Coding 3D Connected Regions with F26 Chain Code

Osvaldo A. Tapia-Dueñas[1], Hermilo Sánchez-Cruz[1(✉)], Hiram H. López[2], and Humberto Sossa[3,4]

[1] Universidad Autónoma de Aguascalientes, Centro de Ciencias Básicas, Av. Universidad 940, 20131 Aguascalientes, Ags., Mexico
black.osvo@gmail.com, hsanchez@correo.uaa.mx

[2] Department of Mathematical Sciences, Clemson University, Martin Hall O-2, Clemson, SC 29634-0975, USA
hlopezv@clemson.edu

[3] Instituto Politécnico Nacional-CIC, Av. Juan de Dios Bátiz S/N, Gustavo a Madero, 07738 Mexico City, Mexico
hsossa@cic.ipn.mx

[4] Tecnológico de Monterrey, Campus Guadalajara, Av. Gral. Ramón Corona 2514, 45138 Zapopan, Jal., Mexico

Abstract. There are many applications in different fields, as diverse as computer graphics, medical imaging or pattern recognition for industries, where the use of three dimensional objects is needed. By the nature of these objects, it is very important to develop thrifty methods to represent, study and store them. In this paper, a new method to encode surfaces of three-dimensional objects that are not isomorphic to the plane is developed. In the proposed method, a helical path that covers the contour is obtained and then, the Freeman F26 chain code is used to encode the helical path. In order to solve geometric problems to find optimal paths between adjacent slices, a modification of the A star algorithm was carried out. Finally, our proposed method is applied to three-dimensional objects obtained from real data.

Keywords: Voxel-based objects · Chain code
Three-dimensional objects · Helical path

1 Introduction

Today, the representation and recognition of 3D objects is a very active field in computer vision. There is a great amount of applications that require 3D images to solve real-life problems, such as medical images, where 3D imaging plays an important role in supporting experts to provide more accurate diagnostics. There are also applications in the preservation of cultural heritage, games, mechanical construction, security and surveillance, computer-aided design (*i.e.*, CAD systems) and in general, in computer vision and pattern recognition. In the literature have appeared proposals to represent a 3D object through another

I. Batyrshin et al. (Eds.): MICAI 2018, LNAI 11289, pp. 3–14, 2018.
https://doi.org/10.1007/978-3-030-04497-8_1

of smaller dimension, which allows an analysis and recognition. In particular, skeletonization has been used for different reasons [1–3]. Although the skeletons maintain the topological properties of the underlying object and can give qualitative results, the original object losses geometric information [1–5].

Other descriptors notably used are the chain codes, which represent movements through the contour of the object. The way in which the contour is visited and the kind of movements that can be produced by different codes, have been exploited for representation and compression, and has attracted the attention of many researchers [6–9].

For the three-dimensional case, there are important proposals for the use and exploitation of chain codes but they have not been used as heavily as in the two-dimensions, being currently a very fertile field [10–12]. In the literature, coding of three-dimensional objects of surfaces that are isomorphic to the plane is presented [13], however, in this work we address the problem of coding surfaces that are not isomorphic to the plane, taking into account the different geometries that are presented, to solve the problem of finding the shortest path that allows to optimally encode the transition from one slice to another, of the given 3D object to encode.

This paper is organized as follows. In Sect. 2, we provide some definitions. In Sect. 3, we explain how to encode a 3D object in a helical path, whereas in Sect. 4 we describe the algorithm used to find the shortest path between one slice and another. In Sect. 5, the application of our method is presented. Finally, in Sect. 6 we give some conclusions and further work.

2 Definitions

In this section we give the most important concepts and definitions used to throughout the paper.

Definition 1. A *voxel*, v, is a resolution cell of a 3D grid with Cartesian coordinates c(x, y, z) and an intensity value $I_v \in \{0, 1\}$. If $I_v = 0$, we say that the voxel is a 0-*voxel*; on the contrary, we say that it is a 1-*voxel*.

Definition 2. A voxel v_0 can share its faces, edges and/or vertices, depending on the 6, 12, 18 or 26-*neighborhood*, which are defined as follows: $N_6(v_0) = \{v | d_e(v_0, v) = 1\}$, $N_{12}(v_0) = \{v | d_e(v_0, v) = \sqrt{2}\}$, $N_{18}(v_0) = \{v | d_e(v_0, v) \leq \sqrt{2}\}$ and $N_{26}(v_0) = \{v | d_e(v_0, v) \leq \sqrt{3}\}$, where d_e is the Euclidean distance between v_0 and its neighbor voxel, v.

Definition 3. A 3D object is a connected component composed of 1-voxels, which is immersed in a 3D array of columns, rows and slices. Each slice is composed by zero or more connected regions: $\mathcal{R}_0^s$, $\mathcal{R}_1^s$, $\cdots$ $\mathcal{R}_m^s$, where s refers to the s-th slice.

Definition 4. If c_1 are the coordinates of voxel v_1, c_2 the coordinates of voxel v_2, and $b = c_2 - c_1$, then v_1 is in the vicinity of v_2 if and only if $b \in \mathcal{B} = \{(i, j, k)\}$, with $i, j, k \in \{-1, 0, 1\} \backslash \{(0, 0, 0)\}$. The set $\mathcal{B}$ is called the *grid basis*.

Definition 5. A *path* is a sequence of adjacent ordered voxels, $\mathcal{P} = \{v_1, v_2, \ldots, v_p\}$, such that v_1 is adjacent to v_2,v_2 is adjacent to $v_3, \ldots, v_{p-1}$ is adjacent to v_p. The vector set $\mathcal{P}_\mathcal{B} = \{b_1, b_2, \ldots, b_{p-1}\} \subset \mathcal{B}$ it is called *basis of path* $\mathcal{P}$.

Definition 6. We give symbols to each of the elements of the grid base $\mathcal{B}$ as follows: $a = (1,0,0), b = (1,1,0), c = (0,1,0), d = (-1,1,0), e = (-1,0,0)$, $f = (-1,-1,0), g = (0,-1,0), h = (1,-1,0), i = (1,0,1), j = (1,1,1), k = (0,1,1)$ $l = (-1,1,1), m = (-1,0,1), n = (-1,-1,1), o = (0,-1,1), p = (1,-1,1)$, $q = (1,0,-1), r = (1,1,-1), s = (0,1,-1), t = (-1,1,-1), u = (-1,0,-1)$, $v = (-1,-1,-1), w = (0,-1,-1), x = (1,-1,-1), y = (0,0,1), z = (0,0,-1)$, So, the alphabet we use is F26 = $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$, and the coding is obtained when each vector b_k is taken from $P_\mathcal{B} = \{b_1, b_2, \ldots, b_{n-1}\}$ associating its respective symbol in F26.

3 Helical Path to Encode 3D Objects

Helical coding allows the chain codes to save starting coordinates as much as possible, while the region in the current slice is a neighbor of the previous region, there is a path to go from n-th voxel ($v_n(s)$) to the first unvisited voxel ($v_1(s+1)$), which allows to recover the shape of the object without need to know in which coordinate we start to codify.

The helical coding is carried out in the following way:

1. Do $s \rightarrow 0$ and define Z as the number of total slices of the 3D grid.
2. While $s < Z + 1$.
3. Visit the first unvisited voxel, $v_1(s)$, of current slice. If it does not exist $s \rightarrow s + 1$, go to 2.
4. Encode the contour $\mathcal{R}_k^s$ and obtain $v_n(s)$.
5. Find($v_1(s + 1)$). If it does not exist $s \rightarrow 0$, go to 3.
6. $s \rightarrow s + 1$. Go to 2.

The Find($\cdot$) function introduced in the previous algorithm is implemented considering the following cases, which are generated by the geometry of the object. Figure 1 shows an object composed of two slices, where dark voxels in upper slice represent the contour of the region $\mathcal{R}_1^{s+1}$.

Case 1. $v_1(s + 1) \in N_{26}(v_n(s))$ (Fig. 1a).
Case 2. It is possible to draw a discrete straight line from $v_n(s)$ to $v_1(s + 1)$ as the shortest path (Fig. 1b).
Case 3. It is not possible to draw a discrete straight line from $v_n(s)$ to $v_1(s + 1)$, since there are 0-voxels between them, for this reason the path can not be created, unless a concavity must be surrounded. This causes that the original $v_1(s + 1)$ could change, since in the i-th step, a voxel of the contour of $\mathcal{R}_{k'}^{s+1}$ has the smallest d_e with respect to the last voxel of the current path (Fig. 1c).

Case 4. There are two candidates $v_1(s+1)$, because both have the same distance with respect to $v_n(s)$ (Fig. 1d).

We must validate which $\mathcal{R}_{k'}^{s+1}$ is a neighbor of $\mathcal{R}_k^s$, selecting the correct $v_1(s+1)$ we use A* to go from $v_n(s)$ to $v_1(s+1)$. To solve these cases, algorithm A* is detailed in Sect. 4.

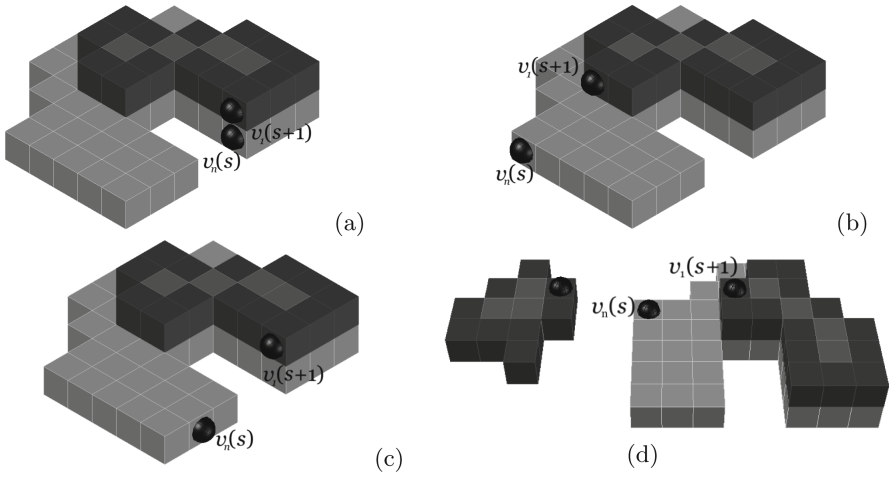


Fig. 1. Four different cases to consider for the starting and target points due to the geometry of the objects.

3.1 Used Symbols of F26

To codify our 3D object, the 26 symbols of F26 are not necessary. Part of the strategy, is to first visit each slice and encode its contours, so we can use eight different vectors for each visited region, $\mathcal{R}_k^s$, since the vectors point to one of the four faces of each voxel, plus four towards the edges. So, the symbols $\{a, b, c, d, e, f, g, h\}$ are required. On the other hand, once the contour is visited, the next step is to move to the contiguous slice, *i.e.*, from $\mathcal{R}_k^s$ to $\mathcal{R}_{k'}^{s+1}$, thus, we need vectors that point to any of its four edges or four vertices, plus one more symbol corresponding to the top face. So, the symbols $\{i, j, k, l, m, n, o, p, q\}$ are used. Therefore, 17 symbols are required, at most.

4 Modifications to A* Algorithm

A* algorithm searches for the shortest path from an initial point to a target point [14]. This heuristic uses information relative to the place where the objective is located to select the next direction to be followed. The formula used to select

the next point in the configuration space is: $f(v) = h(v) + g(v)$, where v is the current cell, $h(v)$ is the heuristic distance (Manhattan, Euclidean or Chebyshev) from v to the destination cell and $g(v)$ is the cumulative cost of moving from the initial state to the state v. Each adjacent point of the current one is evaluated by the formula $f(v)$. The point with the smallest value of $h(v)$ is selected as the next in the sequence [15].

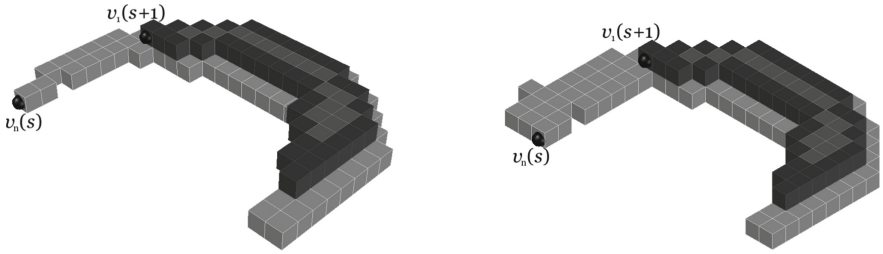
The algorithm A* used for a grid configuration space is restricted to 8-connectivity. This means that we can find a path that is based on the connection between the closest cells. Due to the discretization of the plane, there may be zigzag movements to emulate the straight lines.

In the literature, the target cell does not change under any conditions. In this paper we find that the target point is the closest, and the starting cell is not in the same plane than the target cell. Taking into account these requirements, we modify the A* so that the target cell can be modified.

4.1 Conflict Zones for A*

Since we want to encode the shape of a three-dimensional object with a simple curve, *i.e.* collisions or repeated paths are not permitted, when looking for the shortest path, we avoid to go through 1-voxels that are part of the contour, since it was already visited. A drawback of doing this is that if a column or row only contains 1-voxels that are part of the contour, the A* is not able to find a viable path, so, we must validate this fact before using the algorithm. When this case is presented, we use one of two structuring elements, one to add 1-voxels to the column, and the other to add 1-voxels to the row.

Figure 2(a) shows that going from $v_n(s)$ to $v_1(s+1)$, there is no path that satisfy our conditions. To solve this, we validate each column or/and row that exist between $v_n(s)$ and $v1(s+1)$. If the row or column is one with less than three 1-voxels, the structuring element is used. Figure 2(b) shows the result of applying the structuring element.



(a) Before using structuring elements (b) After using structuring elements

Fig. 2. Example of how to use structuring elements in conflictive zone to use A*.

4.2 Pseudocode of Our A* Modified

For our proposals, we have used A* with adaptations to achieve the optimal path between $v_n(s)$ and $v_1(s+1)$. We introduce the matrices $\mathcal{M}$ and $\mathcal{E}$ composed of 1s and 0s to represent the regions, by following the next steps.

1. If there is a path to go from $v_n(s)$ to $v_1(s+1)$ in $\mathcal{R}_k^s$.
   $\mathcal{M} \rightarrow \mathcal{R}_k^s$.
   $\mathcal{E} \rightarrow \mathcal{R}_{k'}^{s+1}$.
2. else If there is a path to go from $v_n(s)$ to $v_1(s+1)$ in $\mathcal{R}_{k'}^{s+1}$.
   $\mathcal{M} \rightarrow \mathcal{R}_{k'}^{s+1}$.
   $\mathcal{E} \rightarrow \mathcal{R}_k^s$.
3. Fill the matrix h.
   Fill the matrix h with $\times$ if $c(i,j)$ of $\mathcal{M}$ is 1, which represents a 1-voxel of the contour, label with ∞ any other case. We fill the matrix g in the same way as h.
   Consider $c(i,j)$ of $v_n(s)$, and do $d_{ij} \rightarrow 0$.
   Store $c(i,j)$ in open list.
   Do
   For each neighbor $c(i',j')$ of $N_8(c(i,j))$, if its value is equal ∞ in h and 1 in $\mathcal{M}$, calculate the Manhattan distance as following:

$$\begin{aligned} d_{i'j'} &\rightarrow d_{ij} + 1 \quad if \quad c(i',j') \in N_4(c(i,j)) \\ d_{i'j'} &\rightarrow d_{ij} + 2 \quad if \quad c(i',j') \in N_8 c(i,j) \setminus N_4(c(i,j)) \end{aligned} \tag{1}$$

   Each $c(i',j')$ is stored in the open list.
   Search in open list $c(i',j')$, such that $d_{i',j'}$ is the smallest and remove $c(i',j')$ from the open list and do the Eq. (2).

$$\begin{aligned} d_{ij} &\rightarrow d_{i'j'} \\ c(i,j) &\rightarrow c(i',j'). \end{aligned} \tag{2}$$

   While open list is not empty.
4. Fill the matrix g.
   Obtain $c(i,j) \rightarrow v_n(s), d_{ij} \rightarrow 0$.
   Store $c(i,j)$ in open list.
   Do
   For each neighbor $c(i',j')$ of $N_8(c(i,j))$, if its value is equal ∞ in h and 1 in $\mathcal{M}$, do:

$$\begin{aligned} d_{i'j'} &\rightarrow d_{ij} + 10 \quad if \quad c(i',j') \in N_4(c(i,j)) \\ d_{i'j'} &\rightarrow d_{ij} + 14 \quad if \quad c(i',j') \in N_8(c(i,j)) \setminus N_4(c(i,j)). \end{aligned} \tag{3}$$

   Each $c(i',j')$ is stored in the open list.
   Search in open list $c(i',j')$, such that the sum $d_{i'j'}$ in $h + d_{i'j'}$ in g is the smallest and remove $c(i',j')$ from the open list.
   Obtain the Euclidean distance of 1-voxel from the contour of $\mathcal{E}$ to d_{ij}.
   The one with the smallest distance is now $v_1(s+1)$.

While $v_1(s+1) \notin N_8(c(i,j))$.

5. Obtain the shortest path.
   Obtain the $c(i,j)$ of $v_1(i+1)$.
   While $v_1(s+1) \notin N_8(c(i,j))$
      Add $c(i,j)$ to the shortest path.
      Search $c(i',j') \in N_8(c(i,j))$, such that its $d_{i'j'}$ is the smallest in g.
      Assign $c(i,j) \rightarrow c(i',j')$.

To illustrate an example, consider the need to go from $v_n(s)$ to $v_1(s+1)$ through an optimal path, Fig. 1(c) is used for this purpose. To achieve this, let $\mathcal{R}_k^s$ and $\mathcal{R}_{k'}^{s+1}$ the two contiguous regions. If there is a path to go from $v_n(s)$ to $v_1(s+1)$ in $\mathcal{R}_k$, we associate $\mathcal{M}$ with $\mathcal{R}_k^s$ and $\mathcal{E}$ with $\mathcal{R}_{k'}^{s+1}$ (Fig. 3(a) and (b), respectively).

Following our modified A* method, the first step is to fill the matrix h with '×' if $c(i,j)$ of $\mathcal{M}$ is '1', which represents a 1-voxel of the contour, and we fill with ∞ in any other case (see Fig. 3(c)). We fill the matrix g in the same way as h (see Fig. 4).

0	1	1	1	1	1	1
0	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	0	0	0
1	1	1	1	0	0	0
0	1	1	1	1	1	1
0	0	1	1	1	1	1
0	0	1	1	1	1	1

(a)

0	0	0	0	1	1	1
0	0	0	1	1	1	1
0	1	1	1	1	1	1
0	1	1	1	0	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

(b)

∞	×	×	×	×	×	×
∞	×	∞	∞	∞	∞	×
×	∞	∞	∞	×	×	×
×	∞	∞	×	∞	∞	∞
×	∞	∞	×	∞	∞	∞
∞	×	∞	∞	×	×	×
∞	∞	×	∞	∞	∞	×
∞	∞	×	×	×	×	×

(c)

Fig. 3. Matrices (a) $\mathcal{M}$, (b) $\mathcal{E}$, and (c) h, respectively.

Next step is to assign a number greater than or equal to zero to each $c(i,j)$, such that the value of $c(i,j)$ is equal to ∞ in h and '1' in $\mathcal{M}$. This number is the Manhattan distance. We start by making $d_{ij} = 0$ in (i,j) of h, where $c(i,j)$ is the coordinate of $v_n(s)$.

The Manhattan distance is calculated as described in the algorithm of modified A* (Eq. (1)).

The coordinates $c(i',j')$ are stored in the open list to remember the cell to visit their neighbors. We search in this list the coordinates $c(i',j')$, such that $d_{i'j'}$ is the shortest. We remove $c(i',j')$ from the list and solve the assignations of Eq. (2).

We continue with the calculation of the Manhattan distance of each $c(i',j')$ in the open list, until it is empty. The matrix h becomes as in Fig. 5.

If the value of $c(i,j)$ is ∞ in g and '1' in $\mathcal{M}$, we subsequently assign to each $c(i,j)$ a number greater than or equal to zero. We start by making $d_{ij} = 0$ in the cell (i,j) of g, where $c(i,j)$ is the coordinate of $v_n(s)$. Each number is calculated by Eq. (3).

∞	×	×	×	×	×	×
∞	×	∞	∞	∞	∞	×
×	∞	∞	∞	×	×	×
×	∞	∞	×	∞	∞	∞
×	∞	∞	×	∞	∞	∞
∞	×	∞	34	×	×	×
∞	∞	×	30	20	10	0
∞	∞	×	×	×	×	×

Fig. 4. Matrix g

∞	×	×	×	×	×	×
∞	×	9	10	11	12	13
×	9	8	9	×	×	14
×	8	7	×	∞	∞	∞
×	7	6	×	∞	∞	∞
∞	×	5	4	×	×	×
∞	∞	×	3	2	1	0
∞	∞	×	×	×	×	×

Fig. 5. Matrix h.

The coordinates $c(i', j')$ are stored in the open list to remember the coordinates of the square in which we need to visit its neighbors. We search in this list the coordinates $c(i', j')$, such that the sum of $d_{i'j'}$ in h plus $d_{i'j'}$ in g is the smallest, we remove $c(i', j')$ from the list and solve the Eq. (3).

We use $\mathcal{E}$ to obtain the Euclidean distance of each 1-voxel from the contour of $\mathcal{R}_k^s$ to d_{ij}. The one with the smallest Euclidean distance is now $v_1(s+1)$. We continue calculating each $c(i', j')$ that complies with the aforementioned conditions, until $v_1(s+1) \in N_8(c(i, j))$.

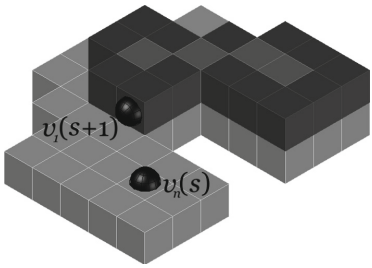


Fig. 6. Final position of $v_1(s+1)$.

On this example, it is true that when surrounding the concavity, $v_1(s+1)$ changes (Fig. 6), because there is a voxel $(v_p(s+1))$ in the contour represented in $\mathcal{E}$, which is the closest to the current voxel $(v_0(s))$. This causes $v_p(s+1)$ to be $v_1(s+1)$. As in the subsequent steps, there is no other $v_p(s+1)$ that is closer to $v_0(s)$, whereas $v_1(s+1)$ does not change until the algorithm ends.

The last step is to find the shortest path between $v_n(s)$ and $v_1(s+1)$. Assign $c(i, j) \rightarrow v_1(s+1)$. While $v_n(s) \notin N_8(c(i, j))$, we do: store $c(i, j)$ in the short path list. Search $c(i', j') \in N_8(c(i, j))$, such that its $d_{i'j'}$ is the smallest in g. Assign $c(i, j) \rightarrow c(i', j')$.

Thus, in our example the optimal path from $v_n(s)$ to the definite target found, $v_1(s+1)$, is $\mathcal{P} = \{(7, 7, 1), (7, 6, 1), (7, 5, 1), (6, 4, 1), (5, 4, 2)\}$, where (7,7,1) and (5,4,2) are the coordinates of $v_n(s)$ and $v_1(s+1)$, respectively.

Finally, the chain code of the helical path representing the two contours is: $S = 771ceefeffgghgaaaaacceedcbaadefpefgahhaacceed$.

5 Experiments

To test our method, we found helical paths to a sample of 3D objects. We used surfaces directly from the voxelization implicit in a data set. These models were previously treated to simplify and eliminate noise in the data.

The models used come from different sources and helped us to test our methodology. We accessed the Stanford Computer Graphics Laboratory site: http://graphics.stanford.edu/data and the Suggestive Contour Gallery site: http://gfx.cs.princeton.edu/proj/sugcon/models.

Figure 7 presents the results of applying our method to obtain the helical path that describes the surface of the contour of each object. The figures show

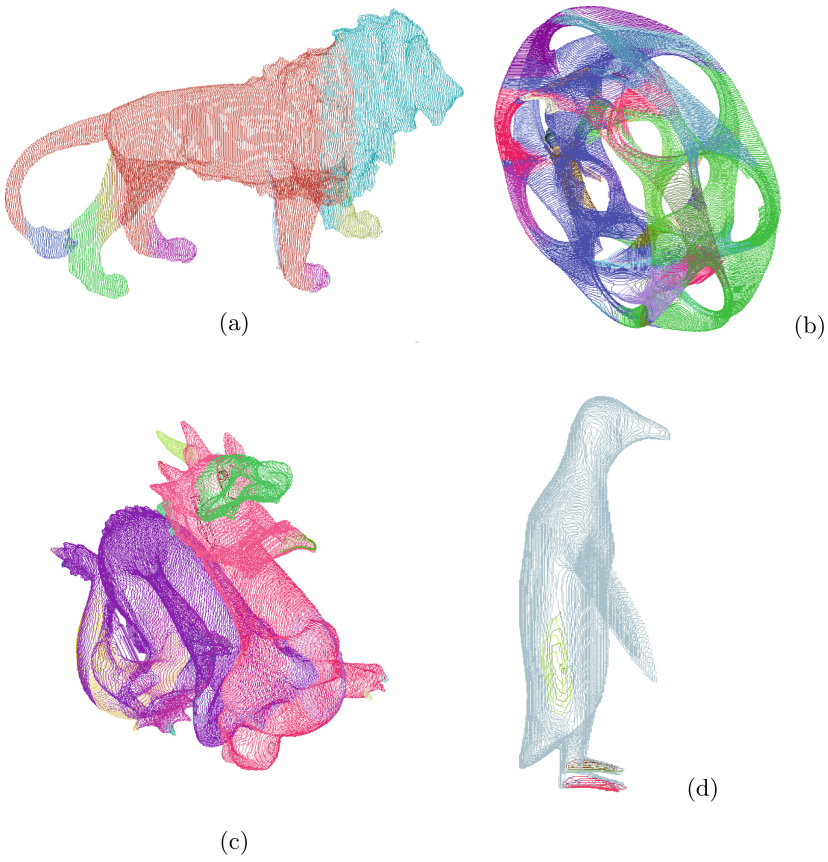


Fig. 7. Helical path found in a sample test: (a) Lion, (b) Heptoroid, (c) Dragon and (d) Penguin (Color figure online)

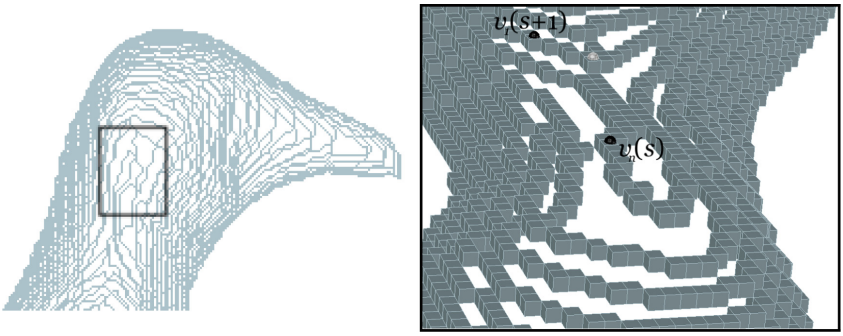


Fig. 8. On the left, the helical path of the penguin head object is shown. On the right, case 3 is exemplified with the contour voxel representation.

Table 1. Frequency of F26 symbols

Symbol	Objects			
	Dragon	Heptoroid	Lion	Penguin
cc	39	42	22	7
a	15934	20912	10414	13440
b	7332	13232	4165	3204
c	8895	27156	5079	2333
d	9252	13175	4003	2015
e	14939	20723	10419	12438
f	5884	13001	4200	3785
g	12524	27392	4828	2095
h	6871	12724	4153	1615
i	301	734	215	39
j	93	168	42	9
k	4	66	5	5
l	30	105	24	27
m	13	21	10	36
n	47	141	50	24
o	17	14	4	0
p	48	110	33	44
q	31	11	3	17
l_{F26}	82215	149685	47647	41126

different colors in their helical path to indicate the different connected components, that recursively were visited to complete the chain coding. Figure 7(a) Lion has 22 different colors, Fig. 7(b) Heptoroid has 42, Fig. 7(b) Heptoroid has 39 and Penguin just has 7.

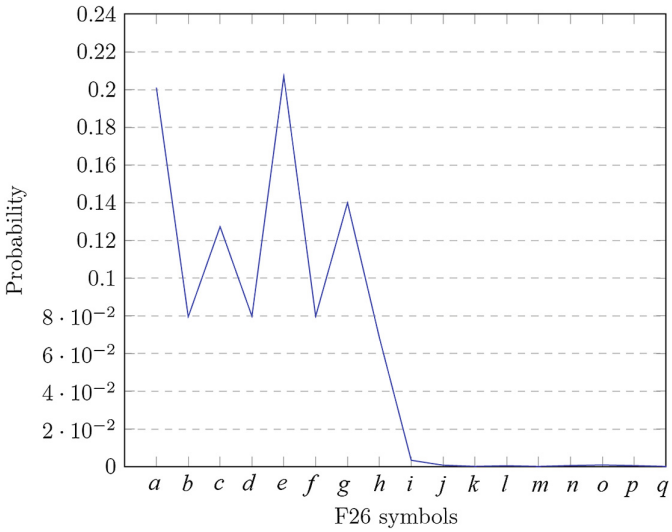


Fig. 9. Probability of occurrence of each symbol.

In Fig. 8 (left) is shown the helical path of the object called Penguin, where Case 3 of our modified A* is exemplified. In this case it is necessary to surround the concavity that exists between $v_n(s)$ and $v_1(s+1)$. This is presented in Fig. 8 (right). This representation only contains the 1-voxels that are part of the contours of coded regions.

On the other hand, Table 1, presents frequencies of the code symbols F26, as well as the length of the chain code (l_{F26}) and the number of connected components (cc) that each object has. The heptoroid object has the longest chain code, while the penguin object has the smallest. As can be seen in the Fig. 9, the presence of the first eight symbols of F26 is much greater than the rest, and of them, symbols a and b are those that appear most likely, followed by g and c. This type of information and analysis is important for dealing with recognition and compression problems.

6 Conclusions and Further Work

We have presented a new method to encode surfaces of voxel-based objects that are not isomorphic to the plane, by means of helical paths. One of the advantages of our work is that the length of the chain code is optimized by visiting the centers of the voxels that make up the contours, as well as finding the shortest trajectories between the slices, preserving the topological and geometric properties. Several applications may result from the work presented here. Information and analysis of the data obtained by the probability of the appearance of the symbols is an important topic to address problems of recognition and compression. So, a future work is to exploit this type of codification for such purposes. As

another future work, it is suggested to analyze the chain code of the helical path, to find dominant points, and thus reduce to a greater extent the information of the shape-of-object without missing valuable information of it.

Acknowledgements. Osvaldo A. Tapia-Dueñas was partially supported by CONACyT. H. Sánchez-Cruz thanks Universidad Autónoma de Aguascalientes, under Grant PII18-8 for the support. Hiram H. López was partially supported by CONACyT, CVU no. 268999, project "Network Codes", and by Universidad Autónoma de Aguascalientes. H. Sossa thanks the Instituto Politécnico Nacional and CONACyT for the economical support under funds: SIP 20180730 and 65 (Fronteras de la Ciencia), respectively to undertake this research.

References

1. Cornea, N.D., Silver, D., Min, P.: Curve-skeleton properties, applications, and algorithms. IEEE Trans. Vis. Comput. Graph. **13**(3), 530–548 (2007)
2. Punam, K., Borgefors, S., Borgefors, G., di Baja, G.S.: A survey on skeletonization algorithms and their applications. Pattern Recogn. Lett. **76**, 3–12 (2016). Special Issue on Skeletonization and its Application
3. Jin, D., Iyer, K.S., Chen, C., Hoffman, E.A., Saha, P.K.: A robust and efficient curve skeletonization algorithm for tree-like objects using minimum cost paths. Pattern Recogn. Lett. **76**(C), 32–40 (2016)
4. Svensson, S., Nyström, I., di Baja, G.S.: Curve skeletonization of surface-like objects in 3D images guided by voxel classification. Pattern Recogn. Lett. **23**(12), 1419–1426 (2002)
5. Arcelli, C., di Baja, G.S., Serino, L.: Distance-driven skeletonization in voxel images. IEEE Trans. Pattern Anal. Mach. Intell. **33**(4), 709–720 (2011)
6. Sánchez-Cruz, H., Rodríguez-Dagnino, R.M.: Compressing bilevel images by means of a three-bit chain code. Opt. Eng. **44**, 44–44–8 (2005)
7. Yong, K.L., Alik, B.: An efficient chain code with Huffman coding. Pattern Recogn. **38**(4), 553–557 (2005)
8. Echávarri, L., Aguinaga, R., Neri-Calderón, A., Rodriguez-Dagnino, R.M.: Compression rates comparison of entropy coding for three-bit chain codes of bilevel images. Opt. Eng. **46**, 46–46–7 (2007)
9. Yong, K.L., Wei, W., Peng, J.W., Alik, B.: Compressed vertex chain codes. Pattern Recogn. **40**(11), 2908–2913 (2007)
10. Freeman, H.: Computer processing of line-drawing images. ACM Comput. Surv. **6**(1), 57–97 (1974)
11. Bribiesca, E.: A chain code for representing 3D curves. Pattern Recogn. **33**(5), 755–765 (2000)
12. Sánchez-Cruz, H., López-Valdez, H., Cuevas, F.J.: A new relative chain code in 3D. Pattern Recogn. **47**(2), 769–788 (2014)
13. Salazar, J.M., Bribiesca, E.: Compression of three-dimensional surfaces by means of chain coding. Opt. Eng. **54**, 54–54–12 (2015)
14. Cui, S.G., Wang, H., Yang, L.: A simulation study of A-star algorithm for robot path planning, pp. 506–509, January 2012
15. Duchó, F., et al.: Path planning with modified a star algorithm for a mobile robot. Proc. Eng. **96**, 59–69 (2014). Modelling of Mechanical and Mechatronic Systems