Conversion and Integration of Boundary Representations with Octrees*

T. K. Chan, I. Gargantini
Department of Computer Science,
The University of Western Ontario,
London, Ontario Canada N6A 5B7
and T.R.S. Walsh
Département de Mathématiques et d'Informatique,
Université du Québec à Montréal,
Montreal, Quebec H3C 3P8
e.mail: irene@uwocsd.uwo.ca

Abstract

This paper addresses the problems of converting polygon-based objects to octree-form and integrating 3D objects expressed in Volume Representation with others expressed in Boundary Representation. To achieve this the authors propose an approach which makes use of: (i) filling by octants, (ii) connectivity labeling, (iii) digitization of straight lines and polygons in 3D. Selected examples are given: among these, one deals with volumes enclosed by slightly inaccurate boundaries, as those generated, for instance, by the use of digitizing tablets; another deals with joining distinct volume-based objects with triangular digitized tiles; another again simulates bone grafting.

Keywords: octrees, 3D modelling, volume representation, boundary representation, 3D filling, polygon-to-octree conversion, digitization

1 Introduction

A variety of methods is available to create an abstract representation of a 3D object: among these we distinguish the Constructive Solid Geometry approach, the Boundary Representation technique and the Volume Representation method [1]. Very few attempts have been made to convert a Boundary Representation *directly* into a hierarchical block structure: the first successful attempts appeared in [2, 3] and are based on techniques to separate maximally-connected subsets. Another related approach is the algorithm used to fill a digitized boundary by octants [4], which starts from a set of Boundary volume-elements and proceeds by aggregating to these their internal cubic neighbours of increasing size.

Converting Boundary-based objects to octree-form is an issue arising in the writing of user interfaces: users, in fact, prefer to enter objects' description via vertices of polygons or control points of splines even when using packages and/or procedures based on octrees; integrating 3D objects expressed in Volume Representation with others expressed in Boundary form is also an important issue because it opens the possibility of incorporating pictorial data (previously created or acquired) into newly defined surfaces in order to create more complex objects. An example is given in Fig. 1 where two cones, created in Volume Representation, are joined by digitized triangular tiles entered in Boundary Representation. The four viewports show how the tiling progresses in a back-to-front display.

The work presented in this paper is part of a larger project dealing with the simulation of bone grafting and custom-designed prostheses using previously acquired objects. These objects, most of the time, are given in Volume Representation - arising, for instance, from reconstructed X-ray Computerized Tomography, Magnetic Resonance Imaging, or Positron Emission Tomography scans. The above-mentioned simulation requires the ability to handle local modifications of the surface of a given solid, followed by the filling of the small gaps created between the locally-created surfaces and the existing volume-based solid, thus generating an *integrated* object with uniform representation.

To reach our goals we have designed and implemented some graphics tools: the first deals with the conversion of polygonal surfaces in 3 dimensions into *octrees*, the second with 3-dimensional filling of the digitized surfaces; the third with the filling of the above-mentioned gaps. This paper describes in detail the first two topics, i.e. *conversion and filling* while only some preliminary results on the third are given.

The organization of the sections is as follows: Section 2 gives an overview of octrees, Section 3 introduces some relevant definitions, Section 4 addresses the problems of digitization, Section 5 presents the new algorithm, Section 6 outlines the testing of the algorithm and Section 7 gives the conclusions.

2 Octrees

A 3D raster is a $(2^n \times 2^n \times 2^n)$-array of volume elements called *voxels* while n is called the *resolution*. A 3D *binary picture* is a 3D raster in which each voxel is either *black* or *white*: the set of black voxels represents the *object*, that of white ones the *background*. If the picture has large blocks of black voxels, it can be represented effectively by an *octree*: a rooted tree, each of whose nodes is a leaf or has 8 sons [5,

*The authors gratefully acknowledge the financial support of the Canadian Government through NSERC grants.

6, 7, 8, 9, 10, 11]. The octree for the picture is generated by recursively grouping the raster's voxels into ($2^m \times 2^m \times 2^m$)-cubes (also called *octants* of order m), with m= 1,2,...,n-1. The root represents the whole raster. If the entire picture is black (white), then the root is a black (white) leaf; otherwise it has 8 sons, each representing an octant of dimensions ($2^{n-1} \times 2^{n-1} \times 2^{n-1}$) and is said to have gray or black-and-white color. This process is applied recursively to each of the 8 sons, and terminates after at most n levels of decomposition.

octant	octal digit	quaternary digit	quadrant
front-north-west	0	0	north-west
front-north-east	1	1	north-east
front-south-west	2	2	south-west
front-south-east	3	3	south-east
back-north-west	4		
back-north-east	5		
back-south-west	6		
back-south-east	7		

Table 1: Labelling octants or quadrants

A *linear octree* [12, 13, 14, 15, 16] is a compressed form of the regular octree described above: we store only the black leaves and label them by a code described below. Octants can be labeled in different ways; in this paper we have adopted the convention shown in Table 1. The code for a leaf representing a ($2^m \times 2^m \times 2^m$)-cube consists of the n-m octal digits representing the recursive subdivision of the raster into octants (labeled as in Table 1) ending in that cube, followed by m copies of the symbol 'X'. The linear octree of an object is formally defined as the list of the octal codes sorted lexicographically in ascending order. For the object shown in Fig. 2, the linear octree is <01,1X,35,51>; the corresponding regular octree is shown in Fig. 3. In a more space-efficient version of linear octrees' representation the (n-m)-digits are padded on the right with m zeros, and followed by the grouping factor m.

For the 2D case, the raster becomes a ($2^n \times 2^n$) square, an octant of dimensions ($2^m \times 2^m \times 2^m$) becomes a quadrant of dimensions ($2^m \times 2^m$), m=1,2,...,n-1, a voxel a pixel, the 8 sons become 4 sons: each node is represented by a string of n quaternary digits, encoded as shown in the first 4 rows of Table 1.

3 Definitions and Data Structures

To represent a voxel Q in space we adopt two notations: the usual matrix notation, with the triplet (j,i,k) identifying Q (with j increasing from west to east, i increasing from north to south, and k increasing from front to back), and the octal labeling given in Section 2. The reason for this double notation is that the algorithms are formulated in an octree environment, while the user will expect to specify the input data in the first representation.

Rasterborder is the set of voxels forming the border of the raster, i.e. the set of voxels with at least one coordinate being 0 or (2^n-1). Two voxels Q=(j,i,k) and Q'=(j',i',k') are *O(p)-adjacent* for p=0,1,2,3 iff

$$0 \leq | i - i' | \leq 1; \; 0 \leq | j - j' | \leq 1, \; 0 \leq | k - k' | \leq 1,$$
$$(| i - i' | + | j - j' | + | k - k' |) \leq p$$

In this case, Q and Q' are said to be *O(p)-neighbours*; the term *neighbours* means O(1)-neighbours.

An *O(p)-path*, p=1,2,3, from voxel Q to voxel Q' is a sequence of voxels $Q=Q_0,\dots,Q_m=Q'$, m≥0, such that for s=1,...,m, Q_{s-1} and Q_s are O(p)-adjacent. Two voxels Q and Q' in a set S are said to be *O(p)-connected* in S, p=0,1,2,3, if there exists an O(p)-path from Q to Q' consisting entirely of voxels in S. This binary relation is an equivalence relation on S, dividing S into equivalence classes called *O(p)-components.* If S has only one O(p)-component then it is called *O(p)-connected.* A region is an O(1)-component of the black voxels. The *Border* or *Boundary* of a binary picture is the subset of black voxels in which each element is either a Rasterborder voxel or is O(1)-adjacent to a white voxel.

We recall that two voxels Q and Q' are referred to as neighbours iff they are O(1)-adjacent. Two octants (or nodes) V and V' are called neighbours if there is at least one voxel in V and one voxel in V' which are O(1)-adjacent; the number of such voxels in each of V,V' is then 4^m, where m= min (order of V, order of V'). A *Rasterborder node* or octant is a node which contains at least one Border voxel.

As the computer representation for a linear octree we use, most of the time, a singly-linked list with each node represented by 4 fields, one for the octal code, one for the grouping factor, one for the adjacency relations (called Block) and one for the pointer to the next node. "Block" itself is a string of 6 bits, called block-bits, one for each of the <east,south,back,west,north,front> directions, here denoted by <E,S,B,W,N,F>.

We set the i^{th} Block-bit, i ∈<E,S,B,W,N,F>, of node Q to 0 and say that Q is unblocked in direction i if we know that each of Q's voxels in its direction-i face is black and has a black direction-i neighbour. If we know that the contrary is true - that is, one of these voxels is white or has a white direction-i neighbour or no direction-i neighbour at all (this face is part of the Rasterborder) - then we set the bit to 1 and say that Q is blocked in direction i. Assume that the color of Q is known, that Q's direction-i face is not on the Rasterborder, and that the color of its neighbour P of the same order s as Q is known (if Q has a black neighbour of order s, then P can be taken as the one order-s subnode adjacent to Q). Let v_j and w_j be the number of order-j black subnodes of Q and P, respectively, which share Q's direction-i face. Since each order-j node contributes 4^j black voxels to the face and 4^s of them are needed on either side, the condition for Q to be unblocked in direction i is that

$$\textstyle\sum_{j=0}^{s} v_j 4^j = \sum_{j=0}^{s} w_j 4^j = 4^s \qquad (1)$$

In 2 dimensions a voxel becomes a pixel, an octant a quadrant, the concepts of O(p)-adjacency and O(p)-connectivity are defined only for p=0,1,2, and in relation (1), 4 is replaced by 2.

4 Digitization

4.1 The two-dimensional case

In this section we describe how to convert polygon's edges into pixels while determining the adjacency relations with respect to the background. The latter is, of course, derived from the direction in which the polygon's vertices are specified: counterclock-wise for external contours, clock-wise for internal. We now adapt the Digital Differential Analyser [17] to express pixels directly in their quaternary codes together with the corresponding Block-bits. Let Q be the starting-point and Q' the end-point of an edge. If Q' is to the north and to the west of Q, then every pixel in the edge has its <E,S,W,N> Block-bits set to < 1,0,0,1>. If Q' is due west of Q, then every pixel in the edge except Q and Q' has its Block-bits set to <0,0,0,1>; Q has its Block-bits set to <2,0,0,1> and Q' has its Block-bits set to <0,0,2,1> (2 means "don't know yet"). For different directions of Q' with respect to Q these rules are rotated by multiples of 90^o and, for clockwise traversal, by an additional 180^o.

Note that two O(1)-adjacent black pixels may be blocked in each other's direction, but mutual adjacency of black pixels is immaterial, since it does not play any role in any of the subsequent filling stages.

Each edge, together with the Block-bits of all its pixels, is digitized individually. The endpoints of these edges will be represented twice, often with conflicting Block-bits; other pixels may also occur several times because of very acute angles, a highly concave polygon, or polygons approaching each other very closely. The following conflict-resolution scheme is used. If among the i^{th} Block-bit of all copies of the same pixel there is at least one 0 and at least one 1 then this bit is changed to 2 for every copy; otherwise all the 2's are changed to conform with the 0 or 1 bits, if such exist.

For Fig. 4 the resulting digitization is given in Fig. 5, where an arrow indicates a blocked pixel (Block-bit = 1), a dark dot an unknown condition (Block-bit = 2) and no symbol indicates an unblocked condition (Block-bit = 0).

4.2 The three-dimensional case

In 3D the digitization deals with 2 cases: isolated straight-line segments and convex polygons. The first case (digitization of an edge in 3D) is similar to its 2D counterpart, discussed previously. We explain it by an example. Without loss of generality, assume that the given straight line (here denoted QQ') starts at Q, points toward the south-east-front direction, and terminates at Q', and its i^{th} coordinate changes faster than either its j^{th} or k^{th} coordinate. Point Q is digitized into a voxel (also denoted by Q) and its southern O(1)-neighbour (say Qc) is created as a candidate for the next voxel. If the line (with endpoints shifted to the middle of voxels Q and P) crosses the (extended) border between Qc and its eastern neighbour at a point more northerly than the middle of Qc then Qc is changed into its eastern O(1)-neighbour (which is renamed Qc); otherwise it is left unchanged. If the line crosses the (extended) border between Qc and its front neighbour at a point more northerly than the middle of Qc then Qc is changed to its front O(1)-neighbour; otherwise it is left unchanged. In all cases the next voxel approximating the given line in 3D space is O(3)-connected to the previous one. As an example, if Q=(2,2,6) and Q'=(8,12,3), then the voxels in the line are (2,2,6),(3,3,6),(3,4,5),(4,5,5),(4,6,5),(5,7,5),(6,8,4), (6,9,4),(7,10,4),(7,11,3),(8,12,3).

The second case is the digitization of a convex polygon which is supposed to form part of the Boundary of a 3D object. The edge digitization is followed by a filling by quadrants [4] and a backprojection in order to produce the output directly into octree form.

This approach uses well-known elementary techniques [1, 17]. Let Vert$(l), l$ =1,...m, m$\geq$3, be the vertices of the polygon to be digitized, given in counterclockwise (clockwise) direction for external (internal) contours. Assume that Vert(1), Vert(2) and Vert(3) are distinct. Let $\overline{N}$= (A,B,C) be the outward normal to the plane F(j,i,k) on which the polygon lies, evaluated as $\overline{N} = (\overline{2}-\overline{1}) \times (\overline{3}-\overline{2})$ where $\overline{m}$,m=1,2,3, is the vector representation of Vert(m), " $\times$ " denotes the cross-product. F(j,i,k) can then be expressed as

$$F(j,i,k) = Aj + Bi + Ck + D = 0 \qquad (2)$$

where D is a constant.

We now select a projection plane among the three principal ones according to the maximum of $<\mid A \mid,\mid B \mid,\mid C \mid>$. If such a maximum is |A|,|B|, or |C|, we select the (ik),(jk),or (ji)-plane respectively. Without loss of generality, let |C| be such a maximum, so the (ji) is the chosen projection plane.

In order to express the area enclosed by the vertices in terms of quaternary codes, we proceed by digitizing the edges and evaluating their adjacency information followed by filling quadrants [4]. We then solve (2) with respect to k , i.e. k = -(Aj + Bi + D)/C, to find the number of units that a pixel should be backprojected. To obtain a linear octree representation an inner merge operation must be incorporated into the backprojection. In our implementation, backprojection and filling are carried out simultaneously in one pass through the Border elements.

5 Double Connectivity Filling

Two existing procedures, namely Filling by Octants [4] and Connectivity Labeling [2, 3], are used to convert a Boundary Representation into hierarchical structures such as binary trees or octrees. Given a 3D border, Filling by Octants proceeds from the border toward the interior of a region by enclosing nodes of increasingly larger size: the distinction between internal and external nodes is made possible by maintaining adjacency information during the filling process. Given the initial adjacency information for the border elements, this procedure fills simply-connected, multiply-connected and nested regions and even a set of these in one pass through the input data. Connectivity Labeling, on the other hand, applies the principle of separating maximally-connected subsets: the interior of a set of 3D regions is determined by letting a part of the rasterborder (which is, of course, external) expand toward each region, while suitably updating that part of the rasterborder by means of active lists of visited nodes. The result is the encapsulation of the regions' interior by the surrounding background. The corresponding algorithm fills objects without holes.

The input to the above algorithms is the Border: the latter is the subset of the black voxels in which each element is either a Rasterborder voxel or is O(1)-adjacent to a white voxel. Now a problem arises: how can we guarantee that Border is determined exactly or, alternatively, how can we guarantee that its voxels' adjacency with respect to the background is determined correctly? In most practical cases this *cannot* be guaranteed. In fact, some (hopefully few) adjacency relations may not be determined or may be determined incorrectly due to the ambiguities introduced by sampling techniques incorporated into input devices or by the digitization itself : an example of the latter occurrence is illustrated in Fig. 4, where for pixels (i,j) = (5,15) and (i,j) = (8,12) the adjacency with the background cannot be established.

This is the motivation behind the introduction of our heuristic approach, called *Double Connectivity Filling*: this technique combines the principles of the two previously mentioned methods, while being capable of resolving most of the Border's ambiguities. When pixels' or voxels' adjacencies (with respect to the background) cannot be ascertained, different cases arise: sometimes the corresponding uncertainty is of no consequence; sometimes it can be resolved by looking at the neighbouring leaves; however, if neither previous case occurs, the undetermined adjacency is flagged as such and sifted out one level up in the tree, where a similar procedure is applied. At each level the criteria contained in Filling by Octants and in Connectivity Labeling are applied to determine the larger octant's adjacencies and to decide if neighbouring nodes can be aggregated to the presently active objects, to the background, or if they form the beginning of a new object. At the root level either a majority vote (based on adjacency relations) can be taken or regions of uncertain classification can be simply listed as "separate" or "undecided" components. The authors [18] have adopted the second alternative, since the absence of these components or their small size is an indication of the success of the algorithm.

Another way to intuitively describe our approach is by the following example. Consider a polyhedron located in the raster, without touching Rasterborder, and such that each face carries the value of its inner normal - defined by the ordered traversal of the face's vertices - pointing toward the interior of the polyhedron. When Double Connectivity Filling starts, the polyhedron's interior is determined by moving inwards - from all faces simultaneously - while propagating the colour *black*. Cuncurrently a *seed-vector*, determined by the voxels on the Rasterborder, starts a flood-fill which eventually surrounds the polyhedron with *white* nodes. In the digitized version of this example, we deal with voxel-based faces whose normal information is encoded into the corresponding Block-bits: the digitization process may introduce inaccuracies, the most common of which is represented by a missing voxel or by voxels with conflicting adjacency information in the neighbourhood of a vertex. Because Double Connectivity Filling fills the background by propagating the *white* colour and the polyhedron's interior by propagating the *black* colour *simultaneously*, nodes whose colour cannot be determined are, in general, restricted to a small set of voxels along the polyhedron boundary. What to do with nodes can be decided after the algorithm terminates, as indicated previously.

6 Experiments

A variety of tests have been performed both in 2D and 3D: of these, some have been planned to deal with common situations (such as vertices well separated in terms of number of pixels or voxels), others with more unusual configurations. The first experiment related to the 2D case of Figs. 4 and 5. Filling by quadrants left one undetermined component (formed of 2 pixels) while Connectivity Labelling could not handle the inside region. Double Connectivity Filling created the region of Fig. 6 out of the contour given in Fig. 5.

The object shown in the North-West quadrant of Fig. 7 has been created out of 30 vertices, the one in the North-East quadrant out of 54 vertices and that in the South-East out of 40 vertices. The cube, in the South-West quadrant, had cavities incorporated, as shown in Fig. 8, for a total of 27 vertices. In Table 2 some data related to the creation of these objects is given. All objects were created in a time between 18" and 48" on a PDP-10 machine. The display was obtained on a Barco 5151 graphics monitor via an in-house package, LINOCT [15, 19], which allows, among others, choice of resolution, selection of light-source location, and the capability of setting the view-point inside the object, in order to simulate an interior cut (see Fig. 8).

Each of these objects has been created with n=6 and displayed with one or two selected light sources: a simple illumination model was used to give some 3D perception without destroying the visibility of the octrees' structure. For these configurations all missing or conflicting adjacencies were resolved while building the final object's octree; no undecided (separate) components were generated, as expected.

Two other experiments deserve mention: one was designed to test how the method worked when vertices defining external surfaces were close to vertices defining internal surfaces: Fig. 9 illustrates this case, where n=8 and the vertices are those of two tetrahedrons, one inside the other. This object was created in 5 minutes, on a 68010-minicomputer (MASSCOMP-500) with 42,207 Border voxels, 243,806 object voxels and 34,485 nodes. In this case too there were no undecided components.

The other experiment was designed to test how integrating Volume Representations and Boundary Representations would work in order to generate a new, more complex object directly in octree form. Two cones were generated with n=7 (31,735 nodes and 426,143 voxels) and stored as a picture file. A set of 20 points along the two cones' cross-sections were calculated and triangular patches entered to bridge the gap between the two previously created cones. Fig. 1 shows the two cones and the tiling surface displayed in a back-to-front fashion. This object was created in 8 minutes (MASSCOMP-500) with a total of 14,821 tiling (green) nodes.

The final experiment, illustrated in Fig. 10, deals with the graphics simulation of grafted bone tissues. A medical object acquired with 99 Computerized Tomography scans produced the object shown on the left, with n=9, 13,778 nodes and 62,785 voxels. In the experiment a segment, roughly equivalent to one eight of the total volume, was removed and graphically reconstructed via lateral triangular tiles. The gap encapsulated between the original solid object and the user's defined patches was filled back: the picture on the right shows the *integrated* object, with the red part showing the reconstructed segment.

object in quadrant	No. of Border voxels	No. of object nodes	No. of object voxels
0	8,114	5,102	26,648
1	6,114	3,214	12,790
2	21,934	11,814	52,113
3	10,960	6,944	20,930

Table 2: Data for pictures of Fig. 7

7 Conclusions

The paper reports on the development of graphics tools to convert a Boundary Representation into octrees and to integrate previously created or acquired objects in volumetric form with objects partially defined in Boundary Representation. To achieve this, several problems have been identified, some solved. The two major problems encountered deal with: (i) finite - sometime inadequate - numerical accuracy with which points in space are entered via interactive devices; (ii) lack of full-proven algorithms to determine if a solid is *indeed* a bounded, closed subset of the three-dimensional Euclidean space. To which extent and in which cases can we then answer the question: how much numerical inaccuracy (introduced by a finite representation of real numbers) can we incorporate into the border definition, and *still* be capable of identifying an interior?

In the absence of theoretical results in this field, the authors have tackled these problems from a heuristic point of view, and addressed them in a restricted environment, i.e. (i) in the case which local - hopefully small - boundary modifications are required, and (ii) in the case the 3D object is in octree form. The heuristic described in this paper deals with conversion of boundary representation into octrees; filling digitized surfaces in space with blobs of successively increasing size; counteracting the presence of numerical inaccuracies in user-defined vertices by combining the previous filling with a labelling component technique. Several examples illustrate the approach proposed by the authors.

Acknowledgement

The authors wish to thank Mr. H.H. Atkinson for the creation of some of the pictures.

References

[1] MORTENSON, M.E., Geometric Modeling. J.Wiley, New York, 1985.

[2] TAMMINEN, M., and SAMET, H., Efficient octree conversion by connectivity labelling. ACM Computer Graphics, Vol. 18, No.3 (July 1984), New York, 43-51.

[3] SAMET, H. and TAMMINEN, M., Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintrees. IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 10, No. 4 (July 1988), 579-586.

[4] ATKINSON, H.H., GARGANTINI. I., and WALSH, T.R.S., Filling by quadrants or octants. Comput. Vision Graph. Image Proc. 33 (1986), 138-155.

[5] JACKINS, C.L., and TANIMOTO, S.L., Oct-trees and their use in representing three-dimensional objects. Comput. Graph. Image Proc. 14 (1980), 249-270.

[6] MEAGHER, D., Geometric modeling using octree encoding. Comput. Graph. Image Proc. 19, 2 (June 1982), 129-147.

[7] HUNTER, G.M., and STEIGLITZ, K., Operations on images using quadtrees. IEEE Trans. Pattern Analysis and Machine Intelligence, 1, 2 (April 1979), 145-153.

[8] SAMET, H., Region representation: quadtrees from boundary codes. CACM 23, 3 (March 1980), 163-170.

[9] SAMET, H. and WEBBER, R.E. Hierarchical Data Structures and Algorithms for Computer Graphics, Part I, Computer Graphics and Appls., Vol. 8 , No. 3 (1988), 48-68.

[10] SAMET, H. and WEBBER, R.E. Hierarchical Data Structures and Algorithms for Computer Graphics, Part II, Computer Graphics and Appls., Vol. 8 , No. 4 (1988), 59-75.

[11] SHAFFER, C.A. and SAMET, H., Optimal quadtree construction algorithm. Comput. Vision, Graphics, and Image Proc. 37, 3 (March 1987), 402-419.

[12] GARGANTINI, I., Linear octrees for fast processing of three-dimensional object. Comput. Graph. Image Proc. 20 (1982), 365-374.

[13] ABEL, D.J., A B^+-tree structure for large quadtrees, Comput. Vision, Graphics, and Image Proc. 27, 11 (July 1984), 19-31.

[14] WALSH, T.R.S., On the size of quadtrees generalized to d-dimensional binary pictures. Math. Comput. Appl. 11 (1985), 1089-1097.

[15] GARGANTINI, I., WALSH, T.R.S., and WU, O.L., Viewing transformations for voxel-based regions via linear octrees. IEEE Comput. Graph. Image Proc. 14 (1986), 249-270.

[16] ATKINSON, H.H., GARGANTINI, I., and WALSH, T.R.S., Counting regions, holes, and their level of nesting in time proportional to the border. Comput. Vision Graph, Image Proc. 29 (1985), 196-214.

[17] FOLEY, J.D., and VAN DAM, A., Fundamentals of Interactive Computer Graphics. Addison-Wesley, Reading, MA, 1982.

[18] CHAN, T.K., GARGANTINI, I., and WALSH, T.R.S., Double connectivity filling for 3D modeling. Tech. Report #155, 1986, Department of Computer Science, University of Western Ontario, London, Ont. N6A 5B7, Canada.

[19] ATKINSON, H.H., GARGANTINI, I., and WU, O.L., LINOCT2.0. Tech. Report #176 (1987), Department of Computer Science, University of Western Ontario, London, Ont. N6A 5B7, Canada.

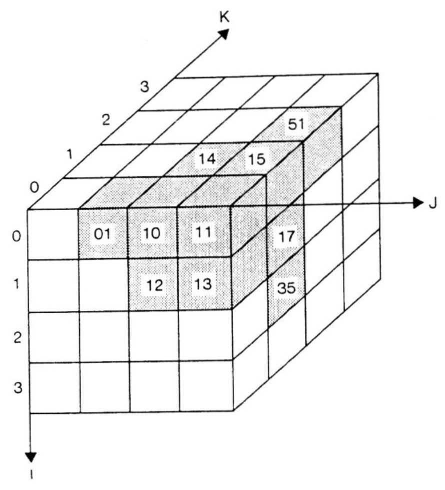

Fig. 2. Representation of an object with n=2

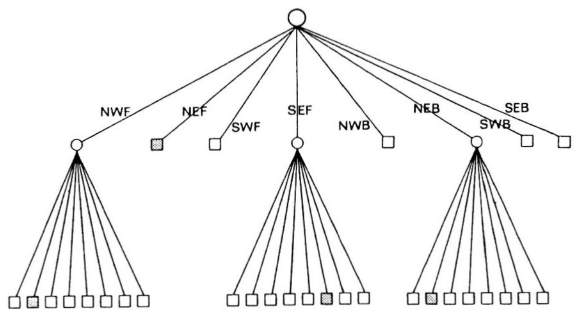

Fig. 3. Octree for the object of Fig. 2

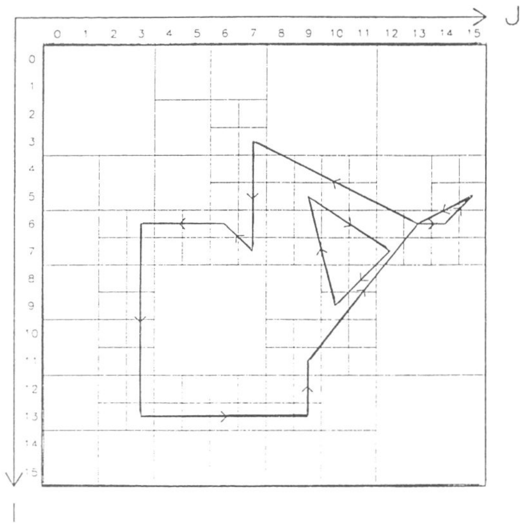

Fig. 4. A 2D Border to be digitized with n=4

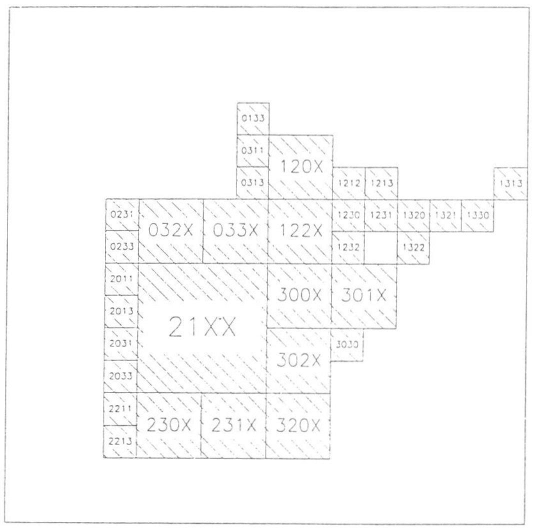

Fig. 6. The created region for the Border of Fig. 4

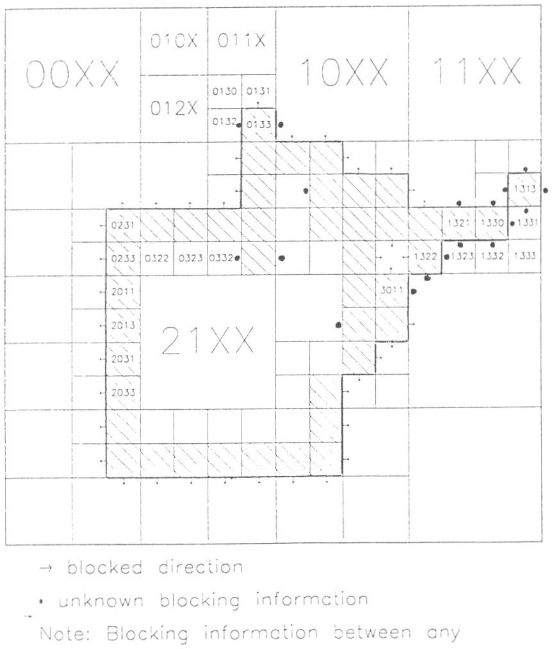

Fig. 5. The Border pixels for Fig. 4

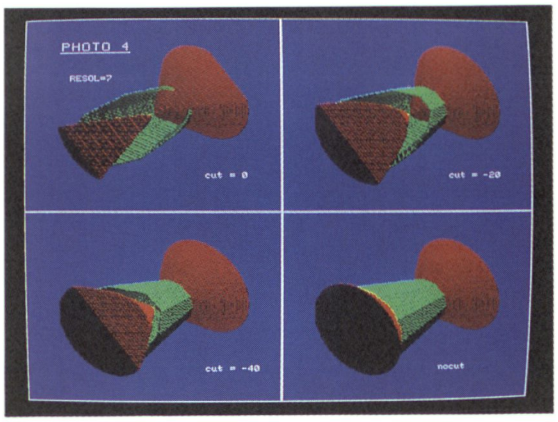

Fig. 1. Two solid cones tiled with digitized triangular patches

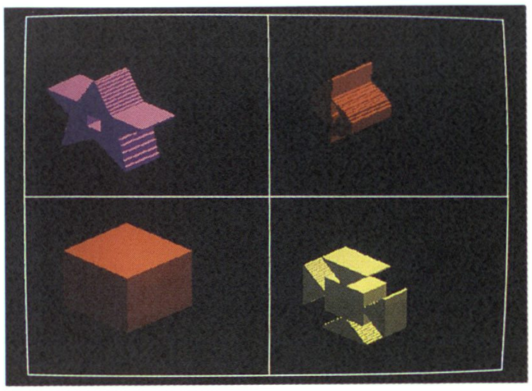

Fig. 7. Objects created by Double Connectivity Filling

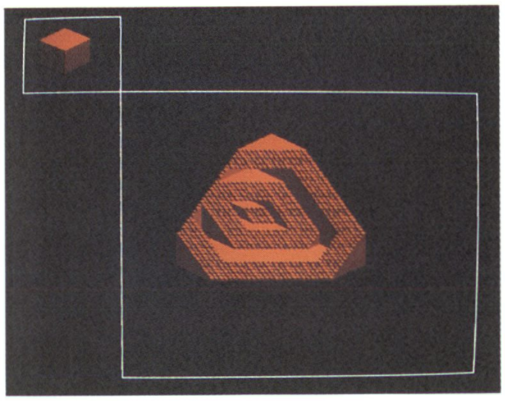

Fig. 8. Internal view of the cube of Fig. 8

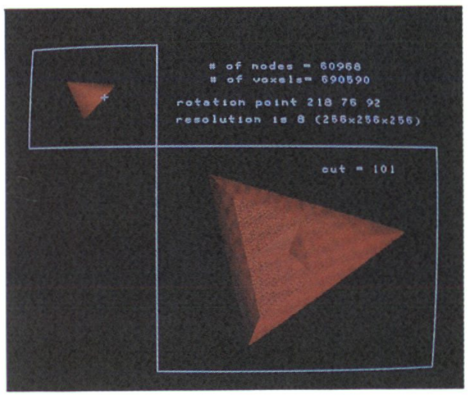

Fig. 9. Two tetrahedrons, one inside the other

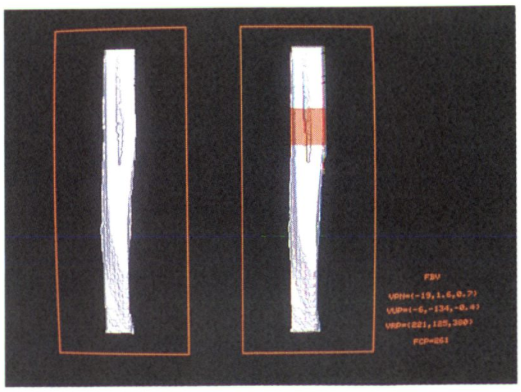

Fig. 10. Simulation of grafted bone tissue