Border Map: A Topological Representation for nD Image Analysis

Yves Bertrand, Christophe Fiorio, Yann Pennaneach

Border Map: A Topological Representation for nD Image Analysis

Yves Bertrand[1], Christophe Fiorio[2], and Yann Pennaneach[1]

[1] LSIIT, 7 Rue René Descartes, F-67084 Strasbourg Cedex
[2] LIRMM, 161 rue Ada, F-34392 Montpellier Cedex 5

Abstract. This article presents an algorithm computing a *border map* of an image that generalizes to the n dimension graph structures used in image analysis. Such a map represents simple and multiple adjacencies, inclusion of regions, as well as the frontier type between two adjacent regions. An algorithm computing a border map, linear to the number of elements of an image, is defined in 2D, then generalized in 3D and in nD.

Keywords: image modeling, adjacency graph, topology, combinatorial maps, generalized maps.

1 Introduction

Before analyzing effectively the contents of an image and starting a process of recognition or data analysis, it is necessary to extract the valuable information and to structure it. This first stage of all analysis process is called image segmentation. Several approaches exist, here we are interested in regions segmentation which segments the image into connected component of pixels. Once this partition obtained, it is necessary to structure this information so that more accurate or specialized processes can be applied. Our aim here is to propose a graph like structure in order, not only to allow more sophisticated analysis of the image, but also to allow a modeling and thus facilitating the visualization and the interactive manipulation of the segmentation result.

A commonly used structure is the region adjacency graph [Ros74]. But such a graph does not indicate the type of adjacency between regions: it does not specify neither the possible relations of inclusion between regions, nor if two regions have one or several common borders. Kropatsch and Macho proposed in [KM95] a dual planar multi-graphs structure allowing to code all these informations. The inclusion is coded with the help of a fictive edge that appears as a loop in the dual graph which allows to differentiate it from classical adjacency edges. Operations are thus homogeneous but require to maintain two graphs. Brun [Bru96] uses planar maps to which he adds some fictive edges to code the inclusion. But the resulting structure is no more homogeneous.

In fact to code information about multiple adjacencies and inclusion leads to realize the topology of the image. Fiorio [Fio96] has introduced the notion of topological graph of frontiers (GTF for short). The interest of a GTF is not

G. Bertrand, M. Couprie, L. Perroton (Eds.): DGCI'99, LNCS 1568, pp. 242–257, 1999.

only to define a topological structure richer in information than the classical structures, but also computable in linear time to the number of pixels of the image, with only two consecutive lines of the image in memory. The immediate problem for this optimal algorithm is its extension to 3D images, and nD images. The purpose of this paper is to propose one solution to this problem. The solution requires the use of the resulting structure of a regions segmentation, and related algorithms of topology-based modeling: combinatorial maps [Jac70,Cor75] and generalized maps [Lie89,Lie91].

We begin, Section 2, by introducing in an informal manner concepts of topology and combinatorial maps. Section 3 gives a short presentation of the GTF, then Section 4 proposes a simplification of this structure, called *border map*. Section 5 indicates how the algorithm which computes such a structure is extended to the 3D and then to the nD case. Section 6 summarizes up the main features of the algorithm presented, and Section 7 gives some examples of applications. Finally Section 8 concludes this paper.

2 Topology and combinatorial maps

Defining a coherent topology for images that are discrete spaces is not a trivial problem. Some simple and efficient solutions exist for the binary case (cf [KR89]) but they are not translatable for grey levels or color images, the case that interests us here. In fact it appears that to define a topology for spaces such as images, it is necessary to introduce and to define a notion of border. Kovalevsky [Kov89], first, proposes to no more consider an image as only a set of pixels and introduces border elements (edges and vertices) of pixels as elements of the image. Khalimsky, Kopperman, and Meyer present in [KKM90] a solution of the same type where the image is defined like a cartesian product of discrete straight lines. Based on these works [Fio95] proposes an interpixel approach for image analysis where he also takes into account border elements (see Figure 1). He

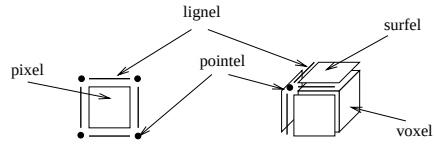


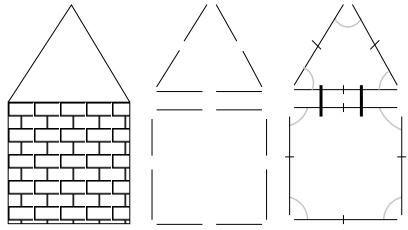
Fig. 1. Interpixel elements for a 2D and 3D image.

defines a topology well adapted for image analysis: the *star-topology* [AAF95]. This topology and the one proposed by Khalimsky, Kopperman and Meyer are combinatorial topologies leaning on the notion of space subdivisions: the space is partioned into elements and border elements and provides incidence and adjacency relations linked to this notion of border. Thus $\mathbb{R}^3$, for example, will be

divided into volumes (basic elements), surfaces (border of volumes), curves (border of surfaces) and vertices (border of curves). A vertex is said to be incident to a curve if it is a border of the curve. A surface is incident to a volume if it is a part of its border. Two volumes are adjacent if they are incident to a same surface.

Generalized maps [Lie97] are a combinatorial structure allowing to code this subdivision, as well as the incidence relations between elements of the space, thus realizing the topology. "*Implementation of these structures (and related algorithms) can be done without loss of information and without loss of properties*". In fact maps or generalized maps [Lie89,Lie91] generalize the combinatorial map notion, which is the first mathematical model for topology-based representation [Jac70,Cor75].

Generalized maps are therefore concerned with modeling subdivision of geometric spaces. For example the object in Figure 2.a is modeled by the G-map of Figure 2.c. This last is obtained by successive decompositions of the object, starting by distinguishing the faces (the triangle and the rectangle), then the edges and at last vertices incidents to edges. Finally we get 14 half-edges called *darts*. An involution, noted α_0 and illustrated on faces by a little stroke, associates to a dart the dart of the adjacent vertex of the same edge and of the same face. Another involution, noted α_1 and illustrated by the gray arc, associates to a dart the dart of the adjacent edge incident to the same vertex and belonging to the same face. Finally an involution, noted α_2 and illustrated by a thick stroke, associates to a dart the dart of the adjacent face of the same edge and incident to the same vertex. In short, involution α_0 puts into correspondence vertices (cells of dimension 0) of the object, involution α_1 link up edges (cells of dimension 1) and involution α_2 joins two faces adjacent (cells of dimension 2).



a. 2D object b. darts c. 2-G-map

Fig. 2. From a 2D object to a 2-G-map

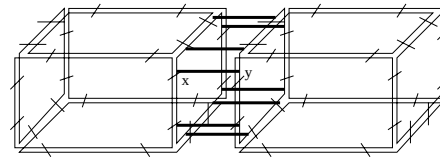


Fig. 3. 3-G-map modeling two adjacent cubes

In general, an involution α_i associates to a dart of a $i-cell\ c$ the dart of the i-cell adjacent to c.

G-map of dimension 2, or 2-G-map (in the following, we writes k-x instead of x to k dimension) realizing the topology of the object of Figure 2.a is the algebra $(B, \alpha_0, \alpha_1, \alpha_2)$, where B is the set of the 14 *darts*, resulting from the

decomposition, and verifying the fact that $\alpha_0 \circ \alpha_2$ is also an involution. The links between darts are called *sewing*.

Extending of G-maps to the third dimension is immediate: it consists in sewing 2-G-maps by an additional involution, noted α_3. Figure 3 represents two cubes sewn along a face (involution α_3 is represented by the thick stroke). Definition of n dimensional G-maps is done in the same way using $n+1$ involutions $\alpha_0, \ldots, \alpha_n$. A G-map may or not represent some orientable subdivisions [Lie89]. Intuitively, it is sufficient to cross two 2-sewings in order to get the topology of a Moebius ring.

An orientable subdivision represented by a G-map can be done by a map with two times less darts that a G-map. Thus, the 2-G-map of Figure 2.b is translated into the 2-map of Figure 4 by only keeping one dart of 2. Such a 2-map is an algebra (B', β_1, β_2), with a permutation $\beta_1 = \alpha_0 \circ \alpha_1$ (grey arcs on Figure 4) and one involution $\beta_2 = \alpha_0 \circ \alpha_2$ (diagonal stroke Figure 4), where B' are the thick darts on Figure 4.

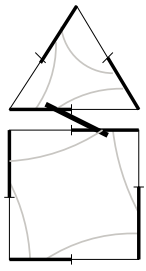
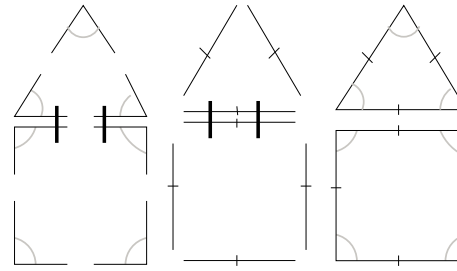


Fig. 4. 2-map Figure 2.b



a. 1-cells b. 2-cells c. 3-cells

Fig. 5. Cells of 2-G-map of Figure 2.c

2-G-maps allow to describe all types of subdivisions of 2-manifolds, and in particular subdivisions of 2D images in regions. They are characterized by the fact that the set of darts can be partitioned into sets corresponding to vertices, edges, or again to faces, whereas this is not the case for darts of 2-maps. To get these sets, it is sufficient to remove respectively involution α_0 ((1) of Figure 2.c) α_1 ((2) of Figure 2.c), α_2 ((3) of Figure 2.c). This makes easier the writing of algorithms which need to operate on each of such types of cells. This is why, in all the following, algorithms are described in terms of G-maps and not maps.

3 Topological Graph of Frontiers

A *GTF* (see definition in [Fio95,Fio96]) corresponding to an image without inclusion of region is precisely a 2-map whose vertices represent regions and whose edges represent frontiers between regions. Such maps are the duals of those presented in Section 2: permutation on darts describe vertices instead of faces.

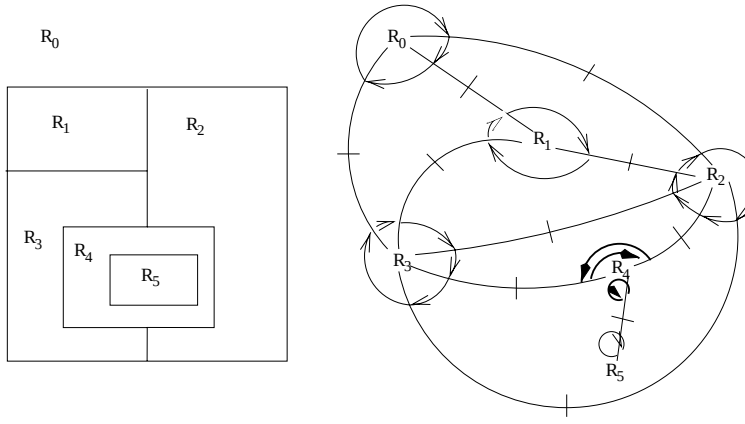


Fig. 6. A Topological Graph of Frontiers

The permutation around a vertex return all the adjacent regions to the region represented by this vertex. A region adjacent to r enclosed regions is represented by a vertex where all incident darts belong to not only one permutation, but to $r+1$ permutations. Figure 6 [Fio96], shows two permutations (grey arrows) are associated to region R_4, one corresponding to the external contour of R_4, and the other to the internal contour.

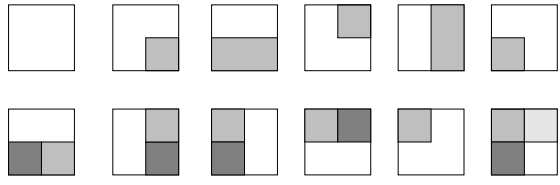


Fig. 7. Precodes

In order to extract a *GTF* from an image, every pixel is assumed to be associated to the region it belongs to. This association is the result of a segmentation and a region labeling, linear in the number of pixels of the image [FG96]. The extraction algorithm [Fio96] is a greedy algorithm: it scan the image line by line, starting with a *GTF* reduced to one vertex corresponding to the outside region of the image, noted R_0 in Figure 6. The scan is done with a 2 pixels window and updates the *GTF* under construction according to 12 possible configurations, called precodes, shown in Figure 7. The algorithm manages an auxiliary list of darts that allows to sew the current darts of one line to darts of the previous line. It is very efficient, since it treats an image of 512×512 in about $0,5$ seconds on a workstation of entrance range.

However, this algorithm presents some drawbacks making it difficult to translate to nD:

- Some retroactive corrections are necessary because, given two distinct contours currently processed, we cannot know if they will merge or not (see Figure 8). Corollarly, the GTF under construction does not necessarily correspond to the part of the image already scanned. The correspondence becomes correct only at the end of the process.
- An auxiliary list of darts and precodes is used in order to scan only once the image and in order to achieve the optimal complexity.
- The inclusion tree obtained is implicit.

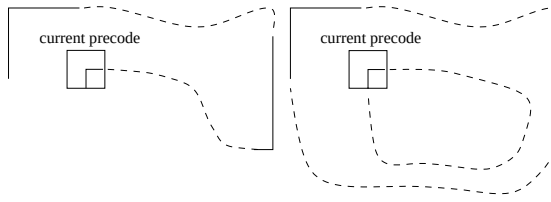


Fig. 8. contours merging

The representation that we propose in this paper as well as the related algorithms become free of these drawbacks so that they facilitate the extension to nD and allow to code explicitly the inclusion tree. Embedding informations are also included in our representation in order to be able to allow the display.

4 Border map of regions

4.1 First algorithm

In order to get rid from drawbacks due to the algorithmic optimizations, let us reconsider the problem and forget momentarily questions of performances. We shall reintroduce them progressively.

Algorithm 1: Naive algorithm

Input: Image I of $n_1 \times n_2$ pixels
Output: 2-G-map of whose faces represent regions and edges borders between regions.

1 Build a 2-G-map G of $n_1 \times n_2$ squares sewn between them;
2 Merge all adjacent faces couples of a same region;
3 Merge all couples of edges of same direction incident to a vertex of degree 2;

Algorithm 1 creates the 2-G-map whose faces represent regions and edges borders between regions.

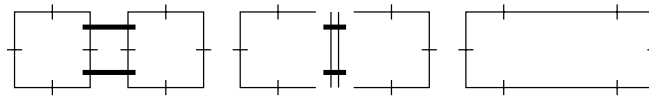


Fig. 9. Faces Merging

The merging of two faces along an edge (Figure 9.a) is done by first α_1-unsewing the edge from its four incident darts (Figure 9.b), then by resewing by α_1 the four darts of the two remaining face pieces become 1-free (Figure 9.c).

The steps described above do not manage the case of regions enclosed in others. A simple manner to describe implicitly the inclusion tree of regions is to indicate for every dart whether it belongs to an internal or an external contour of a region. From now on we will say respectively *internal or external dart* for a dart belonging to respectively external or internal contour. One couple of darts 2-sewn in which one is internal and the other one is external corresponds to an edge of the inclusion tree. The informal definition of an internal or external dart is the following:

- initially, all darts are considered internal,
- the first dart met for a given region is external,
- a dart belonging to a contour with at least one external dart is external.

The space complexity of this algorithm is $O(n_1 n_2)$[1]. Since for every dart, the face it belongs to is scanned in order to determine if it is external or not, the time complexity of the algorithm is $O(n_1 n_2 m)$, where m is the mean number of darts by face. These high complexities require optimization of the naive algorithm in time and in space.

4.2 Algorithm 1 optimizations: the border map

A first optimization of our algorithm consists to start with a G-map including only one square for the first pixel of the image and then, for every new created square immediately try to merge it with the current G-map. The first two steps of the naive algorithm are thus grouped in one. Figure 10 shows the current pixel in gray (Figure 10.a), that has been merged with the part of G-map containing a pixel on the previous line belonging to the same region, but not with the previous pixel on the same line since it is white and does not belong to the same region. The processing of a pixel consists therefore to apply, depending on the horizontally and vertically adjacent regions, zero, one or two times, the illustrated merge operation of Figure 10. This process requires to know the darts

[1] Reminder: the processed image contains $n_1 \times n_2$pixels.

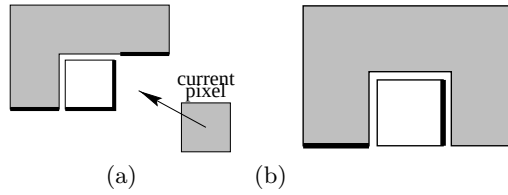


Fig. 10. Current pixel merging

of the image already processed with which the pixel must be sewn. For that, one dart of the pixel previously processed pixel (vertical dart in black on Figure 10.b) is memorized, as well as a dart in the beginning of the current line (horizontal dart in black on Figure 10.b), in the case of the current pixel being the first of the line. These darts belong to the set of darts represented in black to the figure 10.a that closes the G-map . It corresponds to the auxiliary list of darts of [Fio96].

The complexity in space is proportional to the number of darts of the G-map, which is about n_1 or n_2 for images with a weak number of regions, and about $n_1 n_2$ for images having a high number of regions. In practice, the number of darts is weaker than $n_1 n_2$ (about 6 to 10 times less darts).

A second optimization consists in not creating the current pixel explicitly, but foreseing all possible configurations. These are only 4, due to the fact that the current pixel (i, j) can or cannot be merged either with the pixel $(i-1, j)$, or with the pixel $(i, j-1)$. We obtain thus an algorithm with four precodes (see Figure 11).

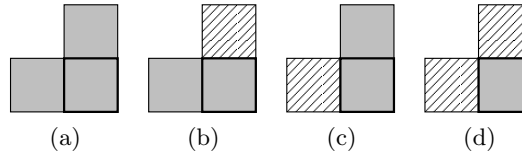


Fig. 11. 4 precodes

Figure 12 illustrates for each of the four cases of Figure 11 the corresponding operations to do on the G-map (Figure 12 .e). In the case (a), no dart is modified, only the coordinates of the vertex are changed, while the creation of a full pixel and the application of two merging operations have created and destroyed eight darts. For the case (d), eight darts are effectively created, and four are 2-sewn. For cases (b) and (c), identical up to the orientation, four darts are created and two are 2-sewn. The theoretical complexity in time does not change, but in practice, the time is highly reduced because, contrarily to the previous algorithm, that darts that are near be destroyed by merging are not created.

A third optimization consists in grouping steps based on 4 precodes and steps of edges merging. It requires then to consider the 12 studied cases by [Fio96].

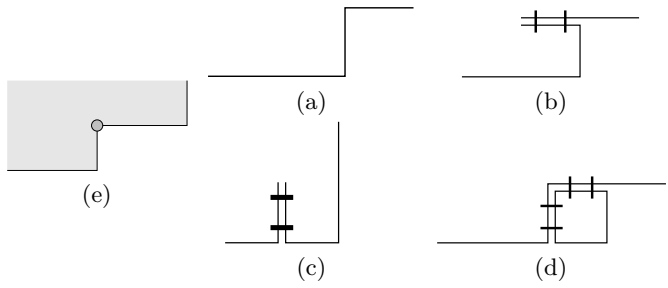


Fig. 12. Operations on the G-maps related to the 4 precodes

We can at the same time examine 3 additionnal precodes, corresponding to cases where regions are not 2-manifolds, these cases are examined exhaustively in [Pen97].

4.3 Inclusion tree

The algorithm of internal and external dart computation can be improved by indicating that all darts of a contour are external as soon as a first external dart has been met. But so, the inclusion tree remains implicit.

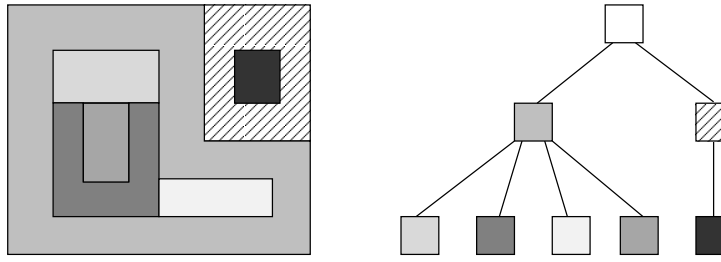


Fig. 13. Example of inclusion tree

In order to construct effectively an inclusion tree (an example of an inclusion tree is shown Figure 13), it is necessary to scan again the G-map connected component by connected component (see Algorithm 2). Algorithm 2 starts by assigning the outside region (line 1) (so-called infinite region) as root of the tree. Then it calls the *AddRegionsFilles* recursive procedure (line 3) which marks recursively the regions enclosed in others and so can add the corresponding vertex to the inclusion tree. This procedure starts with the first dart of the map (line 1). This dart is necessarily a dart of an internal contour of the region R. By applying α_2 (line 2) the dart of an external contour of an enclosed region in R is recovered. From this dart the connected component it belongs to is scanned by applying in any order any number of times involutions $\alpha_0, \alpha_1, \alpha_2$ (line 3). All

Algorithm 2: Creation of the inclusion tree

Input: Image I with a connected component labeling;
The G-map G of the image I
Output: Inclusion tree $\mathcal{TO}$

```
1 R ← ∞;
2 A ← R;
3 AddRegionsFilles(A,R);
```

Procedure `AddRegionsFilles`($\mathcal{TO}$: *an inclusion tree,* R: *root of the tree*)

```
if G ≠ ∅ then
1   b ← first dart of G;
2   b' ← α2(b);
3   foreach darts b_ext of the connected component of b' do
4     let R_f be the region of b_ext;
5     if R_f was not already met then
6       to add an edge R → R_f to A;
      to suppress the dart b_ext of G;
    foreach regions R_f direct descendent from R in A do
      AddRegionsFilles(A,R_f);
```

region Rf met during this scan lead to the creation of an edge $R \rightarrow Rf$ (lines 4 to 6 of the *AddRegionsFilles* procedure).

5 Border map of regions of an 3D image

The computation of the inclusion tree given to Section 4.3 stays in any dimension, with no modifications. Therefore we keep it to extend the construction of the G-map.

The first algorithm is extended easily to dimension 3 for an image of $n_1 \times n_2 \times n_3$ voxels. The creation of the initial 2-G-map is replaced by the one of a 3-G-map of $n_1 \times n_2 \times n_3$ voxels. Faces merging is replaced by volumes merging, edges merging by faces merging, and an edges merging is added.

The first optimization is translated just as merely: it is sufficient to replace the two possible faces merging by three possible volumes merging along the three faces of the current voxel in contact with the 3-G-map under construction. Figure 14 shows an example of merging: one new cube is created (a), 3-sewn along three of its faces (b), then the merging with the cube to the left is done (c).

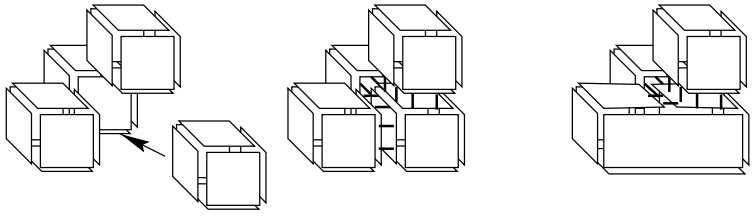


Fig. 14. Example of volumes merging

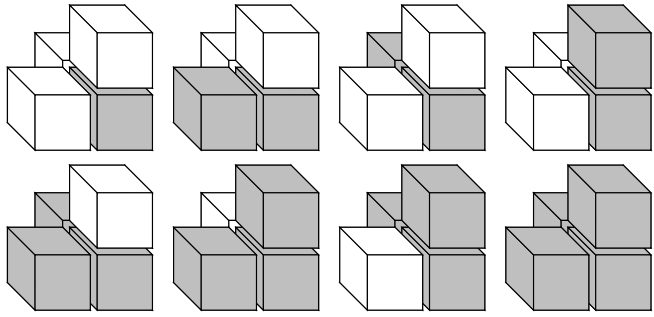


Fig. 15. 8 precodes for the construction of a 3-G-map

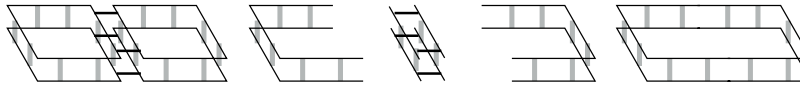
The extension of the second optimization consists only in replacing the 4 pixels-based precodes by 8 voxels-based precodes (see Figure 15 and Algorithm 4), this is due to the fact that each of the three faces of the current voxel in contact with the 3-G-map are either deleted or are not volumes merging.

Algorithm 4: Algorithm with 8 precodes

Input: Image I of $n_1 \times n_2 \times n_3$ voxels
Sortie: The corresponding 3-G-map

Create a first cube corresponding to the first voxel;
foreach *voxel* **do**
  Apply operations related to the current precode to 3-G-map;
foreach *edge of the obtained 3-G-map incident to only two faces* f, f' **do**
  Merge the faces f and f' (Figure 16);
foreach *Incident vertex to two edges* e *and* e' *not on the same line* **do**
  Merge the edges e and e';

The faces merging (illustrated Figure 16), consists to apply twice the previously described one for the dimension 2, whereas in 2D the merging edges is carried on couples of darts (one for each of the two faces), in 3D the merging is



(a) (b) (c)

Fig. 16. 3D faces merging

carried on 4-tuple of darts (one couple for every double-face). Therefore the first three algorithms are easily extended to 3D.

The extension to nD of the 3D algorithm above consists in replacing the 8 precodes by 2^n precodes of 4 cubes of dimension n, and to replace the volumes, faces and edges merging by the merging of n-,$(n-1)$-,..., 1-cells. This extension may have a convenient in 4D to process animated sequences of 3D images.

The extension of the last optimization is more difficult. A first approach consists in enumerating all possible precodes.

Let N_d be the number of precodes to the dimension d, N_d can be computed thanks to numbers S_n^k, called numbers of Stirling of first order. S_n^k gives the number of surjections of a set of n elements in a set of k elements. $\frac{S_n^k}{k!}$ gives then the number of partitions in k subsets of a set of n elements. In general $n = 2^d$, it is then sufficient to add partitions of one set, two, ..., n sets to get the number of precodes. One has therefore: $N_d = \sum_{k=1}^{2^d} \frac{S_{2^d}^k}{k!}$ knowing that $B_n = \sum_{k=1}^{n} \frac{S_n^k}{k!}$ where B_n are Bell numbers given by the recurrence formula: $B_n = \sum_{i=0}^{n-1} \frac{B_i \times n s!}{i! \times (n-i)!}$ with $B_0 = B_1 = 1$.

We can easily compute the number N_d of precodes for the dimension d. In 2D we have the 15 precodes (the 12 precodes of Figure 7 + the 3 cases of no manifold). In 3D, we get:
1 precode if only one color is present, 127 precodes for 2 colors, 966 precodes for 3 colors, 1701 precodes for 4 colors, 1050 precodes for 5 colors, 266 precodes for 6 colors, 28 precodes for 7 colors, 1 precode for 8 colors, thus a set of 4140 precodes (including cases of no-manifold)!

In 4D there are more than 10 000 millions of precodes, so that even without the non-manifold cases we cannot to consider an optimized solution based on precodes as the one for the *GTF* algorithm in [Fio96].

An approach under study for the 3D case consists in decomposing the process for eight voxels in the $2 \times 2 \times 2$ window into three processes of 4 voxels, analogous to the 2D precodes. Thus the number of configuration are limited to 36, with the possibility, to do the merging of edges in the same steps.

The exhaustive process of precodes is optimal in time, but impracticable. On the contrary, the solution with 8 precodes is simple, but requires to create numerous darts that will be immediately destroyed by merging. The solution of the 36 precodes seem to be able to be a good compromise between the difficulty of implementation work and performances in time.

6 Discussion

We indicate here what brings on our algorithm as opposed to the drawbacks evoked for the computation of a FTG in Section 3. Our algorithm constructs a G-map of closed contours in one first scan, and then the inclusion tree in the second. All the time during the construction process, the G-map describes the part of the image already scanned, and bring on our algorithm to be well-adapted to process infinite images. The algorithm of computation of a *GTF* does not possess this characteristic. Indeed the current map is not closed (darts corresponding to the closing are in the auxiliary list) but it is the price to pay in order to have an optimal algorithm.

Note that the second scan which build the inclusion tree takes place on the G-map, and does not require the image any more. But it requires a time proportional to the number of darts of the G-map, whereas the number of operations done by the one of [Fio96] is weaker since it is proportional to the number of contours. The counterpart is that the latter creates only an implicit inclusion tree, whereas the marking of darts allows to create an explicit inclusion tree.

The proposed 2D algorithms provides 2-g-maps with embedded straight darts, so it allows immediate display of the map. The counterpart is that the number of darts is bigger than in the *GTF* case. To get a map having a dart number analogous to the FTG, it is sufficient to avoid the test of alignment. Besides, this characteristic implies in 3D that only the G-map and the current voxel have to be memorized, whereas, without embedding, two slices of the image have to be kept in memory in addition. Moreover, the 2-G-map with straight darts allows to analyse the three cases where the regions are not manifolds.

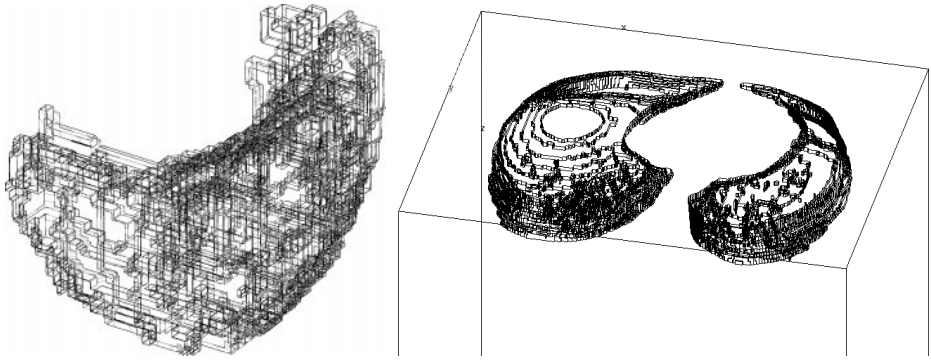
7 Application

The use of G-maps contributes to simplify and to generalize the algorithm. For the sake of performance in space, algorithms can be translated in terms of maps. Due to the actual memory capacity of workstations, this translation seems to be necessary as soon as we have 3D images.

The 2D algorithm provides performances comparable in time to the [Fio96] one. An image processed in $0,57s$ by the latter, requires $0,71s$ for our algorithm, which however provides in addition the inclusion tree and an embedable map. From a segmentation of a $512 \times 512 \times 192$ scanner image the computation of the border map of a stomach (Figure 17.a) requires $496s$ and 22576 darts. The one of the border map of lungs (Figure 17.b) requires $538s$ and 135466 darts. The different optimizations have all been written and tested in CaML. This implementation allowed to measure gains in time for each optimization stage. The final version is written in C.

8 Conclusion

The use of results inf topology-based geometric modeling has allowed to propose for image analysis a modeling of nD image by a border map. Successive



(a) Stomach (b) Lungs

Fig. 17. 3-maps of a stomach and a lung

optimizations of a naive algorithm have shown different aspects of the problem of the construction of a border map of an image allowing in the same time to simplify, optimize and extend the algorithms to nD.

[Fra95,Fra96] showed that concepts and methods developed in topology-based modeling self-apply to discrete geometry. The topology-based modeling often privileges the cellular models (in 2D, faces have any number of sides, and in 3D volumes have any number of faces), whereas the discrete geometry and image analysis are led by nature to simplicial structures [May67,LL95].

The experiments in 3D show that the algorithm is efficient in time, but that the size of images is strongly limited by the memory space available. It comes from the fact that cellular structures used are too general. They do not take into account that all cells are orthotrop. It seems to us that in order to partially clear of this limitation hybrid structures (i.e. simplicial for cells of some dimensions and cellular for others) should be used.

More generally, using general cellular and simplicial structures should allow not only to solve problems of structuring information in image analysis, and especially in 3D more comfortably than with ad hoc structures, but also to unify methods of structuration actually different as the polyedrisation proposed by [KIE96], border map presented here and the different levels of representations proposed by [ADFQ96].

Acknowledgements

We wish to thank Jean Françon to have initiate this work and for his relevant remarks. We are grateful to Ehoud Ahronovitz to have carefully check the final paper.

References

AAF95. Ehoud Ahronovitz, Jean-Pierre Aubert, and Christophe Fiorio. The star-topology: a topology for image analysis. In 5^{th} *Discrete Geometry for Computer Imagery, Proceedings*, pages 107–116. Groupe GDR PRC/AMI du CNRS, september 1995.

ADFQ96. R. Alaya., E. Dominguez., A.R. Frances, and A. Quintero. Determining the components of the complement of a digital $(n-1)$-manifold in z_n. In S. Miguet, A. Montanvert, and S. Ubéda, editors, *6th International Workshop, DGCI'96*, number 1176 in Lecture Notes in Computer Sciences, pages 163–174, Lyon, France, November 1996.

Bru96. L. Brun. *Segmentation d'images couleur à base Topologique.* Thèse de doctorat, Université Bordeaux I, décembre 1996.

Cor75. R. Cori. Un code pour les graphes planaires et ses applications. In *Astérisque*, volume 27. Soc. Math. de France, Paris, France, 1975.

FG96. C. Fiorio and J. Gustedt. Two linear time Union-Find strategies for image processing. *Theoretical Computer Science*, 154:165–181, 1996.

Fio95. Christophe Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation.* PhD thesis, Université Montpellier II, 24 novembre 1995.

Fio96. C. Fiorio. A topologically consistent representation for image analysis: the frontiers topological graph. In S. Ubeda S. Miguet, A. Montanvert, editor, *6th International Workshop, DGCI'96*, number 1176 in Lecture Notes in Computer Sciences, pages 151–162, Lyon, France, November 1996.

Fra95. J. Franşon. Discrete combinatorial surface. *Graphical Models and Image Processing*, 57(1):20–26, january 1995.

Fra96. J. Françon. On recent trends in discrete geometry in computer science. In S. Ubeda S. Miguet, A. Montanvert, editor, *6th International Workshop, DGCI'96*, number 1176 in Lecture Notes in Computer Sciences, pages 3–16, Lyon, France, November 1996.

Jac70. A. Jacques. Constellations et graphes topologiques. In *Combinatorial Theory and Applications*, pages 657–673, Budapest, 1970.

KIE96. Y. Kenmochi, A. Imiya, and N. Ezquerra. Polyedra generation from lattice points. In S. Ubeda S. Miguet, A. Montanvert, editor, *6th International Workshop, DGCI'96*, number 1176, pages 127–138, Lyon, France, November 1996.

KKM90. E. Khalimsky, R. Kopperman, and P.R. Meyer. Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications*, 36:1–17, 1990.

KM95. W.G. Kropatsch and H. Macho. Finding the structure of connected components using dual irregular pyramids. In 5^{th} *Discrete Geometry for Computer Imagery, Proceedings*, pages 147–158, *invited lecture*. Groupe GDR PRC/AMI du CNRS, September 1995.

Kov89. V.A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing.*, 46:141–161, 1989.

KR89. T.Y. Kong and A Rosenfeld. Digital topology: introduction and survey. *Computer Vision, Graphics, and Image Processing.*, 48:357–393, 1989.

Lie89. P. Lienhardt. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In 5^{th} *Annual ACM Symposium on Computational Geometry*, pages 228–236, Saarbrücken, Germany, 1989.

Lie97. Pascal Lienhardt. Aspects in topology-based geometric modeling. In E. Ahronovitz and C. Fiorio, editors, *Discrete Geometry for Computer Imagery*, volume 1347 of *Lecture Notes in Computer Science*, pages 33–48. Springer, December 1997. (invited speaker).

Lie91. P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *C.A.D*, 23(1), 91.

LL95. V. Lang and P. Lienhardt. Geometric modeling with simplicial sets. In *Proc. Pacific Graphics 95*, Séoul, Corée, 1995.

May67. J.P. May. *Simplicial objects in algebraic topology.* Van Nostrand Mathematical Studies, 1967.

Pen97. Y. Pennaneach. Calcul d'une carte des contours d'une image 3d. Master's thesis, Université Louis-Pasteur, Strasbourg, 1997.

Ros74. A. Rosenfeld. Adjacency in digital pictures. *Inform. and Control*, 26:24–33, 1974.