Representation of 3-D objects using enclosing trees

Ernesto Bribiesca,[1] Adolfo Guzman-Arenas,[2] Luis A. Martínez[3]

[1]Instituto de Investigación en Matemáticas Aplicadas y en Sistemas, UNAM;
[2]Centro de Investigación en Computación, IPN;
[3]Instituto de Astronomía, UNAM
ernesto@leibniz.iimas.unam.mx, a.guzman@acm.org, lamb@astroscu.unam.mx

SUMMARY. A common representation of a 3-D object is a 3-D array of contiguous voxels or elementary cubes –a pile of them, so to speak. The surface of this pile can in turn be represented by 2-D data structures, such as the Hamiltonian graph. In selecting a suitable representation, one seeks to reduce its size (storage needed), to preserve the shape of the initial object, and to obtain a simple representation, easy to compute. Reconstruction of the original object is also important, thus a suitable question is whether the selected representation loses information or it allows complete recovery of the original body. If we could in turn represent the surface by lines, further reduction and simplicity may be achieved.

An *enclosing tree* is a three-dimensional tree formed by 3-D lines totally covering an object (as ivy covers a surface). A *simple line* in an enclosing tree is a sequence of elementary line segments (in 5 possible spatial directions). A *line* is a sequence of simple lines and *forks*, represented just by writing its constituents in sequence. A *fork* or subtree appears where several lines meet (akin to *nodes* of a tree), represented by writing its lines in a given order, using parentheses to keep each line distinct. Thus, a tree is just a *line* with a distinguished starting point, its root. An *enclosing tree* is a tree that totally covers (touches, visits) each voxel of the surface of the solid.

Enclosing trees have interesting properties. They are invariant under rotation and translation. They preserve the shape of the original object. The mirror image of the object and other symmetries are easily obtained from the tree. The size of the enclosing tree is small, so they are suitable for representation of complex 3-D bodies.

The paper shows how to form enclosing trees, presents several examples and examines interesting properties.

Keywords: 3-D objects, 3-D representation, shape numbers, chain code, enclosing tree, voxel.

Representation of 3-D objects using enclosing trees

TABLE OF CONTENTS

7.1 INTRODUCTION AND PREVIOUS WORK

Rigid three-dimensional solids can be usefully represented by a 3-D matrix occupancy array of voxels [1] which are filled with matter –tomography usually uses this voxel representation, and we talk of *voxelized objects*, where voxels have to be face-connected (six-connectivity). Other 3-D representations focus on the surface to be represented –for instance, portions of the Earth surface can be described by digital terrain models formed [12] by a tessellation of tilted triangles with edge-connectivity, or by a 2-D matrix of terrain elevations [5].

This paper seeks to represent arbitrary rigid solids by 3-D lines, without loss of information (full reconstruction is possible). In order to represent a solid by 3-D lines, it is useful to divide its surface into elementary patches or regions, and then to describe 3-D lines that will visit (touch) each of these patches. The algorithm must (a) describe how to segment the surface into elementary patches, (b) how to construct the lines that will visit the patches, and (c) how to represent such lines in compact form. For instance, Earth surfaces can be represented by closed lines joining surface points of equal height, a representation commonly used in maps (terrain elevation curves).

The surface of a voxelized object is formed by the exterior surfaces of the exterior voxels, so that part (a) of the algorithm is already given. In the remaining of this section we will focus on part (c) of the algorithm: how to represent suitable 3-D lines and trees. In Section 7.0 we will generate enclosing trees for a given solid, thus addressing part (b) of the algorithm. Section 7.0 will discuss properties of the enclosing trees and give examples of its use; Section 7.4 presents enclosing trees of some real-world objects; Section 7.5 concludes and discusses the findings.

It is interesting to point that, although we show how to build enclosing trees for voxelized objects, they could also be used for other 3-D representations (not necessarily voxel-based) where the surface can be tessellated into elementary patches; these extensions are outside this paper's scope.

7.1.1 TWO AND THREE-DIMENSIONAL ENTITIES

Certain concepts are hereby defined, in order to explain how to build enclosing trees.

- An *entity* is an isolated portion of the real world. It can be two-dimensional (flat), or three-dimensional, in which case it is called an *object* or *body*. We further restrict ourselves to rigid entities. 2-D entities have as boundaries (with the rest of the real world) one or more 2-D lines (only one if the entity has no holes inside), while objects have as boundaries one or more 3-D surfaces (only one if the entity has no holes inside) [13].
- A 2-D entity can be discretized by overlying on it a 2-D array of pixels, each of area 1 and four edges of length 1. If half or more of a given pixel is covered by the 2-D entity, such pixel belongs to the *discretized 2-D entity*, otherwise it does not. In this manner, a 2-D entity gives rise to a 2-D discretized entity. We use pixels of small enough size so that edge-connectivity (4-connectivity) is enforced. The boundaries of the 2-D discretized entity are formed by elementary lines of size 1 and orientation vertical or horizontal (corresponding to the orientation of the edges of the pixels), as in Fig(a). It is im-

portant to say (but we will not abound on this here) that, in order to preserve shape, the orientation of the axes of the pixels, as well as the size of the pixel relative to the size of the 2-D entity, have to be chosen with care [4] so that, from a 2-D entity, a *unique* 2-d discretized entity is obtained.

- A 3-D entity can be discretized by overlying on it a 3-D array of voxels, each of volume 1, six faces each of area 1, and each face having four edges each of length 1. If half or more of a given voxel is filled by the object, such voxel belongs to the *discretized* or *voxelized object*, and otherwise it does not. In this manner, an object gives rise to a voxelized object. We use voxels small enough so that 6-connectivity (face connectivity) is enforced. The boundaries of the voxelized object are formed by the exterior faces of the exterior voxels. The exterior edges of the surface have only three possible orientations corresponding to those of the overlying voxels. Again, selection of the appropriate voxel size and axes orientations is important for shape comparison, but outside of paper's scope. Nevertheless, see Section 7.5, Discussion.

7.1.2 TWO-DIMENSIONAL LINES AND CHAINS

We focus on 2-D closed lines that are boundaries of 2-D discretized entities. As §7.1.1 made clear, these lines are formed by elementary edges of size 1 and orientation "vertical" and "horizontal."

- The *Freeman chain* (or the chain code) of such 2-D line is the sequence obtained by replacing each edge of the line by one of the numbers 1, 2, 3 or 4, selected according to the direction of such edge [10], see
-
-

- *Fig*, keeping the surface to the right. For our purposes, the starting edge is arbitrary, but we can become independent of it by claiming that the sequence is circular, since the line is closed.

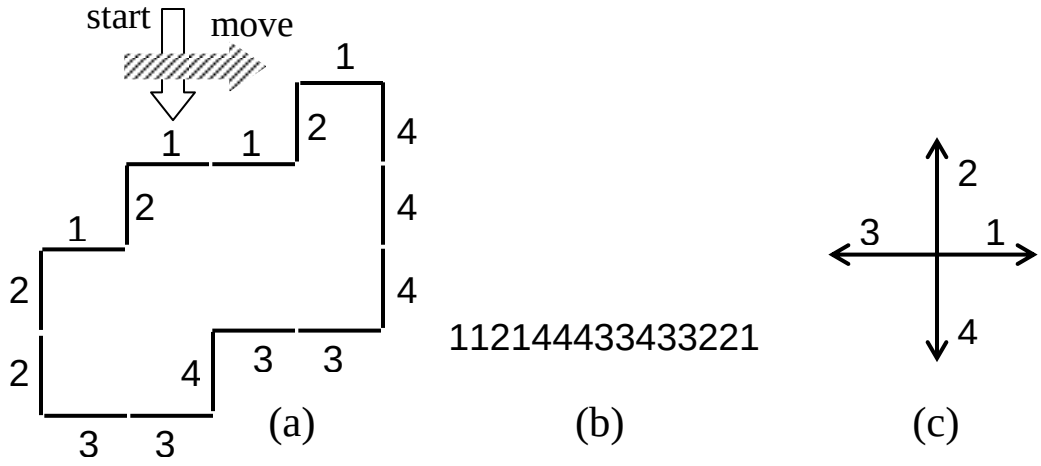


Fig 7.1. Coding the directions of a 2-d closed line. (a) A 2-D closed line. (b) Its Freeman chain. (c) The four possible directions to encode, as the encoder travels the perimeter keeping the 2-D entity to its right

- The *shape number* of such 2-D line is the sequence obtained by replacing each corner of the line by one of the numbers 1, 2 or 3, selected according to the shape of such edge, see [4], and traveling the boundary keeping the surface to the right. The corner to start encoding is arbitrary. To be independent of it, shape numbers are considered circular, so that an obtained shape number is circularly-shifted until a number with the smallest numerical value is obtained, and that is *the* shape number. For instance, in Fig the obtained shape number is 1231122123121213, which is then circularly-shifted until the smallest number, 1122123121213123, is obtained. 1122123121213123 is the shape number of Fig(a). Shape numbers are useful in that they allow easy shape comparison, explained elsewhere [4]. We can think of the shape numbers as being the derivative of Freeman chains.

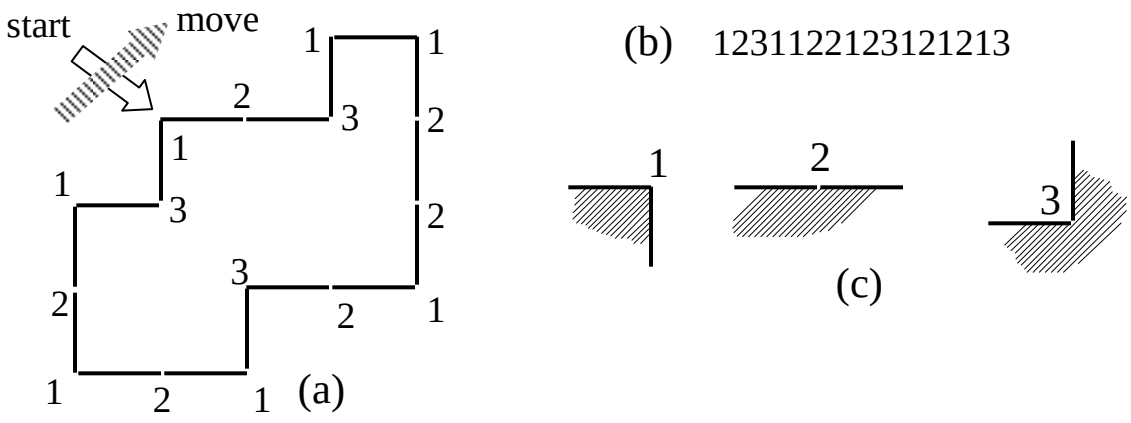


Fig.7.2 Coding the changes in direction of a two dimensional line. (a) A 2-D closed line. (b) Its initially-obtained shape number. (c) The three possible types of corners to encode. After circular shifting of (b) so as to minimize its value as a number, the shape number of line (a) is 1 1 2 2 1 2 3 1 2 1 2 1 3 1 2 3

7.1.3 THREE-DIMENSIONAL LINES AND CHAINS

An arbitrary 3-D line can be discretized by reducing it to a sequence of elementary segments, each of size 1, each having an orientation parallel to one of the axes *x, y* or *z.* As we travel along the line, there are six possible orientations (Fig(a)) for each segment.

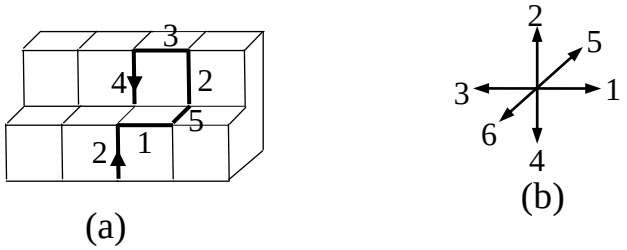


Fig.7.3. Coding the directions of a 3-D line. (a) A 3-D discretized line. (b) The six possible directions of travel. The line (a) is coded as 2 1 5 2 3 4

Instead of coding the directions of each segment, we could code the *change of direction.* There are five possible changes or turns (

Fig(a)). For instance,

Fig(b) shows the turn "3" [14]. The two segments that comprise the first turn constitute the "handle" [see the hand holding the handle in

Fig(a)], which is not coded.

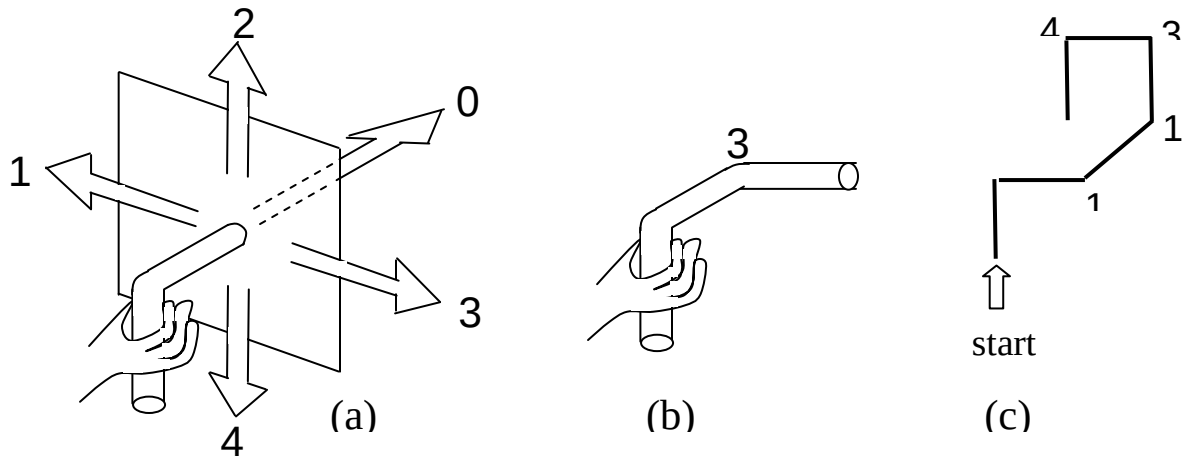


Fig.7.4. Coding turns (changes in direction) in three dimensions. (a) The five possible turns or changes. (b) The change "3." (c) The 3-D discretized line of Fig(a) coded using the turns of (a), obtaining the chain 1 1 3 4

Having obtained chains for 3-D lines or "wires," we now show how to describe trees of such lines.

7.1.4 THE TREE DESCRIPTOR

This section summarizes the initial part of [7].

Fig(a)-(e) illustrate the only five possible chain elements [6, 14]. They summarize the rules for labeling the vertices: to a straight-angle vertex, a "0" is attached; to a right-angle vertex corresponds one of the other labels, depending on the position of such an angle with

respect to the preceding right angle in the path. If the consecutive sides of the reference angle have respective directions b and c (see

Fig(a)), and the side from the vertex to be labeled has direction d (from here on, by direction, we understand a vector of length 1), then the chain element or label is given by the following function,

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

where $\times$ denotes the vector product in $\Re^3$.

7.1.5ALGORITHM TO OBTAIN THE TREE DESCRIPTOR OF A TREE

1. Select an arbitrary starting point of the tree. Select the first turn after this starting point as a reference (not to be labeled).*In*
2.
3.

4. *Fig(f) the starting point is signaled by a small sphere.*
5. Travel along the line as long as it is an elementary line (a sequence of segments), coding each turn in sequence, until a fork is reached (if any). Label the fork as

step (3) indicates. Return (as output of the algorithm) the label of the elementary line followed by the label of the fork.

In the example (

Fig(f)), the second turn (the first to be labeled) is labeled "3", following

Fig*(b). The third turn is "3", too, since the line turns as*

Fig*(b) does. The fourth turn is "1". The fifth and sixth turns (they really go straight) are labeled "0". So far, the elementary line is 333100. Now we come to a fork.*

6. To label each fork, place in parentheses the labeling of each line (using recursively step 2 of the algorithm) of the fork. Order these lines in numerical order.

In the example, we came to a fork. It is formed by three lines. These lines are 2), (1 2 0) and (4 3 3), respectively. The first turn of the line (2 2) is indeed labeled 2 because we entered the fork, we were coming from the direction 1 0 0 so that, according to

Fig(c), the turn indicated a "2." For the same reason, the first turn of the line (4 3 3) is labeled "4" because it follows the pattern of

Fig(e). The three lines (2 2), (1 2 0) and (4 3 3) are ordered by lowest initial starting turn, so their new order is (1 2 0) (2 2) (4 3 3). The final result for the example is 333100(120)(22)(433).

A complete review of the tree notation is in [7].

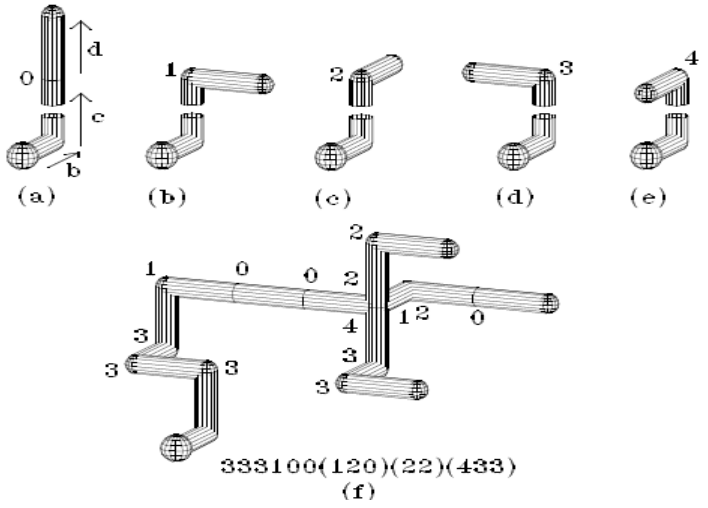


Fig.7.5. The tree descriptor. (a) The element "0." (b) The element "1." (c) The element "2." (d) The element "3." (e) The element "4." (f) A tree of 3-D lines and its tree descriptor: 333100 (120) (22) (433). This tree has four lines: 333100, 120, 22 and 433, and a fork: (120)(22)(433). A fork lists its lines in numerical order of its first turn; thus, 120 precedes 22 (1 is before 2)

7.2 GENERATION OF ENCLOSING TREES

Knowing how to construct the tree descriptor, the next step is to construct a tree that totally covers the surface of a voxelized object. Such tree is called an *enclosing tree.* By "total coverage" we mean that each corner of the surface will be visited by exactly one line of the tree. In this manner, we capture the 3-D coordinates of the surface in the description of the tree, so that (a) the original voxelized body can be reconstructed without loss of information, and (b) the geometrical and shape properties of the body can be derived from the corresponding tree descriptor.

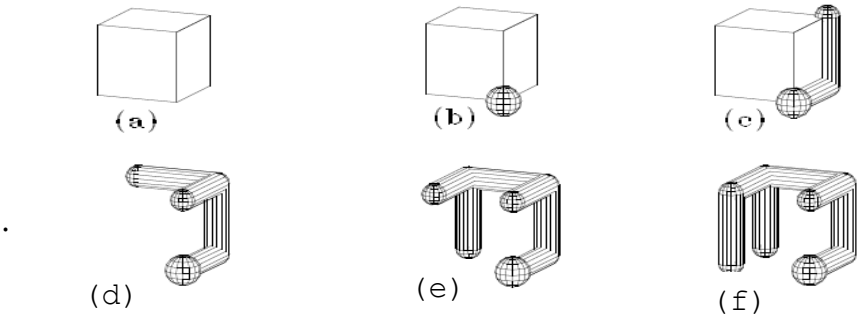


Fig.7.6. Formation of the enclosing tree. (a) A voxelized object consisting of just one voxel. (b) An origin is selected. (c) A turn in direction is selected as reference (not to be labeled). (d) The tree grows in all possible directions, two in our example. One of them is direction 4, the other is direction 3. We have arrived to a fork. We order the lines of the

forks in descending numerical order of its first turn, thus: (3) (4). In the next step we grow each end, starting from the left of the tree descriptor (that is, width-first). (e) shows the growth of (3), which grows first. We then try to grow (4), but it can not grow (all the reachable vertices are already occupied by another part of the tree). So far we have (3 (1) (4)) (4). (f) In the next and last step we grow (1), and we get (1 3). All the vertices have been covered, and the tree descriptor is (3 (1 3) (4)) (4). We simplify parentheses surrounding a single turn when no ambiguity arises, to obtain (3 (1 3) 4) 4 as the final tree descriptor of (a)

7.2.1ALGORITHM THAT BUILDS AN ENCLOSING TREE

1. Select an arbitrary point to start the tree. Select arbitrarily a turn of direction (change in the direction of two consecutive segments) from that starting point (not to be labeled).
   An example in Fig(b) shows the starting point. Fig(c) shows the initial turn of direction or "handle."
2. Grow each branch of the tree one step at a time, from left to right in the descriptor (that is, *width first*). If coming to a fork, grow first in the direction 0, then in the direction 1, and so on. If a vertex of the surface is already occupied, don't grow into it.
   In our example, Fig(d) shows the first growth, in directions 3 and 4. So far, the descriptor is (3) (4). Notice that it is not (3 4), which indicates that turn 4 is after turn 3. It is (3) (4), because turns 3 and 4 occur at the same vertex in our example.
3. Repeat step 2 until no growth occurs. Omit parenthesis surrounding a single turn when no change in meaning occurs. (For instance, (1 (3)) can be changed into (1 3) but (1 (3) (2)) stays unaltered.
   In our example, in the second growth, (3) grows first [into (3 (1) (4))] and then (4) [which can not grow]. The tree descriptor so far is (3 (1) (4)) (4). See Fig(e). In the third growth, 3 grows first, then (1), then (4), then (4), left to right. 3 already grew. (1) grows into (1 3). The first (4) can not grow. The last (4) already tried, unsuccessfully, to grow. See Fig(f). The final tree is (3 (1 3) (4)) (4)

7.2.2AN EXAMPLE WITH SIX GROWTHS

Fig(a) shows an 18-voxel object.

Fig (b) shows an initial starting position and starting turn in it, as well as the first growth, yielding (0) (2) (4), as seen in
Fig(a).
The second growth (

Fig(c)) occurs as follows (see
Fig(b)):
grows into (0 (0) (2) (4)). The new growths are underlined.

(2) grows into (2 (0) (4)).
(4) grows into (4 (3)). The tree so far is (0(0)(2)(4)) (2(0)(4)) (4(3)).
The third growth (

Fig(d)) occurs as follows (see
Fig(c)):
(0 (0) (2) (4)) grows into (0(0(2)(3)(4))(2(0))(4(3)))
(2 (0) (4)) grows into (2(0(1)(4))(4(1)))
(4 (3)) grows into (4(3(0)(1))). The tree so far is (0(0(2)(3)(4))(2(0))(4(3))) (2(0(1)(4))(4(1))) (4(3(0)(1))).
In the fourth growth (

Fig(e)), the tree becomes (
Fig(d)) (new growths are underlined): (0(0(2(0)(1))(3(0)(1))(4))(2(0(1)))(4(3(0))))
(2(0(1(0)(3))(4))(4(1(0)(3))))(4(3(0(4)(1))(1))).
In the fifth growth (

Fig(f)), the tree grows to (
Fig(e)): (0(0(2(0(1))(1(0)))(3(0(1)(4))(1))(4))(2(0(1(0))))(4(3(0))))
(2(0(1(0(3)(4))(3))(4))(4(1(0(3))(3))))(4(3(0(4)(1))(1))).
In the last growth (

Fig(g)), the tree grows to (
Fig(f)); (0(0(2(0(1(0)))(1(0(3))))(3(0(1)(4))(1))(4))(2(0(1(0))))(4(3(0))))
(2(0(1(0(3)(4))(3))(4))(4(1(0(3))(3))))(4(3(0(4)(1))(1))). Notice how, in a fork, the branches starting with lower numbers tend to grow more, since they are the first to grow.

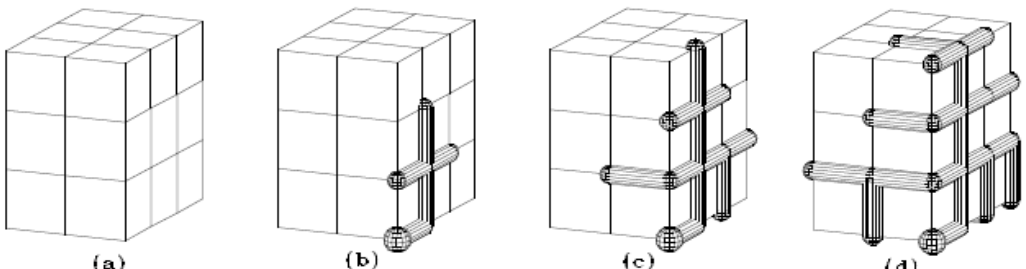


Fig.7.7. Example 0 of an enclosing tree. (a) A voxelized object. (b) An origin is selected, the first turn is selected and the first growth, (0)(2)(4) occurs. (c) The second growth yields (0(0)(2)(4)) (2(0)(4)) (4(3)). (d) The third growth of the tree. (e) The fourth growth. (f) The fifth growth. (g) The sixth and final growth

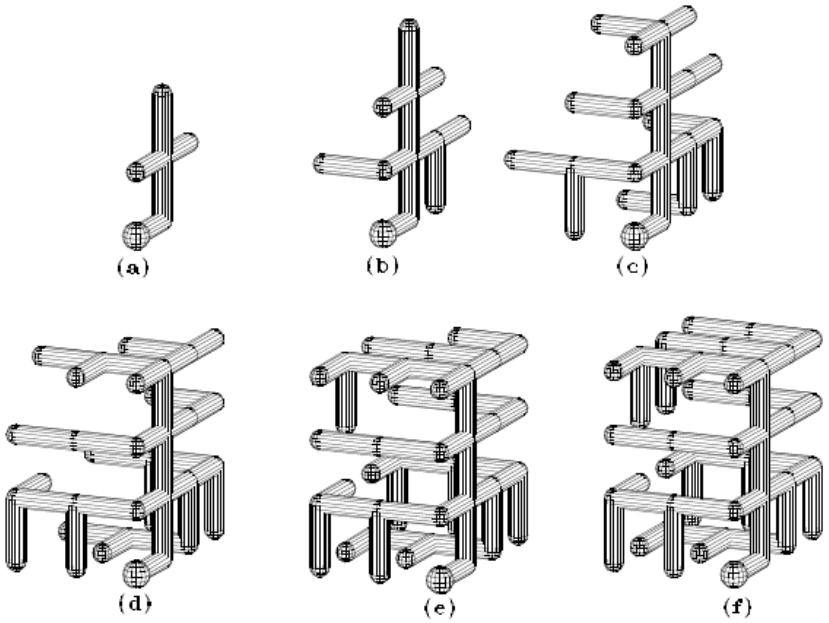


Fig.7.8. The corresponding trees for

Fig. (a) The tree after the first growth. (b) After the second growth. (c) After the third. (d) After the fourth. (e) After the fifth. (f) The sixth growth produces the final enclosing tree of

Fig(a)

7.2.3ANOTHER EXAMPLE

Fig.7.9 shows another voxelized object and the step by step formation of its enclosing tree. Fig.7.10 shows the growth stages of its enclosing tree. The enclosing tree is (0(0(3(1))(4))(1(3))(2(1))(3(0(1(0)))(1(0)))(4(0))) (1(3))(2(1))(3(0(1(0)))(4))(4(0(3))(3(1))).

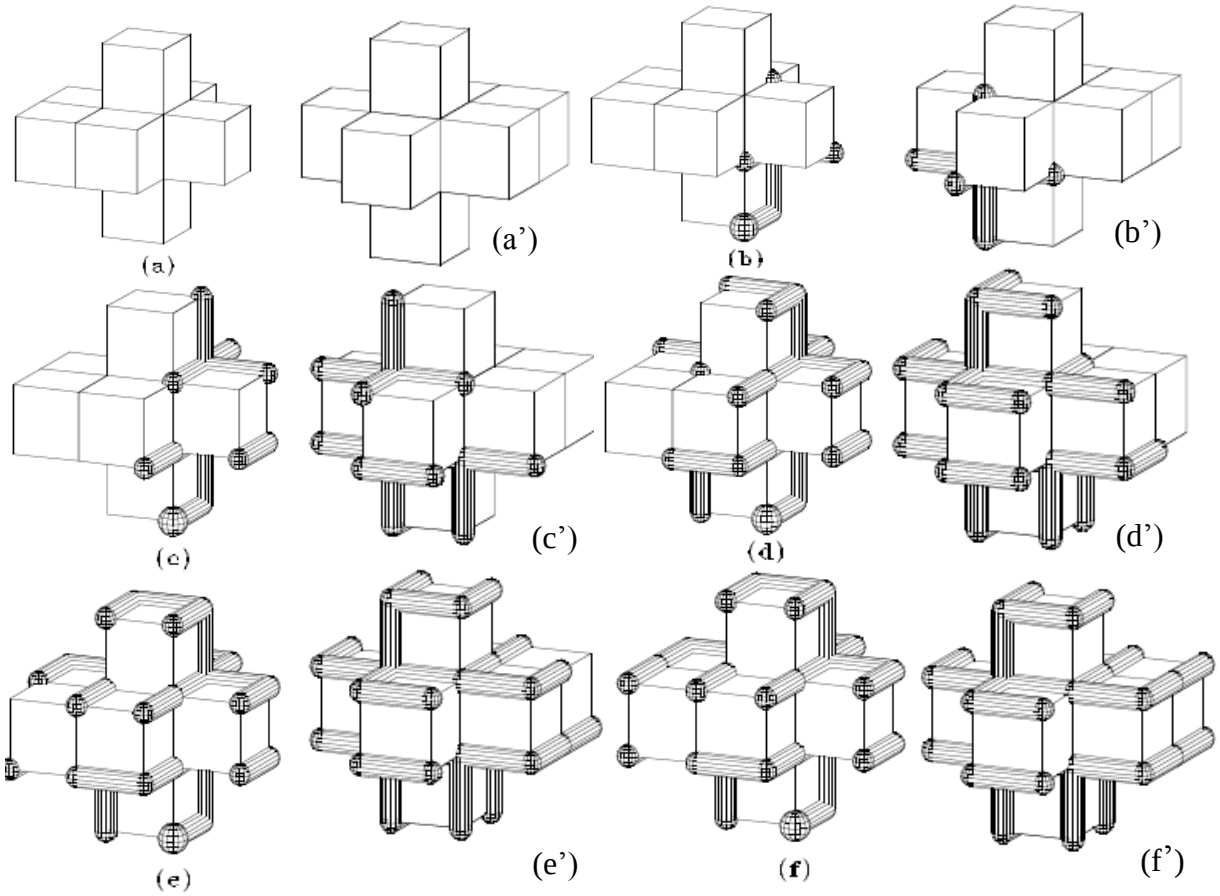


Fig.7.9. Example 0 of an enclosing tree. (a) An object whose enclosing tree is desired. (a') The same object seen from behind (180° rotation). (b) The first growth of the enclosing tree. (b') Figure (b) rotated 180°. (c) and (c') The second growth and its 180° rotation. (d) and (d') The third growth of the enclosing tree and its 180° rotation. (e) and (e') The fourth growth and its rotation. (f) and (f') The final enclosing tree and its 180° rotation

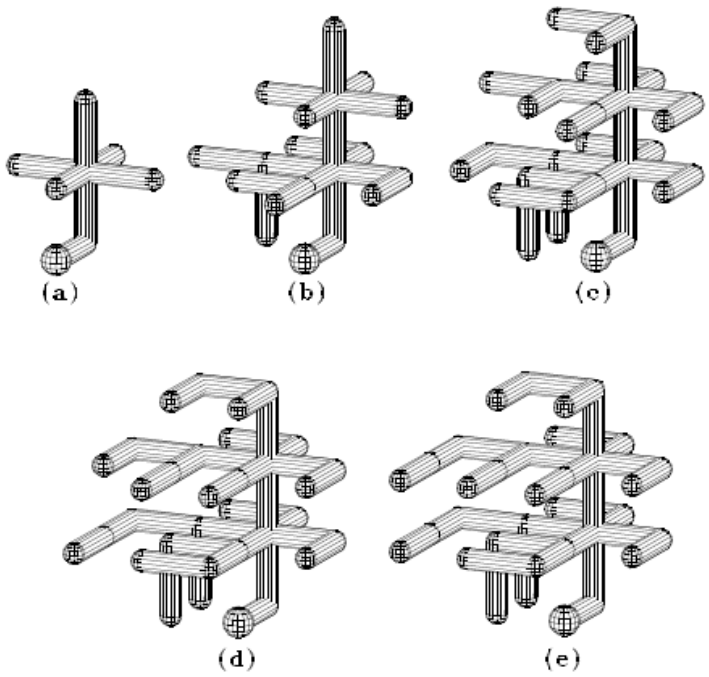


Fig.7.10. The growth stages of the enclosing tree of Fig. (a) First growth. (b) Second growth. (c) Third growth. (d) Fourth growth. (e) Fifth and final growth; the enclosed tree has been formed

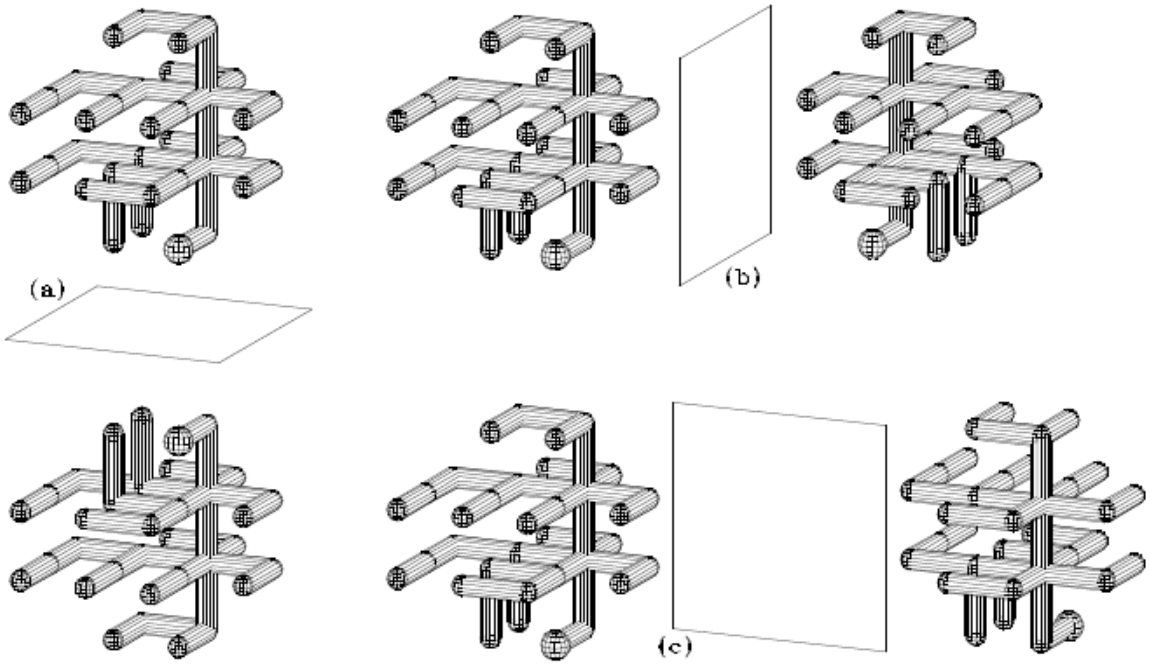


Fig.7.11. Mirror images of the enclosing tree of Fig(e). (a) The reflection is done through the plane XY. (b) Through the plane XZ. (c) Through the plane YZ

7.3PROPERTIES AND EXAMPLES

Some properties of the enclosing tree representation are now discussed.

7.3.1 MIRROR IMAGES

It is easy to obtain the mirror image of an enclosing tree.

Given an object *O* and its enclosing tree *t*, to obtain the enclosing tree *t*' that describes the mirror object *O*', replace in *t* each occurrence of 1 by 3, and each occurrence of 3 by 1, keeping unchanged the nested parentheses.

Example: The enclosing tree of Fig(e) is (0(0(3(1))(4))(1(3))(2(1))(3(0(1(0)))(1(0)))(4(0))) (1(3))(2(1))(3(0(1(0)))(4))(4(0(3))(3(1))).

Exchanging each 1 by a 3 and vice versa, a new enclosing tree *t*' is obtained:
(0(0(1(3))(4))(3(1))(2(3))(1(0(3(0)))(3(0)))(4(0))) (3(1))(2(3))(1(0(3(0)))(4))(4(0(1))(1(3)))

Examining Fig, we see that all mirror images (reflections) of Fig(e) are described by such tree *t*'. This is regardless of whether the reflection is made in the XY plane, the YZ plane or the XZ plane.

In order to see clearly the replacement of 1 by 3 and vice versa, we have not ordered the parentheses in *t*' from lower to higher starting values, as step 3 of algorithm 7.0 indicates.

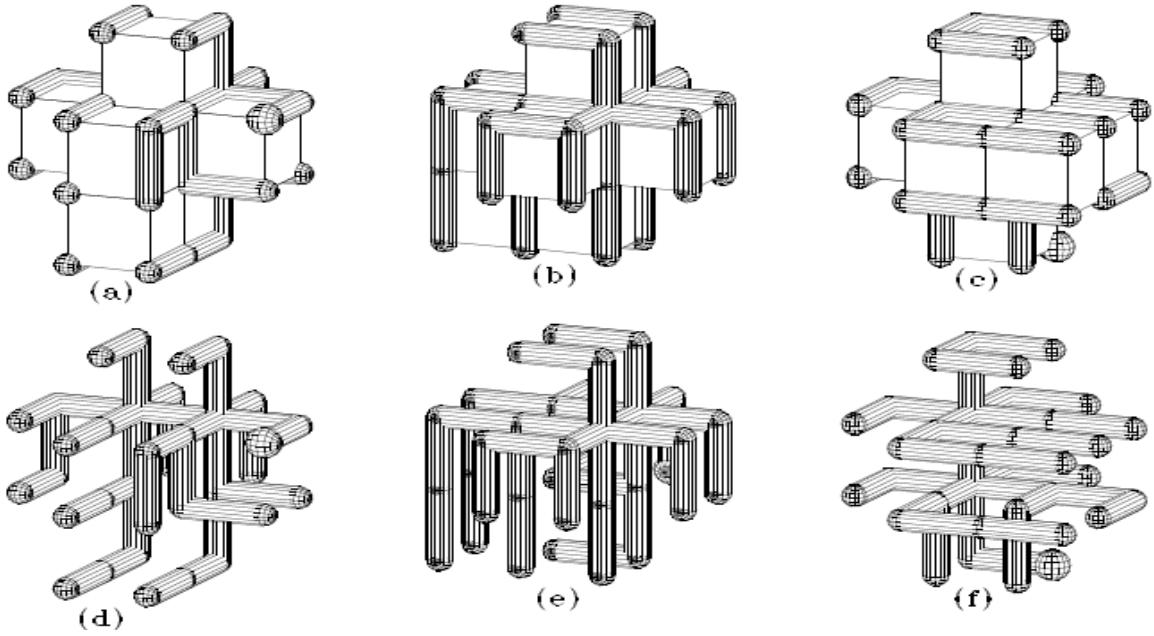


Fig.7.12. The effect of rotation on enclosing trees. (a) Rotation around the X axis of the object in

Fig(f), whose tree is given in Fig(e). (b) Around the Y axis, same object. (c) Around the Z axis. (d) The enclosing tree of object shown in (a). (e) The enclosing tree of object shown in (b). (f) The enclosing tree of object shown in (c). The enclosing tree does not change; it is the same as that of Fig(e)

7.3.2 ROTATION

Independence of rotation. Given an object *O* and its enclosing tree *t*, the enclosing tree of the rotated object *O'* is also *t*, regardless of the rotation taking place around the X axis, the Y axis or the Z axis.

Example. The tree descriptor of Fig(e**)** (corresponding to object in

Fig(f)) is (0(0(3(1))(4))(1(3))(2(1))(3(0(1(0)))(1(0)))(4(0)))(1(3))(2(1))(3(0(1(0)))(4))(4(0(3))(3(1))).

The tree of

Fig(d), representing body in Fig(e**)** rotated around the X axis, is (0(0(3(1))(4))(1(3))(2(1))(3(0(1(0)))(1(0)))(4(0))) (1(3))(2(1))(3(0(1(0)))(4))(4(0(3))(3(1))). The same tree descriptor represents

Fig(e) and

Fig(f). Thus, enclosing trees are invariant under rotation. For symmetry with respect to the origin, see Section 7.0.

7.3.3 COMPRESSION OF OBJECTS

For pattern recognition, shape comparison and image processing, compact representations are important. The enclosing tree is a compact representation, thus becoming a powerful tool for storing voxel-based objects. It is a lossless compression. However, there are more powerful methods (higher compression rates), such as Huffman coding, relying on variable length codes, which applies short codes to highly popular features. In comparison, the enclosing tree is a fixed length code.

7.3.4 GEOMETRICAL PROPERTIES

The voxelized object is exactly described by its surface, which is in turn exactly described by its vertices, and the enclosing tree covers all of them. Thus, their position can be recovered and the initial object can be fully recovered without information loss.

The number of chain elements in a tree descriptor is $m - 3$, where m is the number of surface vertices. This is because the first three points of the tree descriptor correspond to the two contiguous straight-line segments which are considered only for reference.

The surface of the object can have holes; it will still be covered by a single enclosing tree. Think of a gruyere cheese. Nevertheless, if the object can have inner holes (not connected to the outer surface), a separate enclosing tree for each interior surface is needed.

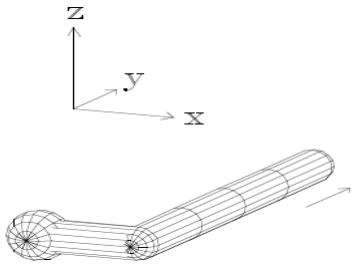


Fig.7.13. The initial plane of the "handle" is x–y; hence, the initial plane of growth of the line is x–y because in that plane is where the handle lies

7.3.5 COMPUTING THE PLANE OF GROWTH OF A LINE

The "plane of growth of a line" is the plane of the current "handle." In Fig the plane of growth is the x–y plane.

The plane of growth does not change as long as we keep adding 2's and 4's to the line. (Rigorously, addition of a 2 changes plane *b–a* into *a–b*, and adding a 4 changes *a–b* into *b–a*, but we usually make no such distinction). Here, *a*, *b* and *c* represent three different axes, 1/3 represents "1 or 3" and 2/4 represents "2 or 4".

Adding a 1 or a 3 changes the plane of growth according to the following equations:

$a\ b + 0 = a\ b$	(adding a 0 keeps the same plane of growth)
$a\ b + 1/3 = b\ c$	(adding a 1 or a 3 changes the plane of growth to *b c*)
$a\ b + 2/4 = b\ a$	(adding a 2 or a 4 changes the plane of growth to *b a*).

For instance, the second equation tells us that if the plane of growth is x y, and we add a 1, the new plane of growth is now y z.

7.3.6 COMPARISON OF ENCLOSING TREES WITH HAMILTONIAN REPRESENTATION

The Hamiltonian representation [17] does not keep the shape of voxelized solids; two different objects may have the same adjacency graph. Also, the enclosing tree is a one-dimensional string, while the adjacency graph is usually represented as a matrix or a graph. In

Fig we see an object and its corresponding enclosing tree. The tree is (2(0(1(0(0(3))(3))(3))(4))(1(0(0(3))(3))(3))(4))(3(0(0(1(3))(4))(1(3))(4))(1(3))(4))(4)

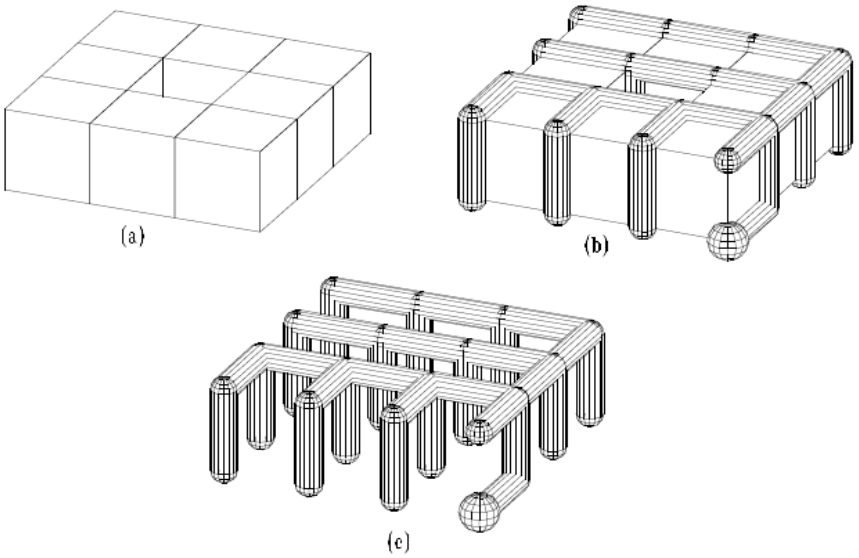


Fig.7.14. Comparison with the adjacency graph. (a) A voxelized object. (b) Growing the enclosing tree. (c) The resulting enclosing tree

The adjacency graph of

Fig(a) is shown in Fig.7.16, which uses the labels of visible faces shown in Fig.

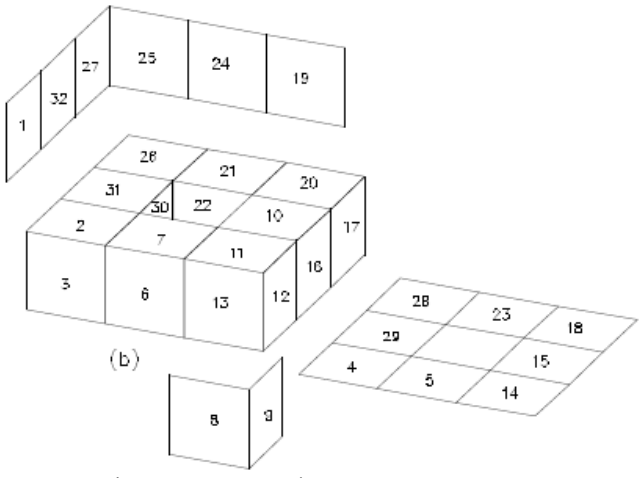


Fig.7.15. The labels of the adjacency graph of the solid of

Fig(a)

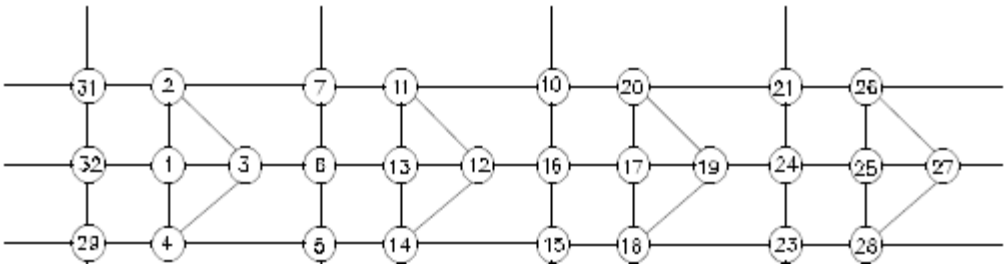


Fig.7.16. The face adjacency graph of the solid of

Fig(a)

7.4 RESULTS

Some examples of enclosing trees of real world objects are now presented. Section 7.0 uses data coming from digital elevation models (DEMs). DEMs are digital representations of the earth's surface. Generally speaking a DEM is generated as a uniform rectangular grid organized in profiles. In order to analyze DEM data by means of the proposed enclosing-tree notation, the models are represented as binary solids composed of regular polyhedrons (voxels).

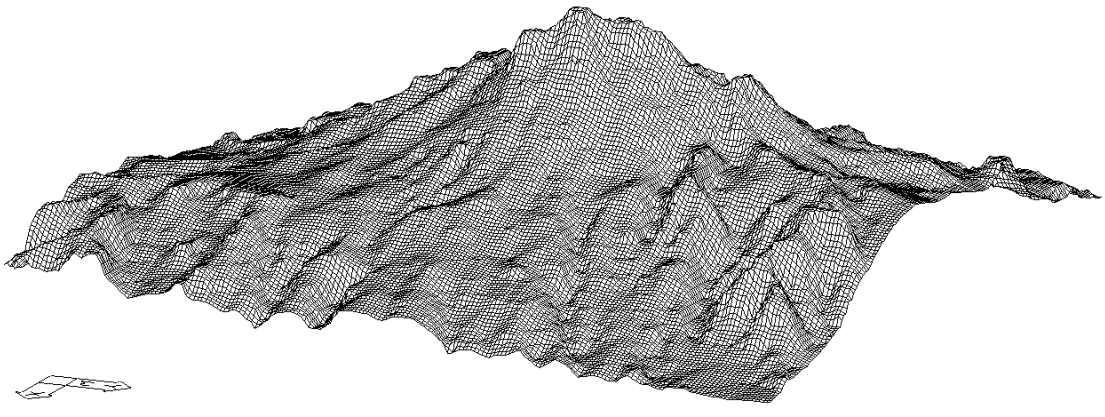
DEMs have a large number of applications [15]. Some of these applications include: urban and regional planning, production of slope, aspect, hill shaded maps, engineering calculations, geomorphology, navigation, line-of-sight calculations, components in complex models, and geographic information systems. Thus, the importance of new representations on DEM data is evident.

In this work the DEMs are represented as three-dimensional (3D) arrays of cells (voxels) which are marked as filled with matter [1]. Several authors have been using different kinds of representations for solids: constructive solid geometry schemes are presented in [3, 18]; generalized cylinders as 3D volumetric primitives are shown in [8]; rigid solids represented by their boundaries or enclosing surfaces are shown in [2]; and superquadrics in [16].

Fig.7.17. Iztaccíhuatl, the "Woman in White," presents the profile of a sleeping woman as seen from the Valley of Mexico. A series of overlapping cones constructed along a NNW-SSE line to the south of the Pleistocene Llano Grande caldera forms the summit ridge of the massive 450 cu km volcano. The mountain has four peaks, the highest of which is 5,230 m high. Together, the peaks are seen as depicting the head, chest, knees and feet of a sleeping female figure. Iztaccíhuatl is at 70 km to the southeast of Mexico City

7.4.1.1THE IZTACCÍHUATL VOLCANO

Iztaccíhuatl or **Ixtaccíhuatl** (5,230 m), is the third highest mountain in Mexico, after the Pico de Orizaba (5,636 m) and Popocatépetl (5,426 m). Its name is Nahuatl for "white woman" (since it is snow-covered). A picture of the White Woman is given in Fig. First, a mesh of latitude and longitude arcs, 3 seconds apart, cover the mountain (Fig.7.18). In it, the height has been exaggerated 2.5 times –compare with Fig). This mesh is used to find the



surface voxels. The voxelized Iztaccíhuatl volcano appears in Fig, and the corresponding enclosing tree can be seen in Fig.

Fig.7.18. A mesh of segment of arcs, 3 seconds apart, now cover the surface of the Volcano

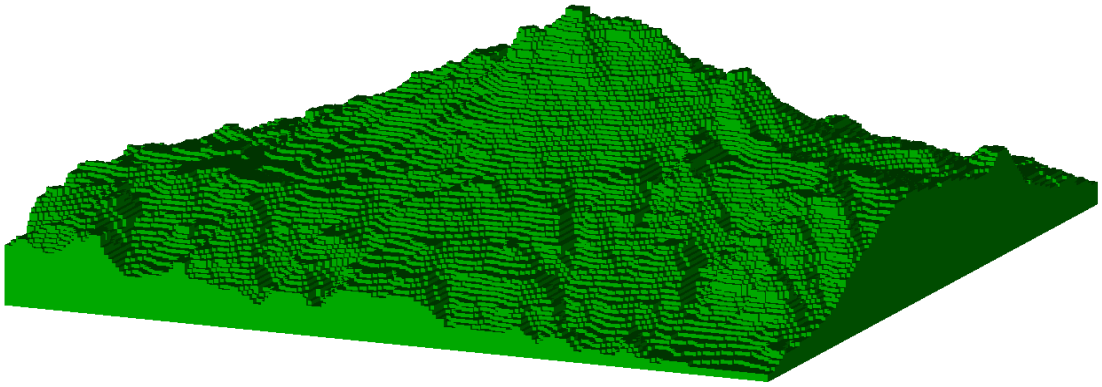


Fig.7.19. The voxelized Iztaccíhuatl, obtained from Figure 18. Each voxel represents on the surface 3 seconds of longitude by 3 seconds of latitude; at the mountain's latitude, 1 seg ≈27m. Thus, a voxel is about 81 x 81 x 81 cubic meters

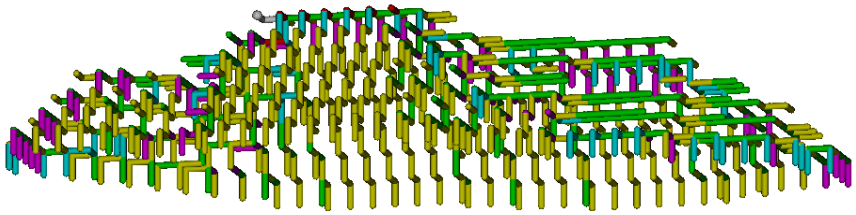
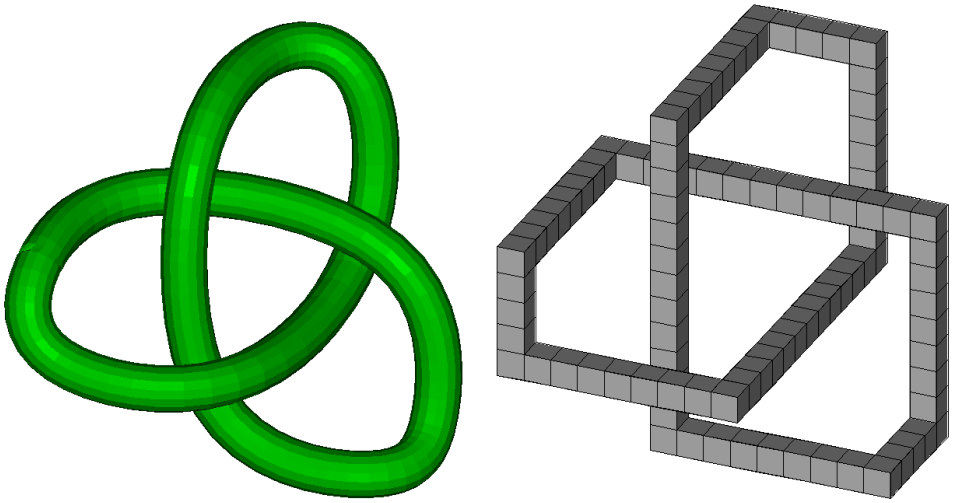


Fig.7.20. The enclosing tree of Iztaccíhuatl. This tree is formed by 3075 characters, which appear elsewhere in the text

The enclosing tree for Iztaccíhuatl (3075 characters) is

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

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



(a) (b)

Fig.7.21. The trefoil knot. (a) The knot. (b) The voxelized trefoil knot

7.4.1.2THE TREFOIL KNOT

The study of simple closed space curves (which can be knotted) is an important topic in computer vision. A knot *K* is a simple closed polygonal curve in three-dimensional Euclidean space R^3. The study of knots has important applications in the synthesis of new molecules, quantum field theory, in DNA research, graph theory, statistical mechanics, etc. The *trefoil knot* [Fig(a)] is the simplest example of a nontrivial knot. The trefoil can be obtained by joining together the two loose ends of a common overhand knot, resulting in a knotted loop. The voxelized trefoil knot is shown in Fig(b). Its enclosing tree (1263 characters), shown graphically in

Fig, is:

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

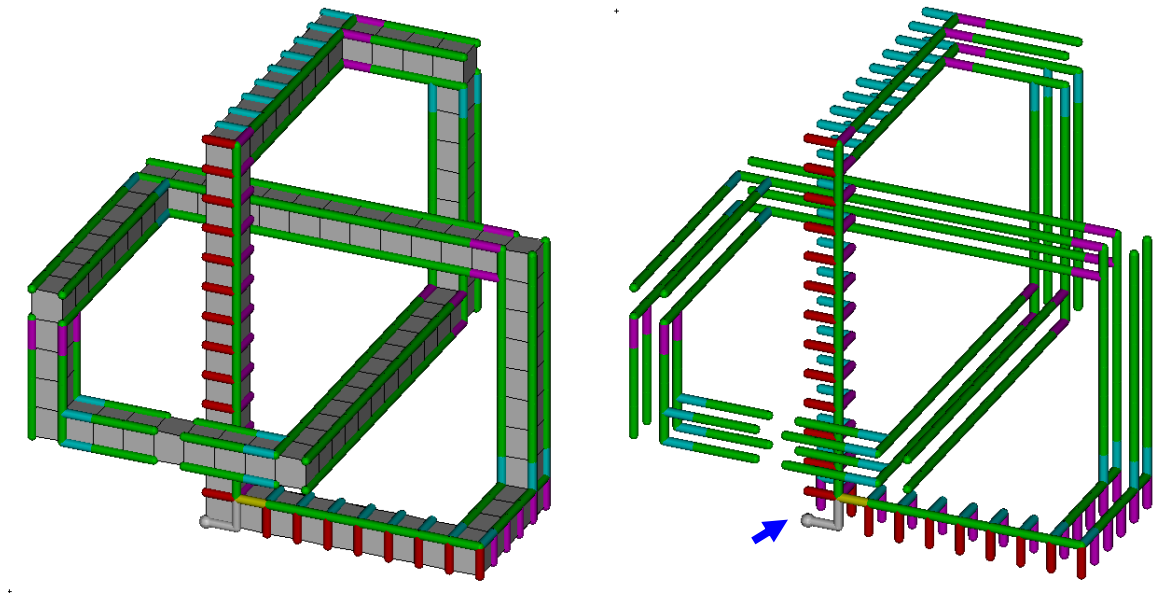


Fig.7.22. The enclosing tree of the trefoil knot. It represents the knot of Fig(b). The tree as a string of characters appears in the text (Section 7.0). An arrow shows the "handle"

7.5 DISCUSSIONS AND CONCLUSIONS

7.5.1 DISCUSSIONS

Some remarks are in order about uses and extensions to enclosing trees.

7.5.1.1 MODIFICATIONS FOR SHAPE COMPARISON

In order to compare the *shape* of two objects *O* and *P* using their respective enclosing trees, some additional normalization is necessary.

A) Alignment of axes. The enclosing tree of an object changes if we rotate the axes. To achieve invariance, the axes of *O* and *P* must be precisely aligned, for instance re-sorting to the major axis of the object. How to achieve this appears in [4, 9, 11].

B) Size of voxels. The enclosing tree of an object changes as we change pixel size. To achieve invariance, a unique pixel size should be selected for each object, for instance by enclosing the object in the smallest box (rectangular prism) that just encloses the object. The sizes of the axes of that box are defined to be of size 100, 100 and 100, or other suitable number. The enclosure will contain one million pixels, the object will occupy less. The size of the voxels should be small enough so as to capture all desired protuberances, dips and other nuances, while keeping a reasonable number of object voxels. See [4] for this technique applied to 2-D objects.

C) Unique origin for the enclosing tree. The enclosing tree of an object changes if we use a different starting point. To achieve invariance, the starting point must be uniquely identified in the object. If possible, the starting point should also be resistant to small changes in the surface of the object. The center of mass, a good candidate for 3-D objects, is generally not available for this purpose (it is usually not on the surface). We propose as starting point of the enclosing tree the point in the surface closest to the center of mass.
D) Unique starting growth plane (Section 7.7.3.5). The enclosing tree changes if we select another growth plane to start growing. To fix this, the X axis should be selected along the longest side of the box in (B), the Y axis should be the next longest side of the box, and the Z axis should be selected so that $Z = X \times Y$.

7.5.1.2 SYMMETRIES

Under rotation, the enclosing tree remains unchanged. Under reflection, the enclosing tree changes its 1's to 3's and its 3's to 1's, as already discussed in Section 7.7.3.1. The tree symmetric with respect to the origin to another tree is also obtained by changing 1's to 3's and 3's to 1's, since symmetry through the origin is equivalent to three mirror reflections.

7.5.2 CONCLUSIONS

The enclosing tree provides a compact representation of voxelized objects. It is a chain of base-5 digits, where lines are sequences of those digits and forks are represented by parenthesized lines. An enclosing tree covers (touches) every vertex of the enclosed surface; so reconstruction is achieved without loss of information. The enclosing tree preserves the *shape* of voxel-based objects, allowing us to know their topological and geometric properties. The enclosing tree is invariant under rotation and translation, and the transformation it suffers under reflection is straightforward to compute. The enclosing tree is a good tool for storing voxelized objects.

ACKNOWLEDGMENTS

DEM data was provided by INEGI. LAM acknowledges support from DGAPA-UNAM. This work was partially supported by CONACYT, IIMAS-UNAM and SIP-IPN.

References

1. Ballard, D. H. and Brown, M.C. *Computer Vision,* Prentice-Hall, Englewood Cliffs, New Jersey (1982).
2. P. Besl and R. Jain. Three-dimensional object recognition, *ACM Comput. Surv*. **17** (1), 1985, 75-145.

3. J. W. Boyse,. Data structure for a solid modeller, NSF *Workshop on the Representation of Three-Dimensional Object*s, Univ. Pennsylvania, 1979.
4. Bribiesca, E. and Guzman A. How to Describe Pure Forms and how to Measure Differences in Shapes using Shape Numbers. (1980) Invited paper to the *IEEE Conference on Pattern Recognition and Image Processing.* Chicago, USA. Also in *Pattern Recognition,* Vol **12**, No. 2, 101-112
5. Bribiesca E. Digital Elevation Model Data Analysis Using the Contact Surface Area, *Graphical Models and Image Processing*. Vol. **60**, No. 2, pp. 166-172 (1998).
6. Bribiesca, E. and Velarde, C. A formal language approach for a 3D curve representation, *Computers & Mathematics with Applications* **42**, 1571-1584 (2001).
7. Bribiesca, E. A method for representing 3D tree objects using chain coding, *Journal of Visual Communication and Image Representation* **19**, 184-198 (2008).
8. Brooks, R. Model-based 3-D interpretations of 2-D images, IEEE *Trans. Pattern Anal. Mach. Intelligence* **5** (2), 1983, pp. 140-150.
9. Bullow, H., Doodley, L, and Wermser, D. Application of principal axes for registration of NMR image sequences, *Pattern Recognition Letters* **21**, 329-336 (2000).
10. Freeman, H. On the encoding of arbitrary geometric configurations. *IRE Trans. EC* **10**, 2, 260-268, 1961.
11. Galvez, J.M. and Canton, M. Normalization and shape recognition of three-dimensional objects by 3D moments, *Pattern Recognition* **26**, 667-681 (1993).
12. Gomez, D. and Guzman, A. A digital Model for Three-dimensional Surface Representation. (1979) *Journal of Geoprocessing* **1**, 53-70. Elsevier Publishing Co.
13. Gonzalez, R.C. and Woods, R.E. *Digital Image Processing*, Second Edition, Prentice Hall, Upper Saddle River, New Jersey 07458 (2002).
14. Guzman A. *Canonical Shape description for 3-D Stick Bodies*. (1987) Technical Report ACA-254-87, Microelectronics and Computer Co.
15. Kikuchi, L., Guevara, J.A., Mark, D. and Marble, E.F. Rapid display of digital elevation models in a mini-computer environment, in *Proceedings of ISPRS Commission IV, Environmental Assessment and Resource Management*, Crystal City, Virginia, U.S.A., 1982, pp. 297-307.
16. Pentland, A. Perceptual organization and the representation of natural form, *Artificial Intelligence* **28**, 1986, 293-331.
17. Sagols, F. Hamiltonian representation of vox-solids, *Computación y Sistemas* **4**, 213-217 (1999).
18. Voelcker, H.B. and Requicha, A.A.G. Geometric modeling of mechanical parts and processes, *Computer* **10**, 1977, 48-57.

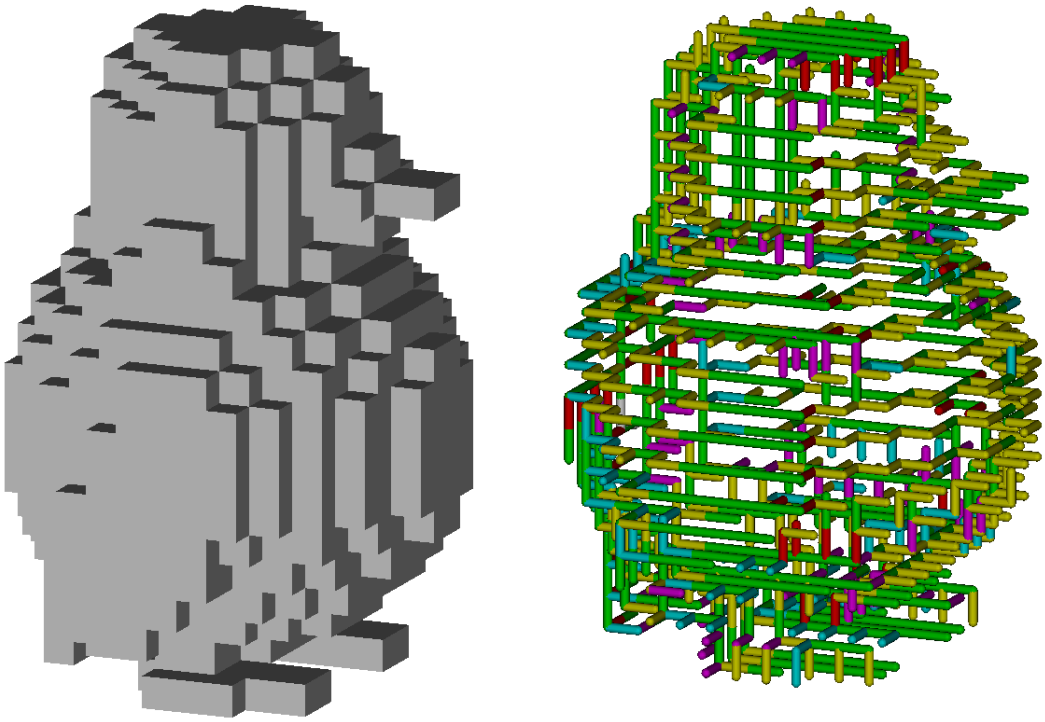


Fig.7.1. A voxelized object (a penguin) and its enclosing tree