An improved boundary extraction algorithm using a plane-sweep technique

Dolors Ayala and Enric Vergara

Universitat Politècnica de Catalunya
(dolorsa,evergara)@lsi.upc.edu

ABSTRACT

Extracting the boundary of a 3D or a 2D image is a fundamental operation in several image processing related fields. In other fields as NC-data generation, obtaining the cutting areas of sculptured surfaces can be performed by first representing the surface using a regular grid model which is equivalent to a 2D binary image. There exist several approaches to extract the boundary of a 3D image. Most of them represent the extracted boundary as a collection of a large number of little triangular or quadrangular faces whereas few approaches give general orthogonal contours with the corresponding inclusion relationships. One of these approaches is based on a secondary model EVM and allows to obtain the orientation of the output primitives (edges and faces). It actually obtains, for each plane of the resulting B-Rep model, a set of oriented edges that have to be rearranged as contours and these contours have to be classified in order to have the corresponding inclusion relationships. These last two processes are performed in a simple brute force way. In this paper, we present an improved algorithm that processes all the edges of a plane and, following a plane-sweep based method, obtains the contours and the inclusion relationships. The presented algorithm can also be applied to extract the boundary of a 2D image.

Keywords: images and volume datasets, cutting areas, boundary extraction, inclusion relationships, plane-sweep techniques

1. INTRODUCTION

Boundary extraction of a 3D image (or otherwise said isosurface extraction of a volume dataset) is approached by two main methods, polygonal and digital. Polygonal (or beveled-form) methods represent the surface as a set of polygons (mostly triangular) [7]. Digital (or block-form) methods represent the surface as a set of voxels or a set of voxel boundary faces (surfels) [15]. These faces can also be general orthogonal faces bounded by four or more edges, not necessarily convex and with possible holes [2].

Digital models exhibit formal properties such as closure, orientedness and connectedness whereas polygonal models and related techniques are devoted mainly to visualization purposes and still lack of a solid modeling foundation [5].

A common drawback of the existing techniques is that the resulting isosurfaces consist of lots of little quadrangular or triangular faces. Several adaptive attempts have been developed to reduce this redundancy [10], [8] but most of them actually make a post-process and produce cracks.

In [9] the authors present an approach to compute the boundary of a quadtree (2D) and an octree (3D). They use a secondary PCS representation which, in the 2D case, is a list of stripes and compute the boundary as a list of polygon borders (2D) or a list of faces (3D). The 2D algorithm obtains oriented contours but it doesn't mention how obtains the inclusion relationships between them.

The problem of inclusion relationships between contours has been addressed by several authors using different denominations. Gargantini [3] uses the term of region containment tree for the same concept extended to 3D and presents a method to obtain it from an octree representation of a volume data. In [4] the authors compute the zonal graph and use it to perform several 2D image processes.

Park and Choi [11] present an algorithm that allows to extract the complete boundary of a 2D image. They work in the field of NC-data generation and the problem they solve is the extraction of cutting areas from a sculptured surface. In their approach, the sculptured surface is first discretized by sampling z-values of the surface into a regular grid (a Z-map) that can be easily transformed to a 2D binary image and, from it, they obtain a B-Rep of the cutting areas. They use a run-length codification of the 2D image and devise an algorithm which is $O(n)$, n being the number of runs, which computes all the contours and the inclusion relationships. Although they obtain faces and holes as general orthogonal contours, the vertical

edges of them are represented by little segments of pixel size which is a consequence of the run length codification used (see Fig. 1).

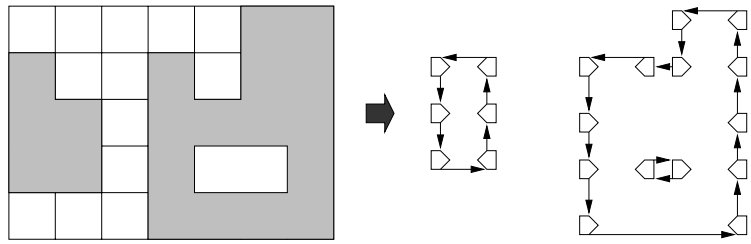


Fig. 1. Example of the method using run-length encoding (from [11]). Binary image (left) and contours obtained (right).

The *Extreme Vertices Model* (*EVM*) is a very concise model whose domain are orthogonal polyhedra and, therefore, it can represent binary 3D images or volumes. It can represent manifold, i.e. well-composed [6] as well as non-manifold topologies. The EVM is actually a complete (unambiguous) solid model [1] and any boundary representation can be obtained from it [2] with faces and holes as general orthogonal contours which are not restricted to any contour or edge size as are in other approaches. Actually, the EVM to B-Rep conversion algorithm obtains, for each plane, a set of oriented edges that have to be rearranged as contours and these contours have to be classified in order to have the corresponding inclusion relationships. These two last processes were performed in a simple brute force way in the first version of this algorithm.

In this paper, we present an algorithm which solves the 2D problem of obtaining the contours and inclusion relationships from a set of edges on a plane. This algorithm can substitute the mentioned two last processes of the previous algorithm and follows a plane-sweep based strategy which takes profit of the model EVM characteristics and which improves the previous approach. Space-sweep and plane-sweep techniques [12] are well-known and extensively used techniques in a lot of computational geometry problems as point location and convex hulls, among others.

The paper is arranged as follows. Next section introduces the EVM and explains the previous EVM to B-Rep conversion method. Section 3 explains the new algorithm and section 4 discusses the obtained results. Finally, section 5 concludes this paper and points out future work.

2. BOUNDARY EXTRACTION USING THE EVM REPRESENTATION

In this section, we include a short review of the EVM and the existing boundary extraction method, to make the paper more self-contained.

Let *P* be a 3D orthogonal polyhedron (OP). A *brink* is the maximal uninterrupted segment built out of a sequence of collinear and contiguous two-manifold edges of *P*. The ending vertices of a brink are called *extreme vertices* (*ev*). The *EVM* represents OP by its (and only its) set of *ev* and can recursively represent 2D and 1D orthogonal objects. See Fig. 2, left.

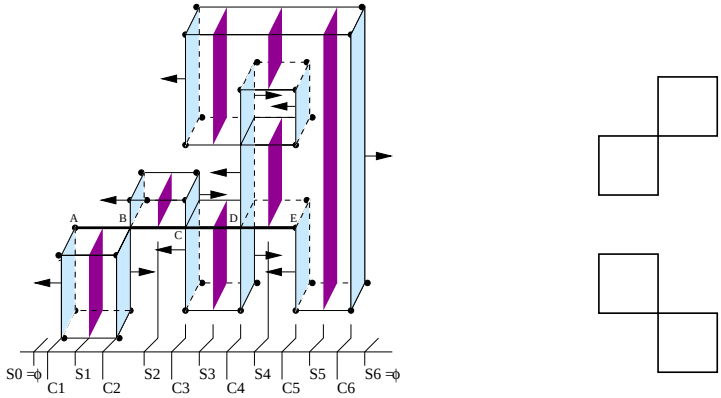


Fig. 2. (Left) An OP with a marked brink from vertex A to vertex E and with non-manifold edges and vertices. Cuts and sections for one direction are shadowed and *FD* and *BD* for each cut are remarked with arrows. (Right) 2D non-manifold configurations corresponding also to non-extreme vertices.

In 2D, the concept of non-extreme vertex corresponds with that of non-manifold and present the topology shown in Fig. 2 right (in higher dimensions this property doesn't apply).
A *cut* (*C*) is the set of vertices of *P* lying on a plane perpendicular to a main axis of *P*. A *slice* is the region between two consecutive cuts. A *section (S*) is the resulting polygon from the intersection between *P* and an orthogonal plane. Each slice has its representative section. Sections can be computed from cuts and vice-versa:

$$\overline{S_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{C_i(P)}, i = 1 \ldots n, \qquad \overline{S_0(P)} = \overline{S_n(P)} \tag{1}$$

$$\overline{C_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{S_i(P)}, i = 1 \ldots n, \tag{2}$$

where $\overline{C_i(P)}$ and $\overline{S_i(P)}$ denote the projections of $C_i(P)$ and $S_i(P)$ onto a main plane parallel to *P*, *n* is the number of cuts and $\otimes^*$ denotes the regularized XOR operation. Applying the definition of the XOR operation, this last equation can be expressed as:

$$\overline{C_i(P)} = \overline{S_{i-1}(P)} \otimes^* \overline{S_i(P)} = \left(\overline{S_{i-1}(P)} -^* \overline{S_i(P)}\right) \cup \left(\overline{S_i(P)} -^* \overline{S_{i-1}(P)}\right), \tag{3}$$

and, thus, any cut, *C*, is decomposed into two terms named *forward difference* $\overline{FD_i(P)}$ and *backward difference* $\overline{BD_i(P)}$.

$$\overline{FD_i(P)} = \left(\overline{S_{i-1}(P)} -^* \overline{S_i(P)}\right) \qquad \overline{BD_i(P)} = \left(\overline{S_i(P)} -^* \overline{S_{i-1}(P)}\right) \tag{4}$$

FDi(P) is the set of faces on C_i *(P)* whose normal vector points to one side of the main axis perpendicular to C_i *(P)*, while *BDi(P)* is the set of faces pointing to the opposite side. In the example of Fig. 2, *FD* are marked with right-pointing arrows and *BD* with left-pointing arrows. This property guarantees correct orientation and, together with the fact that non-extreme vertices can be obtained from *ev*, provide proof that the EVM is a complete solid model. See [1] for more explanations concerning EVM.

Boolean operations on the EVM exploit the fact that the set of *ev* can be ordered following an spatial criterion and apply space and plane-sweep based techniques [12].

In [2] and [13] an algorithm that extracts the boundary from a given volume dataset based on the EVM is presented. This algorithm converts first the voxel model to the EVM and then extracts the boundary (a complete B-Rep model) which consists of general orthogonal faces.

The EVM to B-Rep conversion algorithm works recursively in the dimension. It performs three sweeping processes in order to obtain the three sets of faces parallel to the main planes. From the set of cuts it computes the set of sections and from them, *FD* and *BD* corresponding to each cut. In this process the non-extreme vertices are also obtained.

The same process is performed recursively in 2D for the *FD* and *BD* of each cut. Any *FD* and *BD* is a 2D EVM. The output of this algorithm is a set of oriented edges involving all the contours of the 2D EVM processed. Then, in the first proposal of this method, global contour construction was performed following a domino-like strategy and inclusion relationships were computed applying exhaustive point inclusion tests.

3. NEW 2D-EVM TO B-REP CONVERSION ALGORITHM

The goal of this work is to improve the 2D part of the EVM to B-Rep conversion algorithm, briefly explained in previous section. It is a new 2D EVM to B-Rep conversion algorithm, based on sweeping techniques. It presupposes that the previous 3D EVM to B-Rep conversion algorithm has been executed and that we have, for each *C*, its *FD* and *BD* with their corresponding normal vectors. Any *FD* and *BD* is a 2D EVM composed by 1D cuts (cuts, *C*, from now on). Therefore, we have a set of 2D EVM to process. For instance, in Fig. 2, $C_4(P)$ has a *FD* with 2 contours (2 cuts with 2 brinks each) and a *BD* with 1 contour (2 cuts with 1 brink each).

On the other hand, obtaining the boundary of a 2D binary image can also be performed by a two step process. The first step is the EVM encoding of the image which is computed by the 2D version of a 3D existing algorithm [13]. This algorithm is based on the following property of the EVM: considering all the vertices of all the pixels of a binary image, those vertices shared by an odd number of pixels are extreme vertices. In 2D there are 2^4 =16 possibilities that can be grouped into 6 configurations by rotational symmetries (see Fig. 3) and configuration 1 and 3 correspond to extreme vertices. The second step is the method that we present in the following subsections.

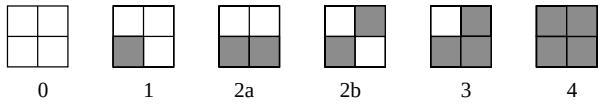


Fig. 3. 2D configurations.

3.1 General algorithm

The input of the presented method is a 2D EVM and the output is the complete set of oriented contours (faces and holes) together with their inclusion relationships. See Fig. 4.

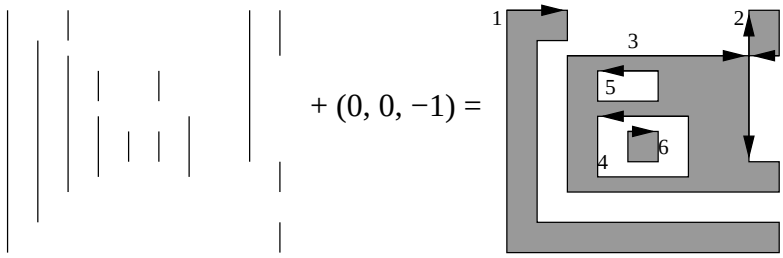


Fig. 4. A 2D EVM with 9 cuts plus the normal vector associated, **n** = (0, 0, -1), equals the complete boundary obtained: faces 1, 2 and 6 without holes and face 3 with holes 4 and 5. Faces 2 and 3 share a non-manifold (non-extreme) vertex.

The method performs two plane-sweeping processes in the same direction. The first one computes the 1D sections and obtains the 2D contours, as sets of oriented edges, while the second one classifies the contours as face or hole and obtains the inclusion relationships between them. The computed 1D sections are stored between both processes because the second one also needs them.

The first sweeping process maintains two lists. One list for the extreme vertices (*ev)* of the active section in which each *ev* has a pointer to the contour to which belongs and another list for the contours, in which each contour is associated with all the edges already classified as belonging to it. The process is summarized as follows. Each new cut *C* is merged with the active section computing the next section, applying Eqn. 1, and doing, at the same time, a contour classification. Merging a section and a cut is an XOR process between the *ev* of both sets (actually the XOR is performed to the projection of the entities onto a main parallel line). If an *ev* of the cut matches with one of the active section, it implies that the *ev* of the cut belongs to the contour associated with the *ev* in the active section (two *ev* match if their projections to the mentioned main parallel line coincide). Then we can classify each brink *(ev1, ev2)* of the cut as follows:

- If neither *ev1* nor *ev2* belong to any contour, then add a new contour with this brink as an edge.
- if both *ev1* and *ev2* belong to the same contour, then add this brink as a new edge of the contour
- if *ev1* and *ev2* belong to different contours, then merge both contours and add this brink as a new edge

To sum up, this first process performs a partition of the vertical and horizontal brinks of the 2D EVM, each equivalence class of it being the set of oriented edges of a contour.

The second sweeping process classifies the contours as face or hole and obtains the inclusion relationships between them. It simply traverses the sections computed in the first process looking up to the contour associated with each vertex of the current section. These lists of contours reflect in 1D the 2D situation and, therefore, the inclusion tests needed to classify the contours are actually made in 1D. Note that the contours corresponding to faces appear before those of their corresponding holes and that all the contours appearing in the first section are faces. Then, when processing a section, we can classify as face or hole all the contours not yet classified by testing 1D inclusion relationships.

In a section, pairs of *ev* corresponding to brinks of this section, as they represent the interior of the object, will allow to identify faces, whereas pairs of *ev* of different consecutive brinks, as they bound exterior zones, will allow to identify holes. Therefore, denoting these pairs of *ev* as B_H and B_F respectively, we can apply the following rules:

- a contour *H* is a hole of a contour *F* if $B_H \subset B_F$ and B_F is the brink in the current section corresponding to the most internal face contour that matches this condition

- a contour *F* is a face if it doesn't exist any contour *r* such that $B_F \subset B_r$ or if there exists a hole contour *H* such that $B_F \subset B_H$ and B_H is the brink in the current section corresponding to the the most internal hole contour that matches this condition

Finally, as any obtained individual contour is an unordered set of vertical and horizontals edges, already classified as faces or holes, a domino-like process is applied to each of them ordering the edges clockwise or counterclockwise depending on the normal vector direction of the 2D EVM processed.

3.2 Case example

In order to clarify these processes, we include an example (see Fig. 5). The sweeping processes are performed parallel to the *Y* axis. For the first process any section is represented by a list of extreme vertices (*ev*) with their associate contour *r*, $[(ev_{i1},r_{i1})(ev_{f1},r_{f1})\ldots(ev_{in},r_{in})(ev_{fn},r_{fn})]$, *n* being the number of brinks of the section and *i* and *f* standing for the initial and final *ev* of the brink. All the operations are performed with *y* coordinates but we store a whole vertex (the last extreme vertex encountered having this *y* coordinate) in order to construct the contours. *R* is the current list of contours with their corresponding lists of edges $R = [\ r_1, \{v^1_{i1} - v^1_{f1},\ldots\},\ldots, r_m, \{v^m_{i1} - v^m_{f1},\ldots\}]$, *m* being the total number of contours that will be found and *v* standing for a vertex, extreme or non-extreme. We show the process for the first 7 cuts.

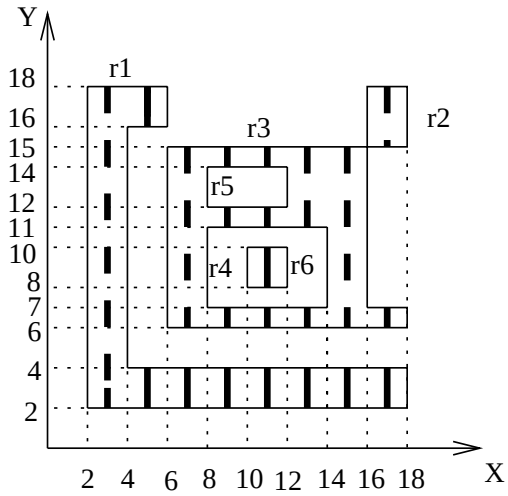


Fig. 5. Case example. Thick lines show the sections.

$C_{x=2}$: brink (y=2, y=18)
$S_{x=2}$ = [((2, 2), 1) ((2, 18), 1)]
R = [1 {(2, 2) - (2, 18)}]

$C_{x=4}$: brink (y=4, y=16)
$S_{x=4}$ = [((2, 2), 1)((4, 4), 2) ((4, 16), 2)((2, 18), 1)]
R = [1 {(2, 2) - (2, 18)}, 2 {(4, 4) - (4, 16)}]

$C_{x=6}$: brinks (y=6, y=15), (y=16, y=18). Brink (y=16, y=18) allows to merge contours 1 and 2
$S_{x=6}$ = [((2, 2), 1) ((4, 4), 1) ((6, 6), 3) ((6, 15), 3)]
R = [1 {(2, 2) - (2, 18), (4, 4) - (4, 16), (6, 16) - (6, 18), (4, 16)-(6, 16), (2, 18) - (6, 18)}, 3 {(6, 6) - (6, 15)}]

$C_{x=8}$: brinks (y=7, y=11) (y=12, y=14)
$S_{x=8}$ = [((2, 2), 1) ((4, 4), 1) ((6, 6), 3) ((8, 7), 4) ((8, 11), 4) ((8, 12), 5) (8, 14), 5) ((6, 15), 3)]
R = [1 {idem}, 3 {idem}, 4 {(8, 7) - (8, 11)}, 5 {(8, 12) - (8, 14)}]

$C_{x=10}$: brink (y=8, y=10)
$S_{x=10}$ = [((2, 2), 1) ((4, 4), 1) ((6, 6), 3) ((8, 7), 4) ((10, 8), 6) ((10, 10), 6) ((8, 11), 4) ((8, 12), 5) (8, 14), 5) ((6, 15), 3)]
R = [1 {idem}, 3 {idem}, 4 {idem}, 5 {idem}, 6 {(10, 8) - (10, 10)}]

$C_{x=12}$: brinks (y=8, y=10) (y=12, y=14) close contours 6 and 5 respectively
$S_{x=12}$ [((2, 2), 1) ((4, 4), 1) ((6, 6), 3) ((8, 7), 4) ((8, 11), 4) ((6, 15), 3)]
R = [1 {idem}, 3 {idem}, 4 {idem}, 5 {(8, 12) - (8, 14), (12, 12) - (12, 14), (8, 12) - (12, 12), (8, 14) - (12, 14), }, 6 {(10, 8) - (10, 10), (12, 8) - (12, 10), (10, 8) - (12, 8), (10, 10) - (12, 10)}]

$C_{x=14}$: brink (y=7, y=11) closes contour 4
$S_{x=14}$ = [((2, 2), 1) ((4, 4), 1) ((6, 6), 3) ((6, 15), 3)]
R = [1 {idem}, 3 {idem}, 4 {(8, 7) - (8, 11), (14, 7) - (14, 11), (8, 7) - (14, 7), (8, 11) - (14, 11) }, 5 {idem}, 6 {idem}]

For the second process, the computed and stored sections are used to obtain the inclusion relationships. $S_{x=2}$ is used to label r_1 as a face. $S_{x=4}$ doesn't contribute because there is any new contour to classify (r_2 has been merged with r_1). $S_{x=6}$ is used to label r_3 as a face. $S_{x=8}$ is used to label r_4 and r_5 as holes of face r_3. $S_{x=10}$ is used to label r_6 as a new face and in $S_{x=12}$, $S_{x=14}$, $S_{x=16}$ and $S_{x=18}$ there aren't new contours to classify.

3.3 Non-manifold vertices

Dealing with non-manifold vertices (NMV) requires a special treatment. Although in many cases these vertices would be correctly treated applying the general algorithm explained in Sec. 3.1 there are cases in which this general algorithm won't be able to correctly merge edges of the same contour. Fig. 6 shows two cases that can be determined and one that cannot.

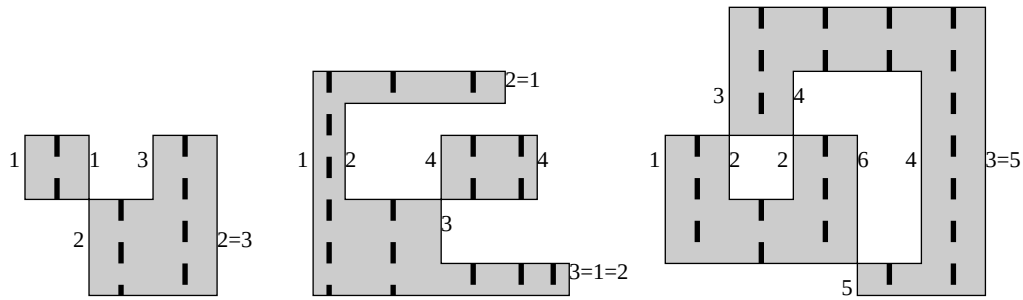


Fig. 6. Examples with NMV. (Left) A trivial example: edges can be correctly labeled when processing the line of vertices with the NMV. (Middle) An example with a NMV, in which contour label equivalences will be set at the end. (Right) An example in which we are not able to set all the contour label equivalences.

When running the first sweeping process, a non-manifold vertex can easily be detected when a brink of the current cut contains one *ev* of the current section (always in the projections to the mentioned main parallel line). But unfortunately, it is not possible to know whether a NMV will be determined at the end of the process or not. The strategy followed is that when a NMV is encountered, both edges converging to it are labeled with the same contour number. Moreover, the contour is marked as having NMV and the vertex is split into two. If a line of vertices has more than one NMV, then all the edges converging to NMV will have the same label. At the end, all the contours sharing NMV will have the same contour label and will constitute a non-manifold face. The separation of these kind of faces into manifold ones will be performed later.

Actually, if the application that uses the obtained B-Rep can deal with non-manifold faces, we won't need to split them into manifold ones and we will continue applying the general algorithm. But if we need to separate non-manifold faces into manifold ones, we have also to adapt the two remaining processes.

In the second sweeping process, when a hole H is found that belongs to a non-manifold face F, it is assigned to this face. At this step, we are still not able to split a non-manifold face into its set of manifold faces in order to assign definitely the holes encountered. But when we are analyzing the section in which $B_H \subset B_F$, we can associate to any of the two vertices of B_F the encountered hole and this relation will allow to assign this hole to its manifold face in the last process.

Finally, when the domino-like process is performed to a non-manifold face, we only have to follow the simple rules shown in Fig. 7 to broke correctly a non-manifold face into its set of manifold ones and each hole is assigned to its corresponding manifold face, based on the associated vertex.

Fig. 7. Rules applied to obtain manifold contours from a non-manifold face or hole.

3.4 Case example (continued)
For the first process only two cuts remain. In the first one there is a non-manifold vertex and the second is the last one and closes all the still open contours.

$C_{x=16}$: brink (y=7, y=18). It contains ev = (6, 15) from the previous section, so there is a non-manifold vertex (16, 15)*
$S_{x=16}$ = [((2, 2), 1) ((4, 4), 1) ((6, 6), 3) ((16, 7), 3)((16, 15), 3) ((16, 18), 3)]
R = [1{idem}, 3 {(6, 6) - (6, 15), (16, 7) - (16, 15)*, (16, 15)* - (16, 18), (6, 15) - (16, 15)*}, 4{idem) }, 5{idem}, 6 {idem}]

$C_{x=18}$: brink (y=2, y=4) closes contour 1; brinks (y=6, y=7) and (y=15, y=18) close contour 3
$S_{x=18} = \varnothing$
R = [1 {{(2, 2) - (2, 18), (4, 4) - (4, 16), (6, 16) - (6, 18), (4, 16)-(6, 16), (2, 18) - (6, 18), (18, 2) - (18, 4), (2, 2) - (18, 2), (4, 4) - (18, 4)},
3 {(6, 6) - (6, 15), (16, 7) - (16, 15)*, (16, 15)*- (16, 18), (6, 15) - (16, 15)*, (18, 6) - (18, 7), (18, 15) - (18, 18), (6, 6) - (18, 6), (16, 7) - (18, 7), (16, 15)* - (18, 15), (16, 18) - (18, 18)},
4 {idem) }, 5 {idem}, 6 {idem}

In the second process, when $S_{x=8}$ is processed, holes r_4 and r_5 will be associated to vertex (6, 6) (or to vertex (6, 15)). Finally, the domino-like process will be able to split the non-manifold contour r_3 into two manifold ones (which we label as r_3 and r_2) and to assign definitely holes r_4 and r_5 to contour r_3. When traversing the edges of the initial non-manifold contour, the ambiguity encountered in the non-manifold vertex is solved following the referred rules shown in Fig. 7. Then, we have:

r_3 = {(6, 6) - (6, 15), (6, 15) - (16, 15)*, (16, 15)* - (16, 7), (16, 7) - (18, 7), (18, 7) - (18, 6), (18, 6) - (6, 6)}

r_2 = {(16, 15)*- (16, 18), (16, 18) - (18, 18), (18, 18) - (18, 15), (18, 15) - (16, 15)*}

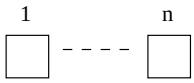
4. RESULTS
In this section we will compare the previous approach (PA) with the new one presented in this paper (NA). Doing a theoretic complexity analysis of both methods is beyond the scope of this work. So, we first will evaluate the complexity of both methods performing a qualitative analysis of the involved processes and then we will show running times for several representative phantom examples and real datasets.

The PA method [2] performs a plane-sweep and computes sections (XOR operation) and 1D *FD* and *BD*, which involve 1D Boolean operations (See Eqn. 4). This gives the full set of oriented edges of this plane. The same process is performed for both directions, X and Y. Then, the contours are computed performing a domino-like process among all the edges of a 2D EVM and are labeled as face or hole, depending on the orientation of their edges. Finally, for each hole, the method performs a search among all the faces of the 2D EVM to find to which face it belongs by applying point-in-polygon inclusion tests.

The NA method also computes sections but only in one direction and at the same time maintains and computes the list of contours. Then it needs to traverse again the sections (but not to compute them again) to obtain the inclusion relationships between contours and without any point-in-polygon inclusion test. Finally, it needs also to perform domino-like processes but among the edges of each contour separately.

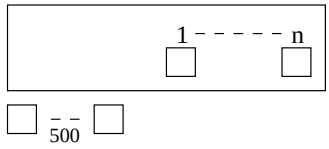
We present the following phantom examples. All the running times are in seconds and the computer used to do these comparisons is a Pentium III at 997 MHz and 512 RAM. The first one is a sequence of simple parallelepipeds. Fig. 8 (left) shows the schematic example and running times. We can see that the behavior of the NA method is linear while the PA is almost quadratic. We have performed a similar test, but with a sequence of objects as those of Fig. 1 right (with a non-manifold vertex each), and the running times are slightly superior for both methods but follow the same pattern.

phantom1



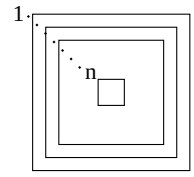
N	*PA*	*NA*
10	0.01	0.03
100	0.23	0.09
200	0.55	0.15
300	1.23	0.21
400	1.79	0.25
500	2.60	0.29

phantom2



n	PA	NA
50	4.36	0.37
100	6.65	0.42
150	8.80	0.48

phantom3



n	PA	NA
10	0.06	0.03
40	0.38	0.16
80	1.45	0.46
160	5.66	1.78

Fig. 8. (Left) First example with running times. (Middle) Example testing holes. (Right) Example testing inclusion levels.

The second and third tests are intended to evaluate cases with several holes and with several levels of inclusion respectively. Fig. 8 (Middle) and (Right) shows the schematic examples and running times. For both cases the running times in both methods are greater than in the simple case showed in Fig. 8 (Left), but the new approach performs rather better than the previous one.

Finally we have processed several well-known 3D real datasets. We have applied both methods to all the 2D cuts and we have computed the average times. Tab. 1 shows, for these datasets, the number of extreme vertices (#ev), the number of cuts in the three main directions (X-cuts, Y-cuts, Z-cuts) and the average running times in seconds for the previous (PA) and new approach (NA). The time of the PA in both cases is about 4 times the time of the NA. As we have expected, these real cases don't give the time differences we have obtained in the phantom examples, which have been analyzed to evaluate the behavior of both methods in extreme situations.

Dataset	#ev	X-cuts	Y-cuts	Z-cuts	PA	NA
Monster	4,288	137	60	103	0.012	0.010
Skull	23,464	123	176	129	0.060	0.018
Aneurism	70,260	427	430	470	0.107	0.021
Engine	85,866	276	389	213	0.119	0.030
Lobster	102,394	485	476	97	0.102	0.023

Tab. 1. Running times (in sec.) for several real datasets of the previous approach (PA) and the new one (NA).

.
Fig. 9 shows one of these datasets. Images of all of them as well as their EVM codification can be found in http://truja.lsi.upc.edu/movibio/ResearchLines/VEVM/.

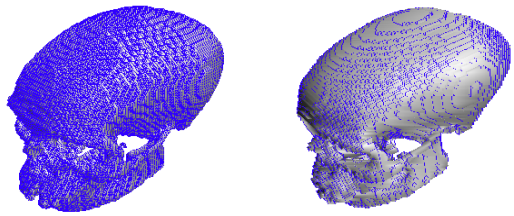


Fig. 9. Skull example: voxelization (left) and 3D model EVM (right).

5. CONCLUSIONS AND FUTURE WORK

An improved 2D EVM to B-Rep conversion method has been presented that complements a 3D existing method. It basically substitutes its last process consisting of the 2D contours arrangement and classification, which were performed in a global not optimized way, by a plane-sweep like process which follows the same methodology of the main 3D algorithm and of most of the EVM involved methods. As a 2D EVM is a suitable codification of a 2D binary image, this method also can be applied to extract the complete boundary of a 2D image. A qualitative analysis has been performed that allows to conclude that the new version is faster than the previous one. This conclusion has been corroborated by experimental results.

We have thought to extend this algorithm to 3D to perform connected component labeling and compute the containment tree as in [3], but there exists an algorithm that solves this problem based on the EVM [13]. Although the method is slightly different, it also follows a space-based sweeping strategy and uses the information associated to the sections of the model, so we don't believe that it could be further improved.

Initially the EVM was developed to represented binary datasets. In [14], it has been extended to represent heterogeneous objects or volumes and the extended model has been called VolumeEVM. As a future work, we are studying the adaptation of the EVM developed operations, as Boolean operations, for heterogeneous objects as well as the suitability of this model to represent time-varying data sets, by taking profit of spatial and temporal coherence.

6. ACKNOWLEDGMENTS

This work has been partially supported by the projects MAT2002-04297-C03-02 and IM3: Molecular Image and Multimodality from the Spanish government and by the CREBEC from the Catalan government.

7 REFERENCES

[1] A. Aguilera. Orthogonal Polyhedra: Study and Application. PhD thesis, Universitat Politècnica de Catalunya, 1998.

[2] A. Aguilera and D. Ayala. Converting Orthogonal Polyhedra from Extreme Vertices Model to B-Rep and to Alternative Sum of Volumes. *Computing Suppl. Springer-Verlag*, Vol. 14, pp. 1-28, 2001.

[3] I. Gargantini, G. Schrack, and A. Kwok. Reconstructing multishell solids from voxel-based contours. *Computer-Aided Design*, Vo. 26, No. 4, pp. 293-301, 1994.

[4] H. J. A. M. Heijmans. Connected morphological operators for binary images. *Computer Vision and Image Understanding*, Vol. 73, No. 1, pp. 99-120, 1999.

[5] G. J.Grevera, J. K. Udupa, and D. Odhner. An Order of Magnitude Faster Isosurface Rendering in Software on a PC than Using Dedicated, General Purpose Rendering Hardware. *IEEE Trans. Visualization and Computer Graphics*, Vol. 6, No. 4, pp. 335-345, 2000.

[6] L. Latecki. 3D Well-Composed Pictures. *Graphical Models and Image Processing*, Vol. 59, No. 3, pp. 164- 72, 1997.

[7] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surfaces construction algorithm. *Computer Graphics*, Vol. 21, No. 4, pp. 163-169, 1987.

[8] C. Montani, R. Scateni, and R. Scopigno. Discretized marching cubes. In *Proceedings of visualization*, pp. 281-287, 1994.

[9] C. Montani and R. Scopigno. Quadtree/Octree to boundary conversion. In *Graphics Gems II*, Academic Press, Inc, chapter IV.7, pp. 202-218, 1991.

[10] H. Muller and M. Stark. Adaptive generation of surfaces in volume data. *Visual computer*, Vol. 9, pp. 182-199, 1995.

[11] S. C. Park and B. K. Choi. Boundary extraction for cutting area detection. *Computer-Aided Design*, Vol. 33, pp. 571-579, 2001.

[12] F. P. Preparata and M. I. Shamos. Computational Geometry: an Introduction. Springer-Verlag, 1985.

[13] J. Rodríguez, D. Ayala and A. Aguilera. EVM: A Complete Solid Model for Surface Rendering. In *Geometric Modeling for Scientific Visualization*. Springer, pp. 259-274, 2004.

[14] J. Rodríguez, D. Ayala, and S. Grau. VolumeEVM: A new approach for surface/volume integration. *Computers & Graphics*, Vol. 29, pp. 217-224, 2005.

[15] A. Rosenfeld, T. Kong, and A. Wu. Digital surfaces. *CVGIP: Graphical Models and Image Processing*, Vol. 53, No. 4, pp. 305-312, 1991.

[16] R. Shekhar, E. Fayyad, R. Yagel, and J. F. Cornhill. Octree-based decimation of marching cubes. *IEEE Computer Graphics and Applications*, pp. 335-342, 1996.

[17] J. Udupa and O. Odhner. Fast visualization, manipulation and analysis of binary volumetric objects. *IEEE Computer Graphics and Applications*, Vol. 4, No. 1, pp. 53-62, 1991.