coordenadas que representan los centros de los vóxeles.

Se usará como base de comparación el almacenamiento de las coordenadas de los centros de los vóxeles, por ser una de las formas más comunes de almacenamiento de una figura voxelizada. Aunque existen otras formas de almacenamiento, se considera que la comparación en contra de ésta es una comparación "justa". Es decir, aunque existen formas de representación de objetos voxelizados que utilizan menos información que las representaciones presentadas, éstas son muy sensibles al ruido o bien para obtener información local de un elemento de la superficie es necesario generar todo el objeto.

Aunque el trabajo incluye la representación de la codificación de una curva de barrido en una superficie voxelizada usando el *5ODCCC*, es claro que la cantidad de información usada por esta representación sería mínima, sin embargo, la representación es muy sensible al ruido y para encontrar información local de los elementos de la superficie se tiene que generar toda la superficie. Adicionalmente, no hay garantía de que la curva de barrido exista, por lo que no puede ser generada para todas las superficies.



(a) Superficie original. (b) Árbol envolvente superficie original.

(c) Superficie A. (d) Árbol envolvente superficie A.

(e) Superficie B. (f) Árbol envolvente superficie B.

(g) Superficie C. (h) Árbol envolvente superficie C.

Fig. 4.1: Distintos ejemplos de superficies y sus árboles envolventes.

En contraste, el árbol envolvente puede ser generado a partir de cualquier vértice y su existencia está garantizada para cualquier figura voxelizada. Aunque las modificaciones hechas a la estructura del árbol envolvente son únicamente para reducir el número de posibles árboles y eliminar información redundante al considerar superficies vóxelizadas (ver la Subsección 3.5.1), éstas no modifican la existencia del árbol envolvente.

Se considera que la codificación del árbol envolvente usando el *5ODCCC* es novedosa, pues a un objeto tridimensional (superficie voxelizada) se le asigna una representación de una dimensión (el código *5ODCCC*) que representa al árbol envolvente. Si esta representación genera una disminución en la cantidad de información utilizada, implica que la representación del *5ODCCC* del árbol envolvente es una representación más compacta de la superficie en comparación con la representación de los centros de los vóxeles, pues usa una menor cantidad de información para su almacenamiento y muestra una menor sensibilidad al ruido con respecto a la curva de barrido.

4.4. Compresión de Huffman

Al generar una cadena de símbolos que representa una superficie voxelizada, ésta se puede comprimir aun más utilizando técnicas de compresión clásica (ver [38]). Aunque existen muchos métodos de compresión para cadenas, en el presente trabajo se utiliza únicamente la compresión de Huffman con la intención de ejemplificar que se puede comprimir la cadena de *5ODCCC* que representa al árbol usando compresiones básicas de cadenas. Si se desease usar compresiones clásicas de cadenas sobre la representación de los centros de los vóxeles, la compresión sería mínima, pues la efectividad de la compresión no es la misma que al utilizar las compresiones sobre la cadena de *5ODCCC*. Lo que se resalta, no es la compresión como tal, si no la posibilidad de que se pueda comprimir aun más y así disminuir la cantidad de datos para almacenar la superficie voxelizada, lo que sería una característica deseable para utilizar dicha forma para representar a una superficie.

La compresión de Huffman se presenta en [22], dentro de la cual se presenta un algoritmo de compresión sin perdida para cadenas generadas dentro de un alfabeto finito. La compresión se establece como una forma para reducir la longitud de las cadenas. Para reducir dicha longitud, el algoritmo establece que los elementos con mayor frecuencia de aparición sean los que menor número de bits utilicen, es decir, que al ser codificados la longitud de éstos sea mínima.

La frecuencia de los elementos del alfabeto se establece con anterioridad a partir de las cadenas a codificar, si son palabras dentro de un lenguaje se obtendría la frecuencia de las letras del abecedario que aparecen en el lenguaje. El algoritmo utiliza la frecuencia de aparición dentro de las cadenas a comprimir, utilizando la frecuencia como probabilidad de aparición, se genera una codificación óptima, la cual se puede representar a partir de un árbol. Utilizando dicha codificación, la cadena original se codifica y la entropía de la cadena es reducida generando una compresión, pues la longitud promedio de las palabras ha sido reducida.

4.4.1. Construcción del código de Huffman

El código de Huffman puede establecerse como un código binario, es decir, es una función de $H\colon \mathcal{A} \to \{0,1\}^*$, recordando que $\mathcal{B}^* = \cup_1^\infty \mathcal{B}^n$ donde $\mathcal{B}^n = \{w_1 w_2 \dots w_n | w_i \in \mathcal{B}\}$. Se puede generalizar a otros tipos de códigos pero en este trabajo únicamente se utiliza la forma binaria. De forma que los elementos codificados por el algoritmo de Huffman son palabras de 0 y 1. Si $a \in \mathcal{A}$ la codificación de Huffman de a es $H(a) \in \{0,1\}^*$. Para referirse a la longitud de una palabra $w = b_0 b_1 b_2 \dots b_k$ se denotará con $L(w) = k+1$, formando así una función $L\colon \mathcal{A}^* \to \mathbb{N}$.

Sea $\mathcal{A} = \{a_0, a_1, a_2 \dots a_n\}$ un alfabeto finito donde la probabilidad generada a partir de la frecuencia de aparición es $\{P(a_0), \dots P(a_n)\}$ donde $\sum_{i=0}^{n} P(a_i) = 1$, sin perdida de generalidad, se puede suponer que la probabilidad de aparición esta ordenada, es decir, $P(a_0) \geq P(a_1), \cdots \geq P(a_n)$. El algoritmo de Huffman codifica a los elementos de $\mathcal{A}$ para que el elemento menos probable tenga una longitud mayor y el elemento más probable una longitud menor. En este caso se tendría que si $P(a_i) \geq P(a_j)$, entonces $L(H(a_i)) \leq L(H(a_j))$.

El algoritmo de Huffman busca que los elementos menos probables tengan la mayor longitud pues éstos aparecen con menor frecuencia, tomando los dos elementos menos probables, éstos se intercambian por un elemento nuevo el cual tendrá la probabilidad sumada de los elementos. Para distinguir a los elementos estos tendrán los mismos prefijos, cambiando únicamente el último elemento de la palabra para poder distinguir entre ellos. Aplicando este método se genera un código y a éste se le conoce como el código de Huffman.

Si $P(a_n)$ y $P(a_{n-1})$ son los elementos cuya probabilidad es menor, entonces se genera un nuevo elemento b, este nuevo elemento remplaza a a_n y a_{n-1}, generando así un nuevo alfabeto $\mathcal{A}' = \{a_1, a_2, \dots b\}$ donde la probabilidad de aparición de b es $P(b) = P(a_n) + P(a_{n-1})$; para distinguir entre a_n y a_{n-1} se añade un 0 o un 1. Este proceso se repite hasta que en el alfabeto quede un único elemento que representa a todos los elementos del alfabeto original.

Como ejemplo, se toman a los elementos del *5ODCCC* y se genera el código de Huffman para un caso particular de un árbol. Se toma como alfabeto al código presentado en la Tabla 4.2 con una probabilidad de aparición para los elementos dada por la Tabla 4.3 la cual se genera a partir del código de cadena de un árbol con $41,580$ elementos.

Símbolos	Probabilidad
0	0.141
1	0.0133
2	0.1641
3	0.0112
4	0.0038
(	0.3333
)	0.3333

Tabla. 4.3: Probabilidades de aparición por símbolo.

Ordenando al conjunto de símbolos por su probabilidad se obtiene que el alfabeto queda como $\mathcal{A} = \{(,), 2, 0, 1, 3, 4\}$. Aplicando un paso del algoritmo de Huffman se obtiene el alfabeto ordenado, el cual queda como $\mathcal{A}_1 = \{(,), 2, 0, 1, b_1\}$, donde la probabilidad de $P(b_1) = 0.015$; en el siguiente paso se sustituye b_1 y 1, quedando como $\mathcal{A}_2 = \{(,), 2, 0, b_2\}$ con $P(b_2) = 0.0283$. El siguiente paso se sustituye a b_2 y a 0 con b_3 quedando como $\mathcal{A}_3 = \{(,), b_3, 2\}$ con $P(b_3) = 0.1693$, sustituyendo a b_3 y 2 con b_4 donde $P(b_4) = .3334$ quedando como $\mathcal{A}_4 = \{b_4, (,)\}$. A diferencia de los pasos anteriores, en este se toma a b_5 y sustituye a (y) con $P(b_5) = 0.6666$ y $\mathcal{A}_5 = \{b_5, b_4\}$.

Para generar el código binario se va asignando 0 o 1 a los elementos dentro de los alfabetos, donde los elementos asignados al mismo elemento en los pasos siguientes tienen el mismo prefijo. Sin perdida de generalidad se asigna a b_5 el 0 y a b_4 el 1. Todos los elementos asignados a b_5 tienen el mismo prefijo, en este caso el 0; de la misma forma, todos los elementos que tienen asignado a b_4 tienen como prefijo el 1. De esta manera se van asignando los correspondientes elementos hasta regresar al alfabeto original. El código de Huffman para este caso en particular es mostrado en la Tabla 4.4.

Símbolo	Huffman
(	01
)	00
2	11
0	100
1	1001
3	10000
4	10001

Tabla. 4.4: El código de Huffman usando las probabilidades de Tabla 4.3.

Utilizando el código de Huffman establecido en la Tabla 4.4 se codifica el código de cadena

obtenido para el árbol en la Fig. 3.10

$$(0(0(1(0(2(2(2)))(3(2(0(1(2)))))))(2(2(0(1(2)))))))$$

cuyo resultado queda como:

01100011000110110110001110111011
11100000001101000111011000110 1
10111000000000000000111011101 1
00011011011100000000000000

Si se utiliza el código binario establecido en la Tabla 4.2 el código queda como:

10100010100010100110100010101 0
10101010101011011011010101110 1
01010100010100110101011011011 0
11011011011010101010101010100 0
10100110101011011011011011011 0
110

Como se puede observar ambos, códigos almacenan la cadena del *5ODCCC* del árbol, el código de Huffman resulta tener una longitud de 116 elementos contra 153 del código binario establecido en la Tabla 4.2.

El caso mostrado ilustra la obtención del código de Huffman para una cadena del *5ODCCC* y la compresión obtenida. Esto muestra que se puede comprimir el código *5ODCCC* con métodos de compresión simples y en nuestro caso se pueden comprimir estructuras $3D$.

Capítulo 5

Resultados

Este capítulo contiene los resultados obtenidos del estudio al comparar las distintas representaciones de superficies voxelizadas. Se detalla como se generan las estructuras de las distintas representaciones construidas en la secciones 2.1, 3.3, 3.5 y 3.6. A partir de dichas representaciones se calculan los datos para almacenar las representaciones usando lo visto la Sección 4.1. Con los datos obtenidos se comparan las distintas estructuras y se hace un análisis de los resultados. Parte de estos resultados se encuentran publicados en [34].

5.1. Generación de las superficies voxelizadas

Para comparar las estructuras con las cuales se representan a las superficies voxelizadas, es necesario tener una base de superficies voxelizadas. Para obtener dichas superficies, se utilizan superficies de rostros humanos, obtenidos de una base de datos donde las superficies se encuentran en forma de "*mesh*". Esta base de datos se utiliza en [26] y [30], donde las superficies son superficies trianguladas de rostros.

Para generar las superficies voxelizadas, se utiliza el programa *Binvox* (ver [27]) el cual voxeliza la superficie triangular de los rostros, usando el algoritmo presentado en [29]. De esta forma se genera una superficie voxelizada que representa los rostros humanos.

Otra de las superficies utilizadas es el *DEM* del volcán "*Iztaccíhuatl*", para que éste sea una superficie voxelizada se agregan los vóxeles necesarios para que los saltos en las alturas del *DEM* no generen discontinuidades, obteniendo así una superficie voxelizada.

5.1.1. Generación de la Curva de Barrido

Por lo observado en la Sección 3.3 la existencia de una curva de barrido que recorra toda la superficie no esta garantizada, al no poder garantizar la existencia de la curva en las superficies de rostros humanos voxelizadas, se decide evitar la generación de ésta en dichas superficies. Para tener un marco de referencia con el cual comparar, se utiliza el *DEM* (modelo digital de elevación) del volcán *"Iztaccíhuatl"* y se obtiene la curva de barrido de éste (ver la Fig. 5.1).

Como la curva de barrido existe en un plano, la curva a utilizar corresponde a la curva que bajo la imagen inversa de la proyección al plano XY genera la curva de barrido de un plano. Esta curva es una curva de barrido del *DEM*. Se genera así una curva que recorre todos los vértices la cual es codificada usando el *5ODCCC*.

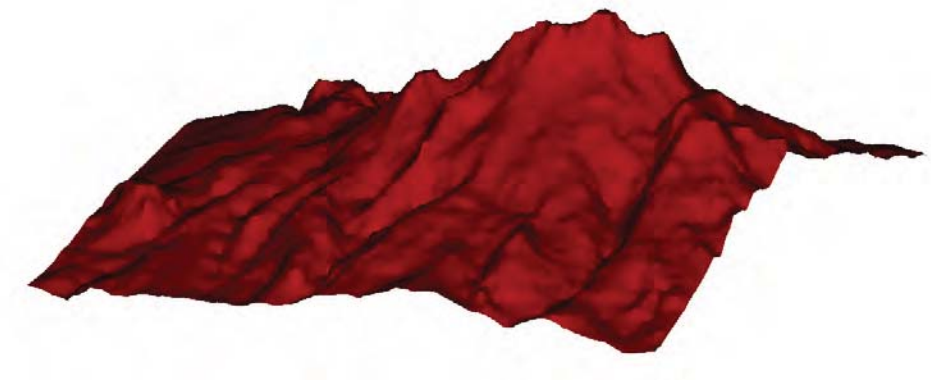
5.1.2. Generación de la gráfica de adyacencia

A las superficies voxelizadas de rostros humanos, así como la superficie voxelizada del volcán *"Iztaccíhuatl"* se les toma como superficies simples conexas, por lo tanto, la gráfica de adyacencia es una gráfica conexa.

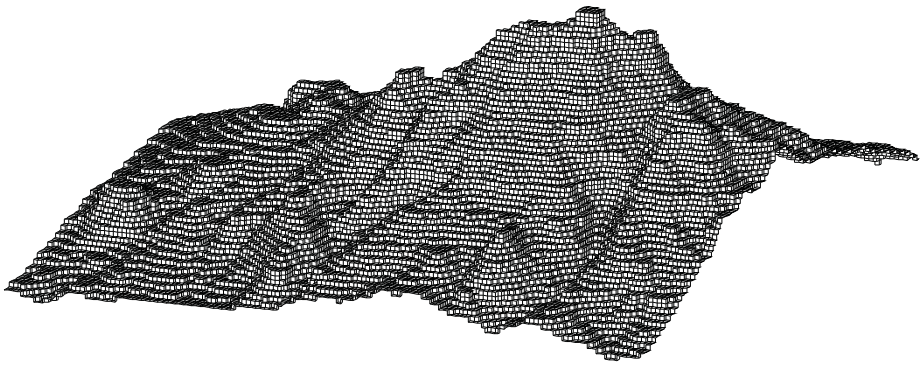
Como la gráfica es conexa, se puede reconstruir la superficie voxelizada a partir de las adyacencias locales de ésta. Aunque la superficie resultante no necesariamente es la misma (por lo visto en la Sección 2.4 después de la Def. 2.44). Para comparar los métodos de representación, únicamente es de interés la cantidad de memoria utilizada por la representación de la gráfica de adyacencia. Por lo visto en la Subsección. 4.1.4 únicamente se guardan las aristas de ésta como pares. Dicha información es la que se utiliza para las comparaciones.

El algoritmo para obtener las aristas de la gráfica de adyacencia, se basa en generar las adyacencias de la superficie voxelizada de forma ordenada, para garantizar que éstas no se repitan. En cada uno de los vóxeles pertenecientes a la superficie voxelizada, sólo se toman la mitad de las posibles adyacencias, las otras adyacencias estarán consideradas cuando se escaneen los vóxeles correspondientes. Se generan así, los pares de las aristas, de esta forma la unicidad de cada arista está asegurada.

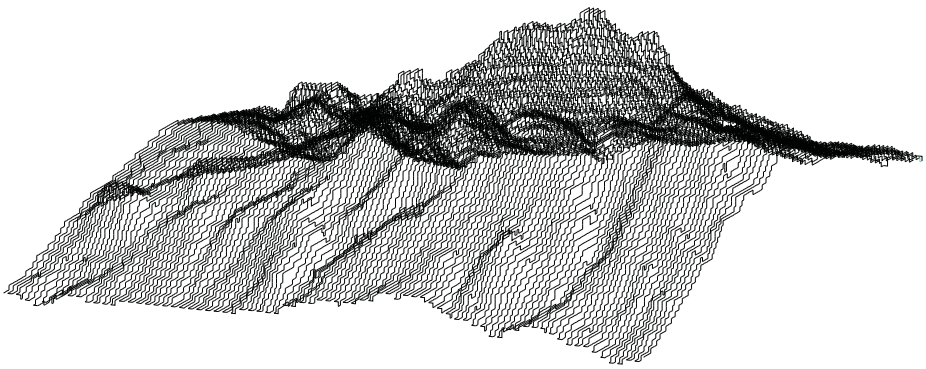
Por la simpleza del algoritmo, éste se hace utilizando programación en paralelo, se decidió utilizar la plataforma y modelo de programación *CUDA* (Compute Unified Device Architecture) desarrollado por la compañía NVIDIA, debido a su fácil integración con C/C++ (ver [14]).



(a) Superficie.

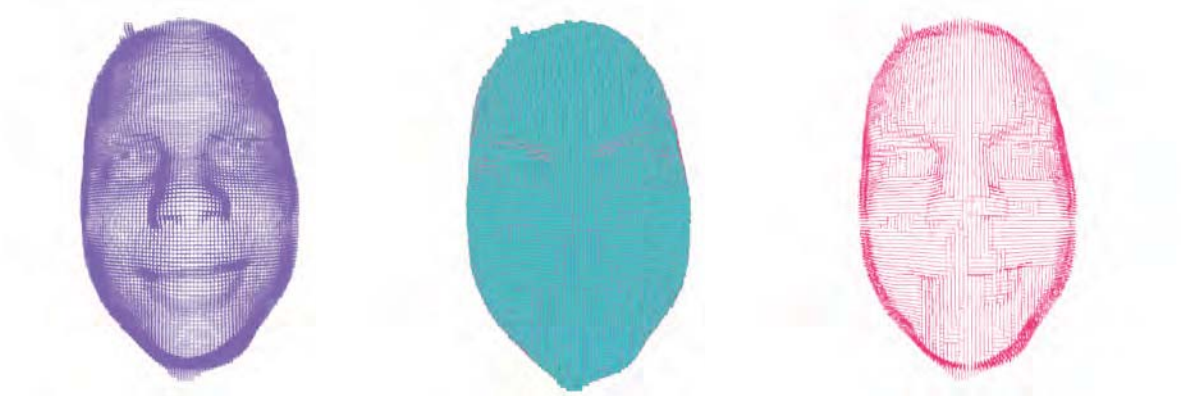


(b) Vóxeles.



(c) Curva.

Fig. 5.1: Las distintas representaciones del Iztaccíhuatl.



(a) Vóxeles. (b) Árbol envolvente sobre vóxeles. (c) Árbol generado a partir del *5ODCCC*.

Fig. 5.2: Rostro y sus distintas representaciones.

5.1.3. Generación del árbol envolvente en superficies voxelizadas

Para generar los datos para comparar la compresión del *5ODCC* en el árbol envolvente, se genera el árbol de la forma descrita en la Sección 3.5, tomando las modificaciones para que el árbol represente a una superficie.

Durante la generación del árbol envolvente cada dos ciclos de la construcción se va almacenando la estructura del árbol, así como el código de cadena generado a partir del árbol hasta ese momento. A partir de estas estructuras se genera una lista de los vóxeles visitados hasta ese momento, utilizando esta lista y la imagen original se genera una nueva lista de los vóxeles que están siendo representados por el árbol envolvente.

El árbol envolvente se genera en las superficies voxelizadas de rostros humanos, así como en la superficie voxelizada del *DEM* (ver Fig. 5.1). Como muestra, en la Fig. 5.2 se muestra la superficie voxelizada de un rostro humano (5.2a), a partir de ésta se generan el árbol envolvente (Fig. 5.2b) y el árbol generado a partir del *5ODCCC* (Fig.5.2c) del árbol envolvente.

Al generar el árbol se puede obtener el *5ODCCC* que lo representa, para comparar la forma de almacenar una superficie utilizando el árbol envolvente se utilizará únicamente la cadena de elementos del *5ODCCC*. Las superficies tomadas se encuentran en las figuras 5.8 y 5.9 con el árbol envolvente generado a partir del *5ODCCC*.

5.1.4. Generación del árbol de bordes en superficies digitales

Para generar los datos de las superficies voxelizadas usando el árbol de bordes en las superficies digitales se obtienen los árboles usando la descripción hecha en la Subsección. 3.6.1.

Se genera la superficie digital a partir de una 2-celda inicial, la superficie está formada únicamente por las 2-celdas que se clasificaran (ver Fig. 5.3) para obtener el árbol de bordes.

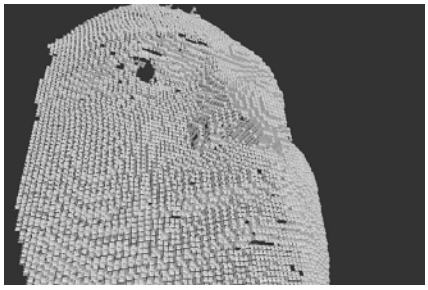


Fig. 5.3: Superficie digital generada a partir de una superficie voxelizada.

Usando la superficie voxelizada se usa la biblioteca *DGTAL* la cual tiene implementado un algoritmo que permite encontrar los planos digitales de una figura de vóxeles, usando el algoritmo *COBA* (ver [11]) y el algoritmo denominado *CHORD* generado a partir de [19]. Se modifica el algoritmo original, debido a que para la generación de los planos digitales de las imágenes de las superficies voxelizadas, resultaron demasiado grandes para el procesamiento con el algoritmo original. Aunque el reconocimiento de los planos sigue utilizando los algoritmos antes mencionados la forma para encontrar dichos planos fue modificada.

Para encontrar los planos digitales de la superficie voxelizada se toma un vóxel inicial, a partir de éste y sus vecinos se generan los posibles planos digitales a los cuales el vóxel inicial debe de pertenecer. La asignación se hace usando el tamaño de los posibles planos digitales dentro de la superficie voxelizada. Se toma al plano de mayor tamaño, se descartan los otros posibles planos y se siguen con los demás elementos de la superficie voxelizada que no se encuentren dentro de un plano ya construido. El proceso continúa hasta que todos los vóxeles de la superficie voxelizada se encuentren dentro de algún plano digital.

Habiendo asignado a cada vóxel de la superficie voxelizada a un plano digital, se genera una clasificación de los vóxeles dentro de la superficie. A partir de dicha clasificación se clasifica a los elementos de la superficie digital generada a partir de las 2-celdas. A cada una de las 2-celdas se les asigna el plano digital al cual pertenecen. La clasificación hecha en los vóxeles a partir de los planos digitales induce una clasificación en las 2-celdas de la superficie digital y ésta a su vez induce una clasificación en las aristas de la superficie digital.

Para todas las aristas que pertenecen a las 2-celdas de la superficie digital se les asigna el plano al cual pertenecen. Las aristas son divididas en dos conjuntos, uno en el cual las aristas sólo tiene asignado un plano, y otro en donde las aristas que pertenecen a más de un plano digital. Las aristas que sólo pertenecen a un plano digital son las aristas internas de los planos digitales, por lo tanto, las aristas que se encuentran en más de un plano digital son las aristas que se encuentran en los bordes de la superficie voxelizada. Dichas aristas son las que pertenecen al árbol de bordes.

El vértice raíz del árbol de bordes se toma de la misma forma que el vértice raíz del árbol envolvente. Al no poder garantizar que dicho elemento forme parte de un borde, se genera un algoritmo, el cual encuentra un árbol cuyas aristas son aristas internas y es el árbol de menor tamaño que conecta al vértice raíz a un elemento que se encuentre dentro de los bordes. Al encontrar dicho elemento se eliminan las ramas del árbol que no contienen elementos de los bordes.

El algoritmo para encontrar el árbol de bordes va recorriendo las aristas y los vértices que se encuentran en más de un plano digital. De la misma forma que el árbol envolvente, el algoritmo del árbol de bordes se termina cuando no se encuentran más aristas y vértices que formen parte de los bordes de la superficie digital.

El árbol resultante es lo que lo que se le llama el árbol de bordes. Los árboles de bordes analizados en el trabajo son obtenidos a partir de las superficies voxelizadas de rostros humanos.

Para hacer una comparación en contra del árbol envolvente, también se genera el árbol envolvente de las superficies digitales.

5.2. Análisis de los datos

El tomar la curva de barrido de una superficie y codificarla usando el *5ODCCC* es una manera muy eficiente de compresión de información. Generar una curva que representa a una superficie tiene desventajas como las analizadas después de la Def. 2.45, en la Sección 3.3 y en la Sección 4.3. Entre las cuales se destaca la gran sensibilidad al ruido, y que las adyacencias se pierden por completo hasta la reconstrucción total del objeto.

A partir del *DEM* del "*Iztaccíhuatl*" se obtiene una superficie voxelizada y es la única superficie de la cual se obtiene la curva de barrido con el *5ODCCC*, el árbol envolvente codificado usando el *5ODCCC* y las gráficas de 6, 18 y 26 adyacencia. Como es de esperarse la compresión obtenida de la curva de barrido es extremadamente eficiente, pero la sensibilidad al ruido y la pérdida casi total de la geometría local son las razones por las cuales esta forma de compresión no es de nuestro interés; sin embargo se considera para tenerla como referencia y ver que su compresión es difícil de alcanzar.

De las estructuras obtenidas y codificando dichas estructuras como se detalla en la Sección 5.1 se obtienen los resultados en la Tabla 5.1. La representación de las coordenadas de los centros de los vóxeles se denota como "*Vóxel*", a la representación del árbol envolvente usando el *5ODCCC* se representa como "*5ODCCC* AE", a la representación del árbol envolvente usando el *5ODCCC* y comprimido con Huffman lo denotamos como "*5ODCCC* + Huffman".

Para las gráficas de adyacencia se denotarán como "n-Adj" donde $n \in \{6, 18, 26\}$ y para la curva de barrido se denotan como "Barrido".

Tabla. 5.1: Número de bits usados para una superficie en todas las representaciones.

Representación	# Numero de bits
Vóxel	539176
5ODCCC AE	477828
5ODCCC + Huffman	336644
6-Adj	1487010
18-Adj	3191130
26-Adj	3597180
Barrido	75861

En la Tabla 5.1 se encuentra el número de bits utilizado para las distintas representaciones de la superficie voxelizada del *DEM* del "*Iztaccíhuatl*". Usando los datos presentados en dicha tabla se puede observar, que para este caso, la hipótesis de que el número de bits utilizados para almacenar una superficie voxelizada usando el *5ODCCC* en el árbol envolvente es menor a la utilizada por la representación de los centros de los vóxeles.

En la tabla se muestra también que la cantidad de bits necesarios para almacenar la gráfica de adyacencia es mayor que la representación en vóxeles.

Por como se genera el árbol envolvente, es posible determinar los elementos visitados por el árbol en cada paso, de esta forma se puede generar listas de los vóxeles visitados en cada paso de la construcción del árbol envolvente, a partir de éstas fue posible determinar el número de vóxeles que van representando en cada uno de los pasos para obtener el árbol envolvente (ver la Fig. 5.4). Debido a esto se puede generar las gráficas mostradas en las figuras 5.5 y 5.6 en donde se muestra el número de bits utilizado en distintas representaciones. La curva de barrido no es generada para estas superficies voxelizadas, pues en el caso de los rostros no se puede garantizar la existencia de ésta.

En la Fig. 5.5 se observa como el número de bits utilizados para almacenar una superficie utilizando el *5ODCCC* en el árbol envolvente es menor que el número de bits para almacenar la superficie utilizando las coordenadas del centro de los vóxeles. En la Fig. 5.6 se pueden observar las distintas representaciones.

Por lo mostrado en la gráfica de la Fig. 5.6 y la Tabla 5.1 se pueden inferir pequeñas conclusiones a partir de estos datos. Para comparar la cantidad de datos usada por cada representación es mejor usar las coordenadas de los centros vóxeles, que la gráfica de adyacencia. Ya que ésta utiliza demasiados bits para ser almacenada, por lo tanto, no se utilizará la gráfica de adyacencia para otras comparaciones.

Otra afirmación, como se observa con anterioridad, es que la curva de barrido es una forma de representación que presenta una gran compresión en la cantidad de datos utilizados. Pero por las razones explicadas con anterioridad se considera que no es una representación deseable. Por lo cual se evita obtener dicha representación y por lo tanto, sus comparaciones.

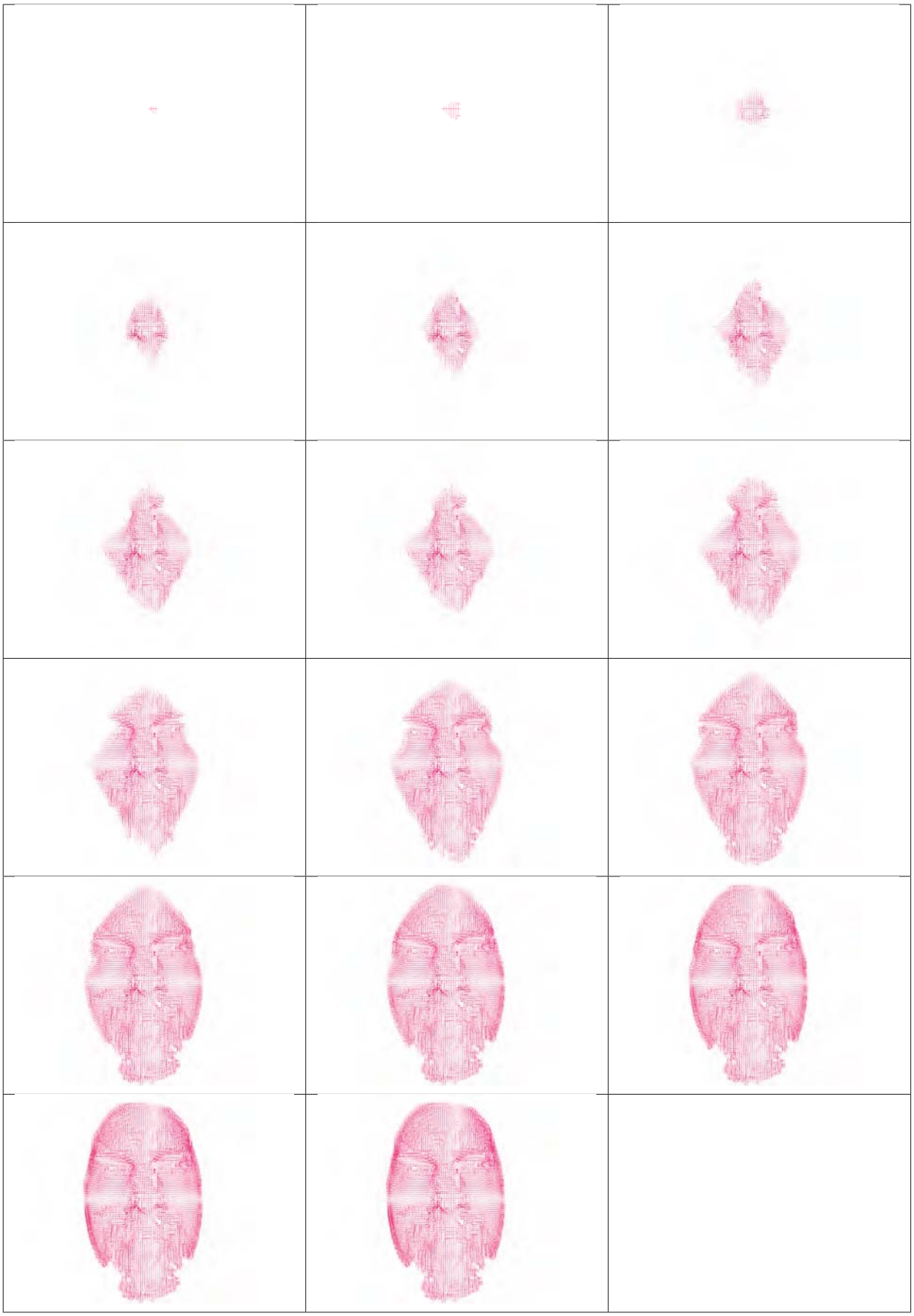


Fig. 5.4: El árbol envolvente en distintas etapas.

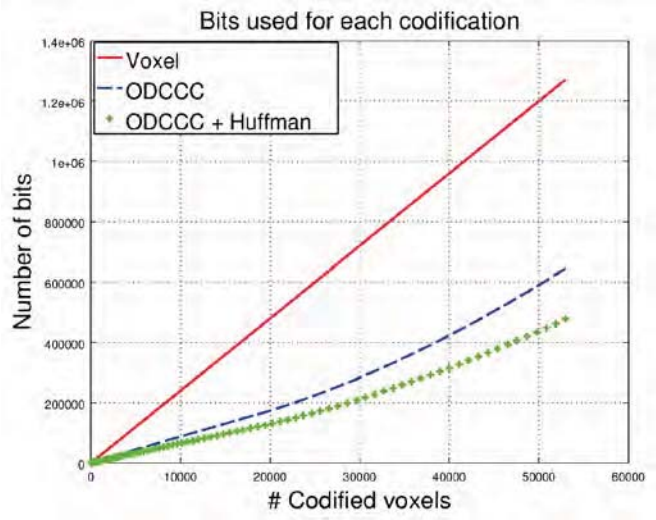


Fig. 5.5: Comparación entre la representación de vóxeles, la representación del árbol envolvente y la codificación de Huffman de éste en distintos pasos.

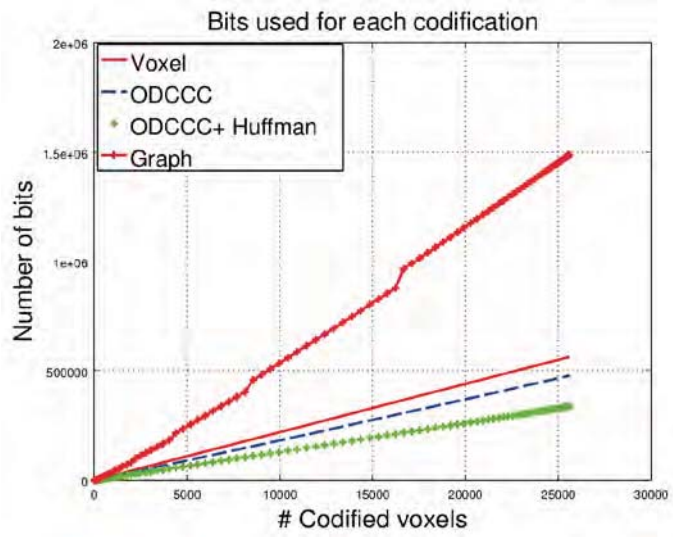
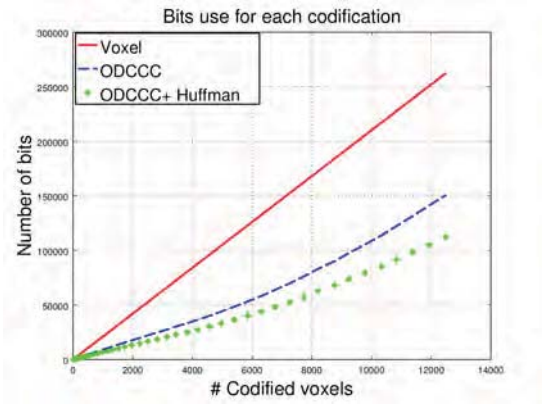
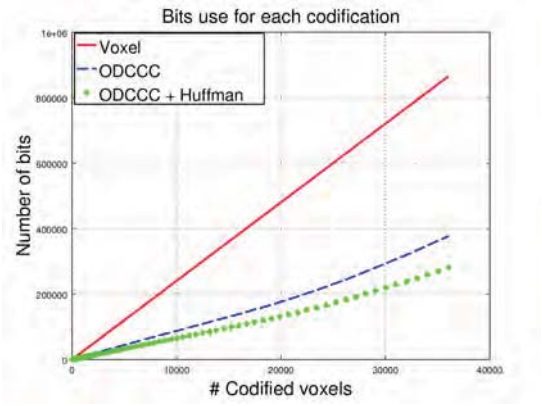


Fig. 5.6: Número de bits usados para distintas representaciones.

Como la cantidad de datos usados por la representación de coordenadas de los centros de vóxeles depende de la resolución del objeto, podría pensarse que disminuyendo la resolución el porcentaje de compresión disminuiría comparado contra el número de bits usados para guardar el árbol envolvente de la superficie con el *5ODCCC*. Sin embargo como muestran las gráficas en las figuras 5.7a y 5.7b la mejora no es significativa. En la Fig. 5.7a se muestra el número de bits usados para representar a una superficie voxelizada en una resolución $128 \times 128 \times 128$ mientras que en la Fig. 5.7b se muestra el número de bits de la misma superficie en una resolución $256 \times 256 \times 256$. Como muestran las gráficas en ambos casos la representación de la superficie utiliza una menor cantidad de bits usando el árbol envolvente. En la Tabla 5.2 se muestra el número de bits usados en cada una de las representaciones en las distintas resoluciones.



(a) Resolución $128 \times 128 \times 128$.



(b) Resolución $256 \times 256 \times 256$.

Fig. 5.7: Gráficas de la cantidad de datos para la misma superficie en diferentes resoluciones

Resolución	bits *Vóxeles*	bits *5ODCCC*	bits *5ODCCC* + Huffman
$128 \times 128 \times 128$	$262,353$	$150,705$	$112,357$
$256 \times 256 \times 256$	$865,224$	$377,379$	$280,972$

Tabla. 5.2: Número de bits usados para diferentes representaciones para la misma superficie en distintas resoluciones.

Como se observa la cantidad de bits para almacenar la superficie en la resolución $128 \times 128 \times 128$ usando el *5ODCCC* para representar el árbol envolvente presenta un $57.44\,\%$ de los datos usados en la representación de las coordenadas de los centros de los vóxeles. Mientras que en la resolución $256 \times 256 \times 256$ el árbol envolvente utiliza $43.61\,\%$ de los datos utilizados en la representación de las coordenadas de los centros de los vóxeles. Aunque la compresión es menor utilizando una resolución menor, la compresión sigue existiendo. Si se aumenta la resolución el porcentaje de compresión aumentará también. Se observa que el porcentaje de compresión depende de la resolución en la que se trabaja, pero la diferencia resulta poco significativa comprado con la perdida de resolución del objeto.

Superficie	# Vóxeles	*5ODCCC*/Vóxel	Huffman/Vóxel	Huffman/*5ODCCC*
surf5-128	13,236	57.44 %	43.19 %	75.19 %
surf6-128	12,493	57.44 %	42.82 %	74.55 %
surf7-128	11,900	51.36 %	38.68 %	75.31 %
sur9-128	104	106.31 %	84.06 %	79.06 %
surf14-128	14,798	51.13 %	37.95 %	74.22 %
surf16-128	13,320	83.29 %	62.21 %	74.69 %
surf5	52,975	50.80 %	37.60 %	74.02 %
surf6	36,051	43.61 %	32.47 %	74.45 %
surf7	40,316	40.73 %	30.21 %	74.18 %
surf11	31,070	39.04 %	28.94 %	74.13 %
surf14	42,292	41.48 %	30.70 %	74.00 %
surf16-2	18,385	68.12 %	51.05 %	74.93 %
surf17-2	32,298	36.51 %	27.11 %	74.27 %
surf18-2	27,446	49.89 %	36.97 %	74.11 %
surf19-2	36,145	36.98 %	27.38 %	74.05 %
surf23-2	45,616	38.06 %	28.16 %	73.98 %
surf24-2	29,320	69.15 %	51.51 %	74.49 %
surf25-2	42,725	37.79 %	27.89 %	73.81 %

Tabla. 5.3: Superficies y los respectivos porcentajes de compresión.

La Tabla 5.3 presenta varias superficies con su número de vóxeles, el porcentaje de bits que se utilizan para almacenar la superficie con el árbol envolvente usando el *5ODCCC* con respecto al número de bits necesario para almacenar la superficie con las coordenadas de los centros de los vóxeles, el porcentaje de bits utilizados de la compresión de Huffman aplicada al *5ODCCC* del árbol envolvente con respecto a la representación de las coordenadas de los centros de los vóxeles, el porcentaje de datos utilizados por la compresión de Huffman aplicada al *5ODCCC* del árbol envolvente con respecto a la representación del *5ODCCC* del árbol envolvente de la superficie.

En la Tabla 5.3 se puede observar que la compresión aparece en la mayoría de las superficies. Sin embargo, la superficie *surf9-128* es una superficie con un menor número de vóxeles que el resto de las superficies. En ésta no existe una compresión de los datos para almacenar la superficie usando el *5ODCCC* en el árbol envolvente. Esto indica que aunque el árbol envolvente depende de la geometría local de la superficie, para que exista compresión en la cantidad de datos usados para almacenar la superficie usando el *5ODCCC* aplicado al árbol envolvente, es necesario que la superficie contenga un mayor número de vóxeles.

Si la superficie no contiene un número suficiente de vóxeles no existe la compresión, e incluso el número de bits utilizados para almacenar la superficie usando el *5ODCCC* en el árbol envolvente de la superficie voxelizada es mayor.

En las Fig. 5.8 y Fig. 5.9 se encuentran las superficies y sus árboles envolventes, a partir de éstos se extrae la información para generar la Tabla 5.3.

Como se observa la compresión varía con respecto al número de vóxeles y con respecto a la resolución de las superficies. Si se toma únicamente a las superficies con resolución $128 \times 128 \times 128$ se tiene que en promedio la compresión de la superficie es alrededor de 67.82 % tomando el *5ODCCC* del árbol envolvente y si se le aplica la compresión de Huffman al *5ODCCC* del árbol envolvente se tiene que se alcanza una compresión promedio de 51.485 %.

En el caso de las superficies con resolución $256 \times 256 \times 256$ como es de esperarse, la compresión resulta ser mayor. En estos casos se consigue una compresión promedio de 41.18 %, al representar la superficie con el *5ODCCC* del árbol envolvente. Si se aplica la compresión de Huffman a estas cadenas se obtiene en promedio una compresión de alrededor 34.2 % con respecto al almacenamiento de las coordenadas de los centros de los vóxeles.

Si se consideran los dos casos se obtiene una compresión promedio con respecto a la representación de las coordenadas de los centros de los vóxeles de 52.72 %, al tomar la representación del árbol envolvente usando el *5ODCCC*. Y al aplicar la compresión de Huffman se obtiene una compresión de 39.95 %.

Al analizar la Tabla 5.3 se observa una compresión adicional al aplicar la compresión de Huffman al *5ODCCC* del árbol envolvente, al comparar con la representación de las coordenadas de los centros de los vóxeles ésta no es una compresión constante. Pero la tabla muestra que aunque no existe una compresión en la cantidad de datos usados para almacenar el *5ODCCC* del árbol envolvente de la superficie *surf9-128*, aun así se puede comprimir la cantidad de información usando la compresión de Huffman sobre la cadena de *5ODCCC*. Esto afirma que la aportación del trabajo, no sólo se basa en el hecho de que la representación de la superficie con el *5ODCCC* aplicado al árbol envolvente sea una compresión en la mayoría de los casos. La importancia recae en poder representar a la superficie como una cadena de símbolos, debido a esto se puede comprimir aun más utilizando técnicas de compresión ya conocidas.

La geometría local al ser determinante para la obtención del árbol envolvente, se encuentra codificada dentro de la estructura de árbol. En comparación con otros métodos como la curva de barrido o la gráfica de adyacencia en donde se pierde gran parte de esta información y no se recupera hasta recobrar al objeto en su totalidad.

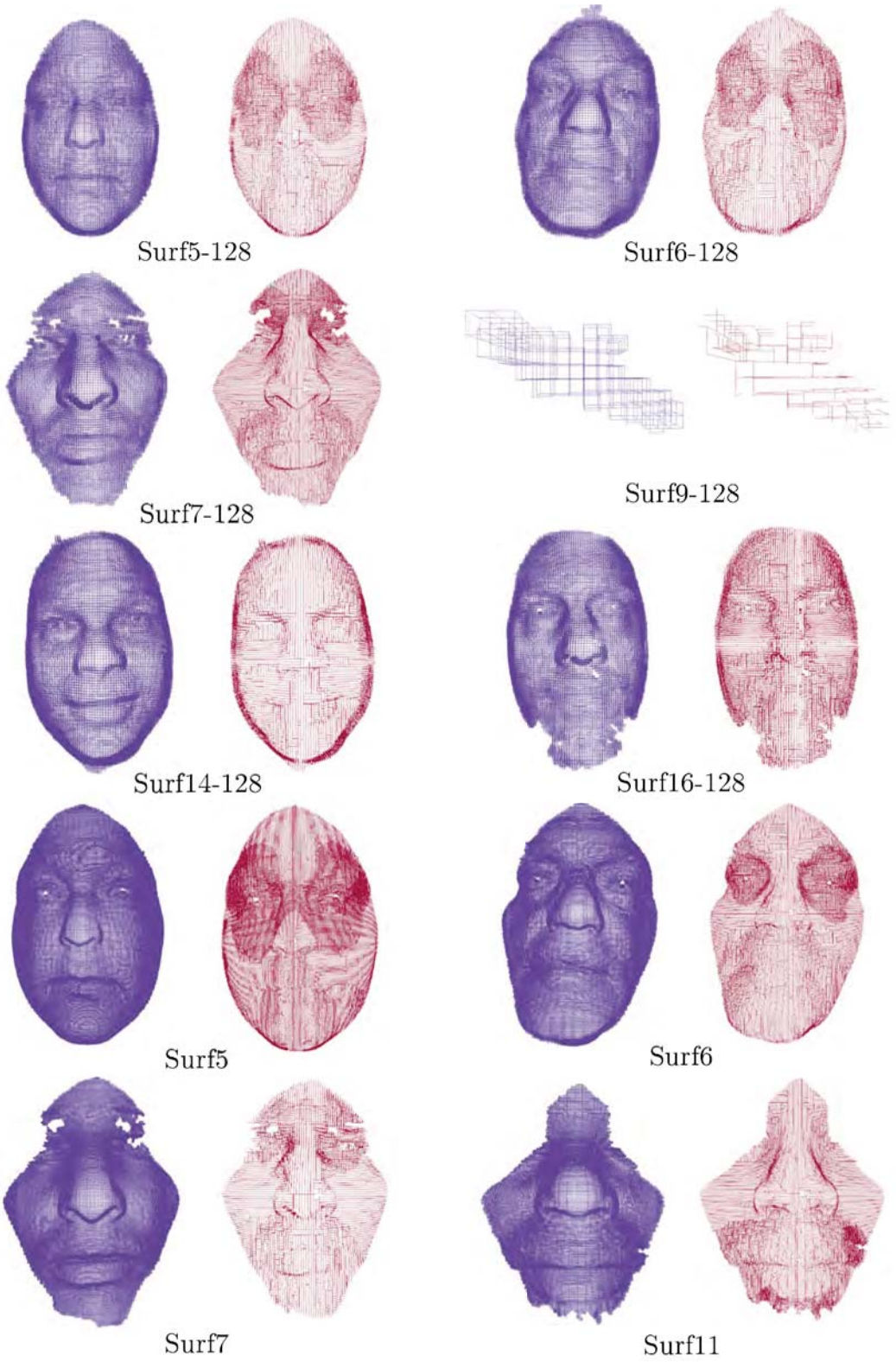


Fig. 5.8: Superficies estudiadas 1.

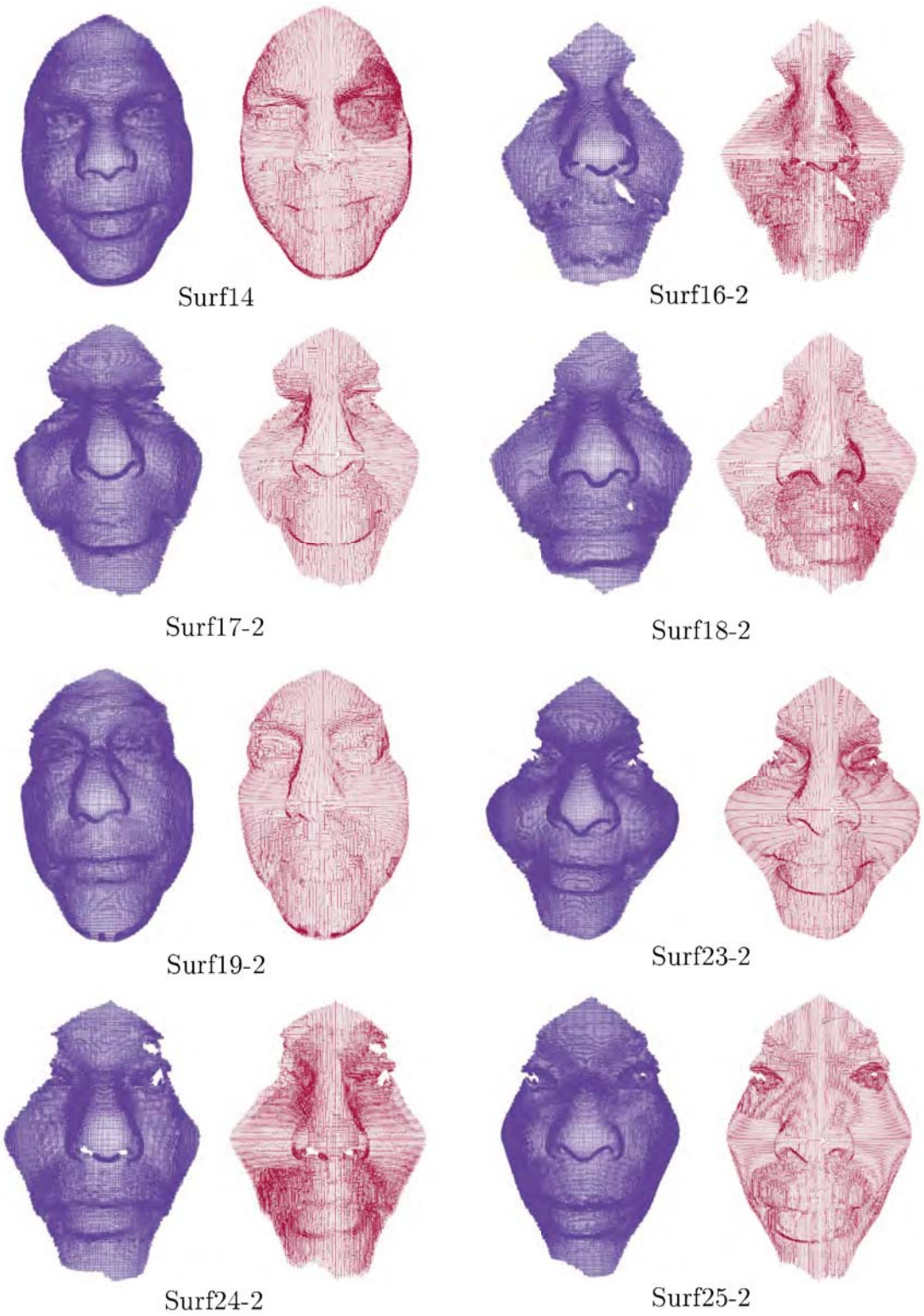


Fig. 5.9: Superficies estudiadas 2.

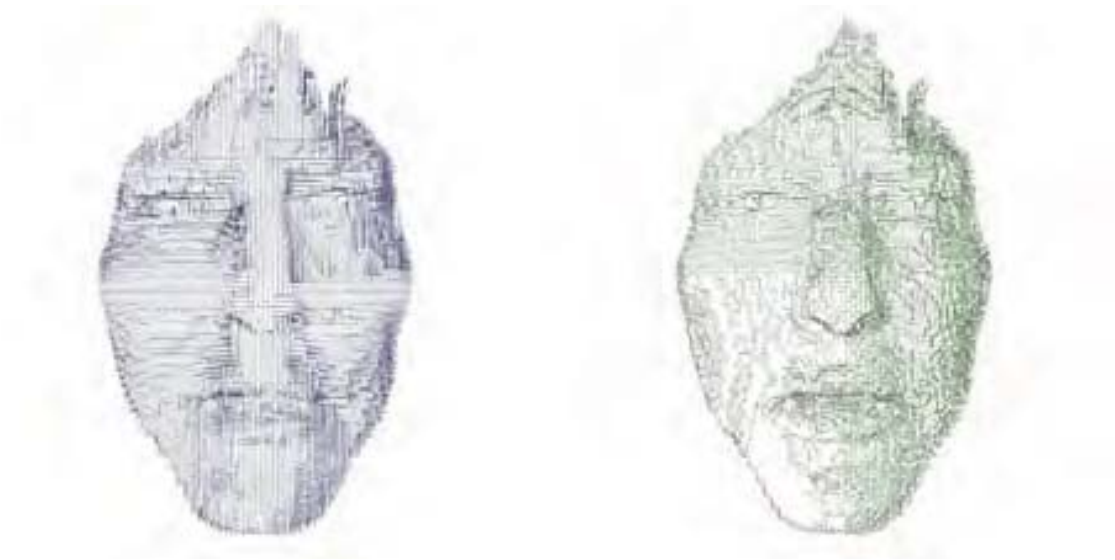
Superficie	Entropía *5ODCCC*	Entropía Huffman
surf5-128	2.13367532356	0.96804743659
surf6-128	2.11662140828	0.970857847153
surf7-128	2.14219825064	0.969098654846
sur9-128	2.26043609152	0.97114143771
surf14-128	2.09963265626	0.970379427369
surf16-128	2.11938511064	0.969622599996
surf5	2.08242779988	0.96841556384
surf6	2.10306855318	0.968580129988
surf7	2.09057186378	0.968223944648
surf11	2.08380389296	0.967408964636
surf14	2.08187943815	0.96838339623
surf16-2	2.11989472719	0.967260574988
surf17-2	2.08022356210	0.964658807347
surf18-2	2.08432071590	0.967239783062
surf19-2	2.08483792847	0.968221410535
surf23-2	2.06866257589	0.965561132957
surf24-2	2.09689647678	0.966410231596
surf25-2	2.07185526019	0.968478468899

Tabla. 5.4: Entropía.

Compresión en el Árbol de Bordes

La representación del árbol envolvente usando el *5ODCCC* de una superficie voxelizada presenta una compresión con respecto al número de bits necesarios para almacenar la superficie usando las coordenadas de los centros de los vóxeles. Aunque la compresión es buena, como lo reportan los datos presentados en la Sección 5.2 el número de elementos utilizados para almacenar las características geométricas de la superficie son demasiado grandes. En [25] se presenta un árbol similar al árbol envolvente llamado *árbol de bordes* el cual representa a una figura de vóxeles utilizando los bordes de la figura.

Como se explica en la Subsección. 3.6.1 y por como se genera el árbol de bordes (ver la Subsección. 5.1.4) los elementos que no se toman en cuenta, son los vértices que pertenecen al interior de los planos digitales. Por lo tanto, a diferencia de la representación del árbol envolvente, la estructura del árbol de bordes, es una representación con perdida. De forma similar a la gráfica de adyacencia, pues se puede tener el mismo árbol de bordes para dos superficies distintas.



(a) Árbol envolvente. (b) Árbol de bordes.

Fig. 5.10: Los árboles; envolvente y de bordes.

Los vértices internos de los planos digitales no pueden ser reconstruidos utilizando al árbol de bordes, esto es debido al grosor de los planos digitales. Al tener únicamente los bordes de los planos digitales, se pueden determinar los planos digitales que pasan por dichos bordes, pero no es posible determinar si todos los elementos del plano, entre los bordes, pertenecen a la imagen original.

La característica principal del árbol de bordes, al igual que el árbol envolvente, es que a partir de esta representación se puede generar una representación de una estructura $3D$ en una cadena de símbolos. La diferencia con respecto al árbol envolvente es que no se toman en cuenta elementos (vértices) que no aportan a la complejidad de la superficie.

Para la generación del árbol envolvente a partir de las superficies voxelizadas, se extrae la superficie digital generada a partir de un surfel inicial de la superficie voxelizada (ver Fig. 5.3), a partir de ésta se construye el árbol envolvente y el árbol de bordes (ver Fig. 5.10). Por los resultados presentados en la Sección 5.2 es claro que usando el $5ODCCC$ en el árbol de bordes presentará una mayor compresión con respecto al número de bits necesario para almacenar las coordenadas de los centros de los vóxeles. Lo que es de interés, es saber la cantidad de elementos que no son representativos para la figura, pues estos serán eliminados y no estarán dentro del árbol de bordes.

Superficie	Número de Vóxeles	Bordes/Envolvente
Surf11	10,026	76.13 %
Surf16	13,320	79.75 %
Surf16b	15,980	79.24 %
Surf17	16,299	71.66 %
Surf17b	22,407	78.34 %
Surf19	16,221	76.74 %
Surf19b	11,895	76.94 %
Surf23	9,681	71.32 %
Surf23b	12,575	74.52 %
Surf24	13,258	76.78 %
Surf27b	12,548	72.34 %
Surf28	11,097	76.87 %
Surf28b	10,037	72.76 %
Surf32	13,290	75.76 %
Surf32b	12,525	73.82 %
Surf34	11,833	70.19 %
Surf34b	20,258	81.08 %
Surf38	10,573	74.20 %
Surf40	15,638	76.41 %
Surf40b	10,989	79.52 %
Surf46	10,966	76.52 %
Surf46b	12,405	73.61 %
Surf47	12,659	74.08 %
Surf48	9,951	72.85 %

Tabla. 5.5: Bordes / Envolvente.

Como resultado se obtiene la Tabla 5.5 a partir de la superficies en las figuras 5.11, 5.12, 5.13, 5.14, 5.15 y 5.16. En donde se observa que el número de vóxeles de la superficie voxelizada y los porcentajes de compresión del número de bits necesarios para almacenar el árbol de bordes usando el *5ODCCC*, con respecto al número de bits necesarios para almacenar el árbol envolvente usando el *5ODCCC*. Como se puede observar en las superficies analizadas, se obtiene una compresión mínima de 18.92 %, esto implica que la representación del árbol de bordes utiliza el 81.08 % de los datos necesarios para almacenar el árbol envolvente de la superficie "Surf34b". De la misma manera se observa que la compresión máxima fue de 28.81 % en la superficie "Surf34". En promedio se alcanza un compresión de 24.52 %.

Al generar una cadena que representa al árbol envolvente se puede comprimir aun más la representación utilizando métodos de compresión comunes, de esta forma se utiliza la compresión de Huffman para ver si dicha compresión varía. Como resultado se obtiene las compresiones de los *5ODCCC* de los árboles de bordes y cuyos porcentajes de compresión con respecto al código original aparecen en la Tabla 5.6. Como se observa al tomar el código de Huffman, del árbol envolvente mantiene un porcentaje similar al visto en la Sección 5.2, pero al tomar el código de Huffman en el *5ODCCC* el porcentaje de compresión disminuye.

Superficie	Envolvente Huff	Bordes Huff
Surf11	73.33 %	81.32 %
Surf16	73.58 %	80.17 %
Surf16b	73.37 %	80.14 %
Surf17	73.17 %	80.91 %
Surf17b	73.04 %	80.04 %
Surf19	73.30 %	80.22 %
Surf19b	73.78 %	80.18 %
Surf23	72.75 %	81.28 %
Surf23b	72.69 %	81.27 %
Surf24	73.55 %	81.05 %
Surf27b	73.26 %	80.86 %
Surf28	73.24 %	80.15 %
Surf28b	73.35 %	80.81 %
Surf32	73.47 %	80.91 %
Surf32b	73.56 %	80.79 %
Surf34	73.08 %	80.33 %
Surf34b	72.48 %	80.09 %
Surf38	73.23 %	81.28 %
Surf40	73.85 %	80.91 %
Surf40b	73.67 %	80.14 %
Surf46	73.32 %	81.06 %
Surf46b	73.33 %	81.08 %
Surf47	73.46 %	80.79 %
Surf48	73.54 %	81.26 %

Tabla. 5.6: Porcentajes de uso de datos de los correspondientes códigos de Huffman.

En promedio la compresión usando el código de Huffman en el *5ODCCC* del árbol envolvente presenta una reducción de alrededor de 26.69 % mientras que al utilizar el código de Huffman en el *5ODCCC* del árbol de bordes presenta una compresión promedio de 19.29 %. Aquí se observa una diferencia significativa de la compresión de Huffman, como se utiliza el mismo código la diferencia observada no se debe a la utilización del *5ODCCC*. Por lo tanto, la diferencia cae en la frecuencia de los símbolos del alfabeto del código. Es decir, la frecuencia de los símbolos es distinta y significativa al utilizar el árbol envolvente o el árbol de bordes.

Al calcular la entropía de los códigos (ver la Tabla 5.7) se observa que la entropía del *5ODCCC* del árbol envolvente es menor, con un promedio de 2.08 que la entropía del *5ODCCC* del árbol de bordes, con promedio de 2.30. Esto explica el hecho de la diferencia en la compresión de Huffman entre el código del árbol envolvente y el código del árbol de bordes. Lo que afirma que los elementos eliminados son los más regulares de la superficie y por lo tanto, el *5ODCCC* obtenido contiene a los elementos que aportan la mayor complejidad local a la superficie, en este caso los bordes de la superficie.

Como se observa en la Tabla 5.7, la entropía de los códigos de Huffman tanto del árbol envolvente como del árbol de bordes no presentan diferencias significativas.

Los resultados muestran que se puede obtener una estructura que representa a la superficie voxelizada de forma robusta, al igual que al utilizar el árbol envolvente. Ya que al eliminar los elementos que no son significativos dentro de la superficie se puede obtener una representación de la superficie pero con perdida de información. Esta forma mantiene los datos más significativos de la superficie y sirve como representación de la superficie voxelizada como se muestran en las figuras 5.11, 5.12, 5.13, 5.14, 5.15 y 5.16.

Superficie	Envolvente	Bordes	Envolvente Huffman	Bordes Huffman
Surf11	2.0913313476	2.3181844847	0.9685234342	0.9736336149
Surf16	2.0966254703	2.3026375346	0.9695211135	0.9727668191
Surf16b	2.0824604537	2.299024968	0.9683188452	0.9720433243
Surf17	2.0726168652	2.3016637971	0.9671835563	0.9706822359
Surf17b	2.061429266	2.2888478807	0.9665563406	0.9698907847
Surf19	2.0761407465	2.3053096987	0.9679871239	0.9745780793
Surf19b	2.1016883031	2.3043058891	0.9706462104	0.9737842182
Surf23	2.0584102837	2.3130001262	0.9650590823	0.9710624728
Surf23b	2.0541623336	2.3133699322	0.9647050609	0.9715643676
Surf24	2.0943265259	2.3109022837	0.9692992137	0.9735869618
Surf27b	2.0799825609	2.3029551379	0.9680750906	0.9723981004
Surf28	2.0838667047	2.3017237771	0.9679410394	0.9739743316
Surf28b	2.0890321257	2.3017960993	0.9686973777	0.9729890371
Surf32	2.0943927979	2.3036554295	0.9692661299	0.9723456096
Surf32b	2.1011127728	2.2972032497	0.9702352181	0.9708135084
Surf34	2.069184065	2.3103792554	0.9669170747	0.9751719606
Surf34b	2.0327663366	2.2907493619	0.963308919	0.970366886
Surf38	2.0787144954	2.3158587037	0.9676691444	0.9737011052
Surf40	2.1102159796	2.3059841933	0.9712395358	0.9733673573
Surf40b	2.1034208377	2.297910316	0.9698804788	0.9721078397
Surf46	2.082140592	2.3059822514	0.9679076151	0.9707526751
Surf46b	2.0835700991	2.3115901154	0.9681838129	0.9739533921
Surf47	2.0864807165	2.3006631278	0.9691421379	0.9731054749
Surf48	2.0951946958	2.3179636489	0.9696183933	0.9751171651

Tabla. 5.7: Las entropías de las distintas cadenas de códigos.

Surf11.

Surf16.

Surf16b.

Surf17.

Fig. 5.11: Surf11, Surf16, Surf16b y Surf19 voxelizadas, sus árboles de bordes y sus árboles envolventes.

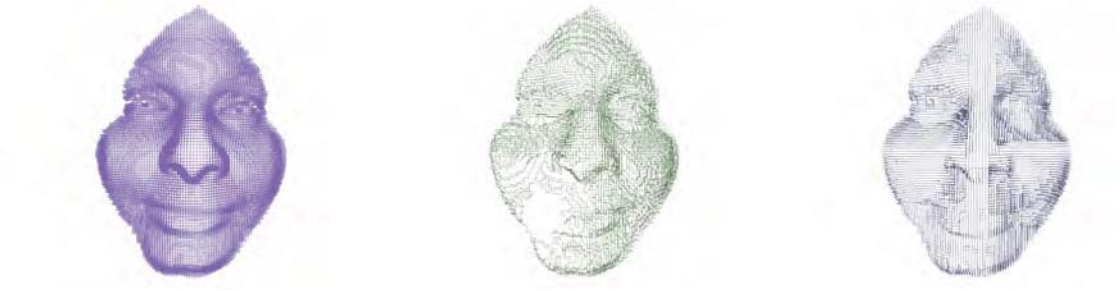
Surf17b.

Surf19.

Surf19b.

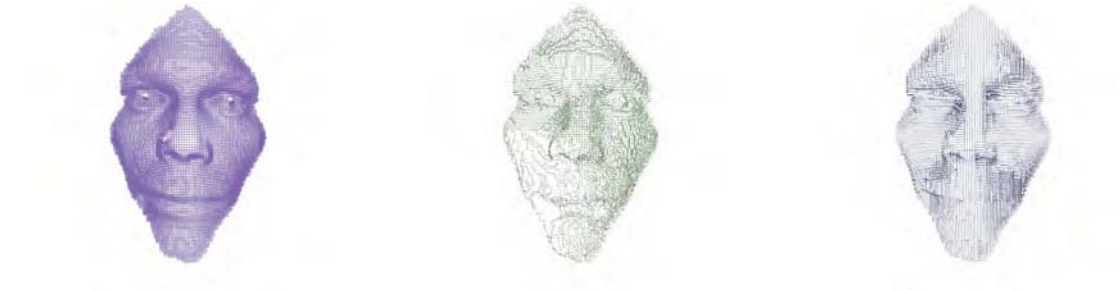
Surf23.

Fig. 5.12: Surf17b, Surf19, Surf19b, y Surf23 voxelizadas, sus árboles de bordes y sus árboles envolventes.

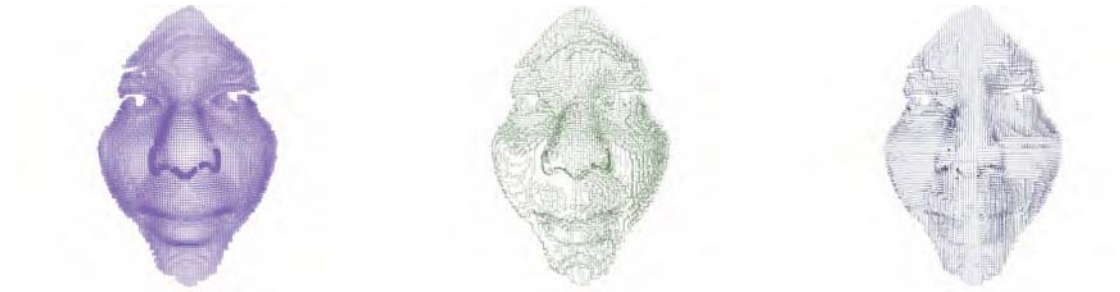


Surf23b.

Surf24.

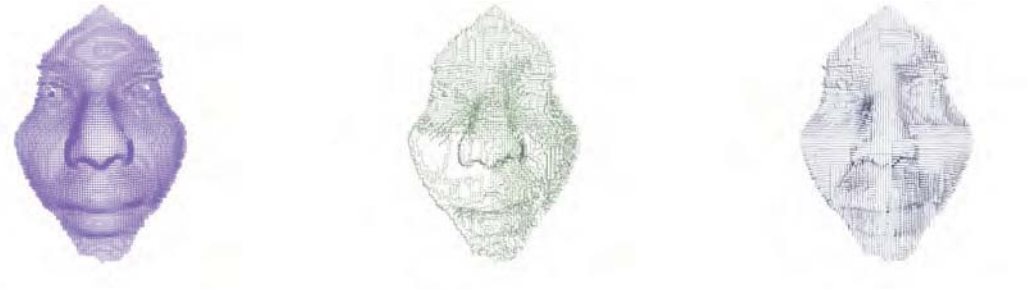


Surf27b.



Surf28.

Fig. 5.13: Surf23b, Surf24, Surf27, Surf28 voxelizadas, sus árboles de bordes y sus árboles envolventes.



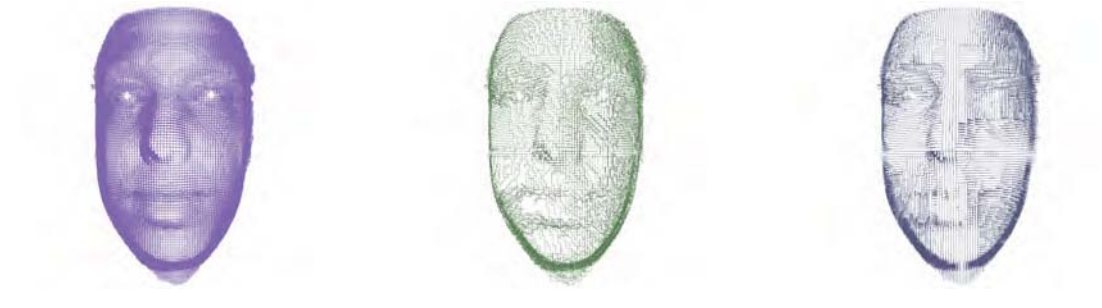
Surf28b.

Surf32.

Surf32b.

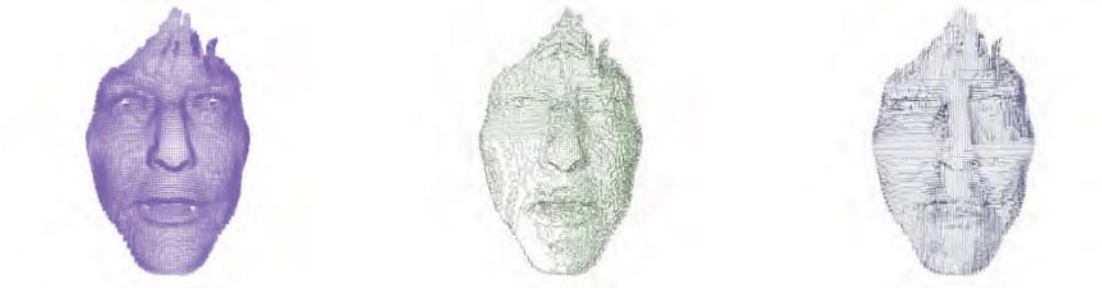
Surf34.

Fig. 5.14: Surf32, Surf34, Surf34b, y Surf38 voxelizadas, sus árboles de bordes y sus árboles envolventes.



Surf34b.

Surf38.



Surf40.

Surf40b.

Fig. 5.15: Surf34b, Surf38, Surf40, Surf40b voxelizadas, sus árboles de bordes y sus árboles envolventes.

Surf46.

Surf46b.

Surf47.

Surf48.

Fig. 5.16: Surf46, Surf46b, Surf47 y Surf48 voxelizadas, sus árboles de bordes y sus árboles envolventes.

Capítulo 6

Conclusiones

6.1. Sumario

A continuación se presenta un sumario de los capítulos anteriores del trabajo de investigación doctoral en el cual se hace un resumen de las ideas principales de éstos.

- El Capítulo 1 hace una breve introducción a la representación de objetos $3D$, y se presentan los distintos desafíos de representar a un objeto $3D$ y los usos más comunes para dichas representaciones, dentro de los cuales se destaca la detección y comparación de objetos. Se argumenta y se define el objetivo de la investigación, y se establece el objetivo principal de esta investigación, el cual es analizar las distintas representaciones de superficies voxelizadas, las cuales sean invariantes ante traslaciones y rotaciones, y comparar la cantidad datos necesarios para su almacenamiento.

- En el inicio del Capítulo 2 se presenta los conceptos de teoría de gráficas, para establecer las bases para las estructuras con las que se trabaja. El resto del capítulo, es una introducción a los conceptos de la *geometría digital* para hacer referencia cuando se detallen las construcciones de las distintas representaciones de las superficies voxelizadas dentro del espacio $\mathbb{Z}^3$.

- En el Capítulo 3 se presentan los códigos de cadena, haciendo una breve presentación de la historia de éstos. Se detalla el uso de la *curva de barrido* como representación de una superficie y se establece el uso del $5ODCCC$ para representar los árboles encajados en $3D$ y así representar un objeto $3D$ usando una cadena de símbolos.

Se detalla la construcción del *árbol envolvente* y las modificaciones hechas a la forma del árbol presentada en [9] para que la representación del árbol envolvente sea única a partir del vértice raíz de éste y represente a una superficie voxelizada, y no a un objeto voxelizado.

También se establece la forma de extraer una superficie digital de una superficie voxelizada, para la construcción del *árbol de bordes* a partir de la superficie digital y la construcción del mismo.

- El Capítulo 4 inicia con la presentación de los casos básicos de compresión de superficies $3D$ y describe la bibliografía actual sobre la compresión de superficies en $3D$. El tema principal del capítulo es presentar como se calcula la cantidad de bits necesaria para almacenar cada representación usada dentro del trabajo. Haciendo énfasis en las representación de una superficie voxelizada a través del *5ODCCC* del árbol envolvente y del árbol de bordes.

  Se dan las bases para a hacer las comparaciones en el Capítulo 5 y se argumenta la utilización de las representaciones usadas en el trabajo.

  Se detalla la compresión de Huffman para la compresión de cadenas, pues es una compresión que se consideró simple, esto con la finalidad de hacer énfasis en la posibilidad de obtener una mayor compresión usando el *5ODCCC* del *árbol envolvente* y el *árbol de bordes*.

- Durante el Capítulo 5 se detalla como se generan las superficies voxelizadas utilizadas en el trabajo. A partir de éstas se generan las representaciones a comparar, haciendo énfasis en las representaciones que utilizan al *5ODCCC* para representar a las superficies voxelizadas como son el *5ODCCC de la curva de barrido*, el *5ODCCC del árbol envolvente* y el *5ODCCC del árbol de bordes*.

  Se calculan las representaciones y la cantidad necesaria de bits para almacenar las representaciones para todas las representaciones dentro del trabajo las cuales son: centro de vóxeles, gráfica de adyacencia, el *5ODCCC* de la *curva de barrido*, del *árbol envolvente* y del *árbol de bordes*, respectivamente.

  Usando dichas representaciones se compara el número de bits necesarios para almacenar estas estructuras. Aquellas representaciones que son una cadena de símbolos y representan a un árbol se comprimen, usando la compresión de Huffman y adicionalmente calcula su entropía.

Se hace un análisis de los resultados obtenidos y se concluye que la representación del árbol envolvente y del árbol de bordes presentados como una cadena de símbolos del *5ODCCC* presentan una compresión.

6.2. Conclusiones

Al representar una superficie voxelizada usando el *5ODCCC* del árbol envolvente se obtiene una representación lineal de ésta. La representación originalmente es presentada en [9], se utiliza en su versión original para representar objetos voxelizados a través de su superficie externa, las cuales son superficies cerradas. El trabajo presentado modifica al árbol envolvente para representar superficies simples voxelizadas, generando una representación novedosa de un objeto bidimensional (una superficie voxelizada) encajado dentro de un espacio $3D$, a través de una cadena de símbolos, en este caso el *5ODCCC*.

Otra de las modificaciones hechas en el trabajo se hace con la intención de que la representación de las superficies voxelizadas usando el *5ODCCC* del árbol envolvente no dependa de otros factores más que el vértice raíz del árbol envolvente. Esto se consigue eliminando la manija para establecer los códigos iniciales de los vecinos del vértice raíz. Al eliminar la manija, el orden de los vecinos del vértice raíz del árbol envolvente depende únicamente del vértice inicial, lo que facilitaría la comparación entre los *5ODCCC* de los árboles envolventes.

La representación de árbol envolvente usando el *5ODCCC* es invariante ante rotaciones y traslaciones, además presenta una compresión en la cantidad de datos a almacenar (en la mayoría de los casos) en comparación con la representación de las coordenadas de los centros de los vóxeles.

El *5ODCCC* del árbol envolvente puede no presentar una compresión en la cantidad de datos utilizados para su almacenamiento (como se ve en la Sección 5.2 en la Tabla 5.3). Pero al ser una cadena de símbolos, ésta se puede comprimir de forma eficiente utilizando formas de compresión conocidas como la compresión de Huffman, a diferencia de una imagen de vóxeles o la gráfica de adyacencia de la superficie las cuales no son buenas para comprimir usando métodos similares a Huffman.

Aunque la compresión del *5ODCCC* del árbol envolvente no alcanza la compresión presentada por una curva de barrido con respecto a la representación de los centros de los vóxeles, la representación de la superficie voxelizada a través del *5ODCCC* del árbol envolvente es menos sensible al ruido y a diferencia de la curva de barrido, donde la existencia de ésta no esta garantizada, el árbol envolvente siempre puede ser construido en una superficie voxelizada simple. Se evita reconstruir la superficie para comparar con la superficie original, pues esta representación se sabe que es una compresión sin perdida por lo que su reconstrucción no debe ser distinta.

Como el árbol envolvente de una superficie voxelizada es una representación que no es fácil de usar para comparar, pues ésta contiene tantos vértices como la superficie digital de la superficie voxelizada que representa. Por tal motivo se presenta la representación de la superficies voxelizadas a través del *5ODCCC* del árbol de bordes. La representación del árbol de bordes se ve como una forma para representar a la superficie con un árbol cuyo número de vértices es menor al número de vértices del árbol envolvente. Esta representación es una representación con pérdida pues al utilizar los planos digitales y eliminar los elementos internos de éstos, se pierde la geometría local de los elementos en los planos internos, los cual genera que elementos geométricos como los hoyos dentro de planos internos se pierdan en la reconstrucción.

Se presenta una forma distinta para generar una superficie digital a las establecidas dentro de la literatura especializada (ver [13], [28] y [32]). La representación de una superficie voxelizada usando el *5ODCCC* del árbol de bordes, depende de la selección de planos dentro de la superficie digital. Por lo que el número de elementos del árbol de bordes depende de como se seleccionen los vóxeles que corresponden a los planos digitales. Otro elemento a considerar es el grosor de los planos digitales, pues éste puede disminuir o aumentar el número de planos que se considera para representar a la superficie voxelizada.

La representación del árbol de bordes depende de más parámetros que la representación del árbol envolvente. En las figuras 5.11, 5.12, 5.13, 5.14, 5.15 y 5.16 (en donde se muestran las superficies voxelizadas, el árbol de bordes y el árbol envolvente respectivamente) el árbol de bordes sigue representando a la superficie voxelizada, eliminando los elementos que se consideran planos dentro de la superficie digital.

Se sabe que al reconstruir la superficie voxelizada a partir del árbol de bordes no contendrá los mismos elementos, como se menciona en la Sección 5.2. No se consideró hacer una reconstrucción a partir de la representación del árbol de bordes para comparar con la superficie original, pues la mayoría de las superficies utilizadas contiene hoyos, como sabemos que la reconstrucción eliminaría a éstos, por lo que la comparación no sería justa.

El trabajo presentado muestra la posibilidad de utilizar representaciones de una superficie voxelizada a partir de una cadena de símbolos. Las características del árbol envolvente y del árbol de bordes para representar a las superficies voxelizadas permiten que la representación de la superficie sea un objeto lineal (la cadena de símbolos), en ambos caso se muestra que al representar las superficies, éstas muestran una compresión, mientras que la representación del árbol envolvente es una representación sin perdida, la del árbol de bordes muestra una representación con pérdida.

6.3. Comentarios finales

Las representaciones presentadas en este trabajo se pueden utilizar para almacenar las superficies voxelizadas, aunque existen otras aplicaciones para el uso de las representaciones como la graficación por computadora o la comparación de los objetos representados. Para la comparación de superficies voxelizadas, nos interesan las representaciones invariantes ante traslaciones y rotaciones, pues al tener dichas características se evita la manipulación y el preprocesamiento de los objetos, eliminando una fase dentro de la comparación.

Al comparar las representaciones, son de nuestro interés aquellas que analicen las propiedades geométricas y eviten comparaciones estadísticas. Aun no es claro, que tipo de métricas se deben utilizar sobre las representaciones para que estas estructuras puedan ser usadas en la comparación de superficies voxelizadas. Pues los métodos clásicos de comparación de árboles etiquetados y comparación de cadenas no son suficientes para hacer un reconocimiento robusto. Esto sería un trabajo a futuro, se piensa que al utilizar la representación del árbol de bordes podría ayudar a establecer una forma de cómo hacer comparaciones entre los $5ODCCC$ que representan a los árboles, pues no se conocen métricas para comparar los $5ODCCC$ de los árboles.

Bibliografía

[1] Andrea F Abate, Michele Nappi, Daniel Riccio, and Gabriele Sabatino. 2d and 3d face recognition: A survey. *Pattern recognition letters*, 28(14):1885–1906, 2007.

[2] Pierre Alliez and Craig Gotsman. Recent advances in compression of 3D meshes. *Advances in Multiresolution for Geometric Modelling*, pages 1–25, 2005.

[3] Bruce G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, AFIPS '75, pages 589–596, New York, NY, USA, 1975. ACM.

[4] John Adrian Bondy. *Graph Theory With Applications*. Elsevier Science Ltd., Oxford, UK, UK, 1976.

[5] Ernesto Bribiesca. A new chain code. *Pattern Recognition*, 32(2):235 – 251, 1999.

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

[7] Ernesto Bribiesca. Scanning-curves representation for the coverage of surfaces using chain coding. *Computers & Graphics*, 27(1):123–132, 2003.

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

[9] Ernesto Bribiesca, Adolfo Guzmán, and Luis A. Martínez. Enclosing trees. *Pattern Analysis and Applications*, 15(1):1–1, 2012.

[10] Arthur Cayley. Xxviii. on the theory of the analytical forms called trees. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 13(85):172–176, 1857.

[11] Emilie Charrier and Jacques-Olivier Lachaud. Maximal planes and multiscale tangential cover of 3d digital objects. In *International Workshop on Combinatorial Image Analysis*, pages 132–143. Springer, 2011.

[12] Li Chen. *Digital and Discrete Geometry: Theory and Algorithms*. Springer Publishing Company, Incorporated, 2014.

[13] Li Chen and Jianping Zhang. Digital surface definition and fast algorithms for surface decision and tracking. In *Optical Tools for Manufacturing and Advanced Automation*, pages 169–178. International Society for Optics and Photonics, 1993.

[14] Nvidia Corporation. Cuda parallel computing plataform. `http://www.nvidia.com/object/cuda_home_new.html`, 2007. Accessed on 2016-12-06.

[15] Maxime Crochemore and Wojciech Rytter. *Jewels of stringology: text algorithms*. World Scientific, 2003.

[16] Thomas Fabry, Dirk Smeets, and Dirk Vandermeulen. *Surface representations for 3D face recognition*. INTECH Open Access Publisher, 2010.

[17] Herbert Freeman. On the encoding of arbitrary geometric configurations. *Electronic Computers, IRE Transactions on*, EC-10(2):260–268, June 1961.

[18] Herbert Freeman. Computer processing of line-drawing images. *ACM Computing Surveys*, 6(1):57–97, january 1974.

[19] Y. Gerard, I. Debled-Rennesson, and P. Zimmermann. An elementary digital plane recognition algorithm. *Discrete Applied Mathematics*, 151(1-3):169–183, 2005.

[20] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Pearson Education, 2009.

[21] Robert M. Haralick and Linda G. Shapiro. Glossary of computer vision terms. *Pattern Recognition*, 24(1):69 – 93, 1991.

[22] David A Huffman et al. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[23] Reinhard Klette and Azriel Rosenfeld. *Digital Geometry*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufman Publishers, San Francisco, 2004.

[24] Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Computing Surveys*, 47(3):1, 2015.

[25] LA Martínez, E Bribiesca, and A Guzmán. Voxel-based object representation by means of edging trees. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[26] Ajmal Mian. Illumination invariant recognition and 3d reconstruction of faces using desktop optics. *Optics express*, 19(8):7491–7506, Apr 2011.

[27] P. Min. [binvox] 3d mesh voxelizer. `http://www.cs.princeton.edu/~min/binvox/`, 2011. Accessed on Feb 2015.

[28] David G. Morgenthaler and Azriel Rosenfeld. Surfaces in three-dimensional digital images. *Information and Control*, 51(3):227–247, 1981.

[29] Fakir S. Nooruddin and Greg Turk. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9:191–205, 2003.

[30] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter. A 3d face model for pose and illumination invariant face recognition. In *Advanced Video and Signal Based Surveillance, 2009. AVSS '09. Sixth IEEE International Conference on*, pages 296–301, Sept 2009.

[31] Jingliang Peng, Chang-su Kim, and C Jay Kuo. Technologies for 3D mesh compression : A survey. 16:688–733, 2005.

[32] George M Reed and Azriel Rosenfeld. Recognition of surfaces in three-dimensional digital images. *Information and Control*, 53(1-2):108–120, 1982.

[33] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[34] J. Miguel Salazar and Ernesto Bribiesca. Compression of three-dimensional surfaces by means of chain coding. *Optical Engineering*, 54(12):124102, 2015.

[35] Hermilo Sánchez-Cruz and Ernesto Bribiesca. Study of compression efficiency for three-dimensional discrete curves. *Optical Engineering*, 47(7):077206–077206–9, 2008.

[36] Matthew Turk and Alex Pentland. Eigenfaces for Recognition, 1991.

[37] W. T. Tutte. *Graph Theory As I Have Known It*. Series in Mathematics and Its Applications. Oxford Lecture, 1998.

[38] Borut Žalik and Niko Lukač. Chain code lossless compression using move-to-front transform and adaptive run-length encoding. *Signal Processing: Image Communication*, 29(1):96 – 106, 2014.

[39] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM computing surveys (CSUR)*, 35(4):399–458, 2003.