Topological Model for Two-Dimensional Image Representation: Definition and Optimal Extraction Algorithm

Guillaume Damiand, Yves Bertrand, Christophe Fiorio

Topological Model for 2D Image Representation: Definition and Optimal Extraction Algorithm

Guillaume Damiand [a,*] Yves Bertrand [a] Christophe Fiorio [b]

[a] *IRCOM-SIC, bât SP2MI, BP 30179, 86962 Chasseneuil Cedex, France*

[b] *LIRMM, 161 rue Ada, 34392 Montpellier Cedex 5, France*

Abstract

In this paper, we define the two dimensional *topological map*, a model which represents both topological and geometrical information of a two dimensional labeled image. Since this model is minimal, complete and unique, we can use it to define efficient image processing algorithms. The topological map is the last level of a map hierarchy. Each level represents the region boundaries of the image and is defined from the previous one in the hierarchy, thus giving a simple constructive definition. This model is similar to two existing structures but the main innovation of our approach is the progressive definition based on the successive map levels. These different maps can easily be extended in order to define the topological map in any dimension. Furthermore we provide an optimal extraction algorithm which extracts the different maps of the hierarchy in a single image scan. This algorithm is based on local configurations called precodes. Due to our constructive definition, different configurations are factorized which simplifies the implementation.

Key words: topological model, image representation, interpixel boundaries, combinatorial map, segmentation

1 Introduction

In this paper we present a combinatorial structure which represents interpixel boundaries of a two dimensional labeled image: the *topological map*. First we

* Corresponding author.

Email addresses: damiand@sic.univ-poitiers.fr (Guillaume Damiand), bertrand@sic.univ-poitiers.fr (Yves Bertrand), fiorio@lirmm.fr (Christophe Fiorio).

Article published in Computer Vision and Image Understanding-93(2)-pp. 111-154

give a formal definition of the topological map, then we propose an optimal algorithm which builds this structure from a labeled image by a single image scan and a minimal number of operations. The problem of the definition of such a structure has been widely studied [36,29,22,12,35,7,9,5]. Indeed, it allows an efficient implementation of the main geometrical and topological operations used in principal image processing algorithms and more particularly in image segmentation.

There are many different data structures which represent the region boundaries of a labeled image. These structures are all more or less derived from the Region Adjacency Graph (RAG) [36]. This graph is composed of a set of vertices, one by region of the image, and there is an edge between two vertices if and only if the two corresponding regions are adjacent. However, the RAG has several drawbacks: it does not represent multiple adjacencies, does not make the difference between adjacency and inclusion, does not keep the order of the edges around a vertex, does not represent the faces but only vertices and edges and last it is not unique, i.e. two different images could be represented by the same RAG [28]. To solve these problems, the RAG model has been extended.

For example [29,31] define dual graphs structure, which are two multi-graphs representing inclusion relations. The first graph is equivalent to the RAG, but with multi-edges in order to represent multi-adjacency. The second graph is the dual of the first one. In order to avoid disconnection, edges are added between distinct boundaries of the same region in the primal graph. These special edges become loops in the dual graph. This allows the differentiation of the inclusion relation and the adjacency relation. But dual graphs are not topological representatives of the images: we can have two topological different images having the same dual graphs. Another drawback of this structure is that each operation has to be applied twice (once to the primal graph and another to the dual one) in order to maintain the correspondence between the two graphs. At last, this structure is difficult to extend in higher dimension.

Two similar approaches have been proposed in order to give a solution to the problem of defining a structure representing all information resulting from a region segmentation. These two approaches are defined in the works of J.P. Domenger et Al. [19,11,12,7,10] and C. Fiorio et Al. [21,22,1]. The basic idea is to use combinatorial maps as a basis to represent the topology of the image. Indeed, combinatorial maps are a good model of space subdivision representation. They are defined in any dimension and represent all the cells of the subdivision and all the adjacency relations. Moreover, they can easily be linked to a geometrical model in order to represent the object geometry. Last, they are an efficient model to retrieve and to update information contained in the image [33].

The main difference between the two approaches is the algorithm which builds the topological map for J.P. Domenger and the topological graph of frontiers for C. Fiorio. The first one uses a contour tracking algorithm, the second one uses the notion of precodes (Sec. 7, see also [13,21]) allowing to compute the structure in a single image scan. We can also quote the Frontier Graph defined by J.M. Jolion and J.G. Palloncy [35] which is very similar to the two previous structures. These structures have been defined in 2D. The need to work with images of higher dimension and specially in 3D has led to study how to extend previous works to higher dimension. But we rapidly reach the conclusion that structures are strongly related to the dimension two and so prevent their direct extension to the nD case.

To solve this problem, [5] proposes the definition of a new structure in nD: the *border map*. This structure, also derived from combinatorial maps, represents the interpixel boundaries of a labeled image. But its major drawback is that this structure is not stable according to geometric transformations (translation, rotation, homothety, local modifications): border maps of two isomorphic partitions for these transformations can be completely different. This is a major drawback since the structure does not characterize images. This implies problems for matching algorithms and shows that the border map model does not provide a minimal encoding.

More recent works [9,8] extend the topological map in 3D. But the contour tracking algorithm which allows to construct the topological map in 2D is difficult to extend in 3D. They need to avoid particular configurations of voxels in the image: when two voxels with same label are adjacent by an edge. So before extracting the topological map, they first modify the image in order to remove all these configurations. This is an heavy time-consuming pre-processing which is not satisfying since it modifies the initial image.

To propose a solution, we reexamine the 2D topological map in order to give a formal definition which easily allows to extend it to higher dimension. This model has to be:

- *minimal* in the number of cells, to optimize memory occupation but also complexity of algorithms that have a direct access to the adjacency information;
- *complete*: it must represent both topology and geometry of images;
- *unique*: it must be invariant for geometric transformations: two topological equivalent images must have the same representation in order to facilitate matching algorithms.

To propose a new definition, we introduce the new notion of *simplification levels* which allows us to give a simple and constructive definition of the 2D topological map based on the removal operations. Moreover, due to the gener-

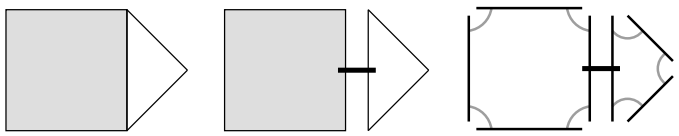
icity of the constructive definition and the removal operations, the extension to the 3D case is facilitated [2,3].

First we give in Sec. 2 a brief recall on combinatorial maps which are the basic model used in this work. Then we give in Sec. 3 some notations and some recalls on the interpixel notion. We introduce in Sec. 4 our new notion of simplification level, and we define the topological map by using several intermediary levels. This section presents only the topological part of our model in order to simplify the understanding. The geometrical part and the link between the two models is the object of Sec. 5. Then we present a first extraction algorithm in Sec. 6 which comes directly from the constructive definition. In Sec. 7 we give the optimal extraction which uses the precode notion. Precodes were already used in previous works [13,21] (only in 2D), but we show here a parallel between precodes to consider and the level we want to extract. We study all the different cases to process in order to extract any map level in a single image scan, and we show that some configurations can be factorized. In Sec. 8 we give our algorithm that computes the inclusion tree. In Sec. 9, we present more precisely the two similar existing approaches in order to understand the main differences and to see why our solution can be easier be extended in higher dimension. We present some experimental results in Sec. 10. Finally we conclude this paper in Sec. 11 and give some perspectives for future works.

2 Combinatorial Maps

The combinatorial map is a mathematical model of space subdivision representation based on planar map [20,37,24,14,15]. The subdivision of a 2D topological space is a partition of the space into 3 subsets whose elements are *cells* of 0, 1 and 2 dimension (respectively called vertices, edges and faces, and noted i-cell for a i-dimensional cell). Border relations are defined between these cells, where the border of a i-cell is a set of $(j<i)$-cells. We say that two cells are *incident* when one belongs to the border of the second, and that two i-cells are *adjacent* if they are both incident to the same $(j<i)$-cell. Combinatorial maps encode all the subdivisions and the incidence relations between all the different cells of the space, and so represent the topology of this space. They are defined formally for any dimension, and we call *n-map* an n-dimensional combinatorial map. The n-maps can encode all orientable manifold subdivisions of an n-dimensional space without boundary. They were generalized in [32,34] in order to encode all n-dimensional, orientable or not and with or without boundary subdivisions (see [33] to find a connection between maps and several other models).

A combinatorial map can be obtained intuitively by successive decompositions



(a) (b) (c)

Fig. 1. The successive decompositions of an object to obtain the corresponding 2-map. (a) A 2D object. (b) Disjoined faces. (c) Disjoined edges.

as we can see in Fig. 1. To represent the 2D object shown in Fig. 1(a), we first distinguish the faces of this object (Fig. 1(b)) then the edges of these faces (Fig. 1(c)). Adjacency relation between the two faces is represented by the black segment and adjacency relations between each couple of edges is represented by grey arc circles. The obtained elements after all the decompositions are called *darts* and are the single basic elements of the combinatorial map definition. To obtain the map, we only report each adjacency relation onto darts, and call β_i the relation between two i-dimensional elements. Let us see now the formal definition of 2D combinatorial maps that we can find for example in [33]:

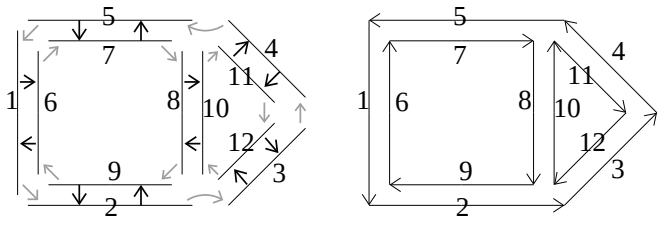
Definition 1 (2D combinatorial map) *A 2D combinatorial map, (or 2-map) is a triplet* $M = (D, \beta_1, \beta_2)$ *where:*

(1) D *is a finite set of darts;*
(2) β_1 *is a* permutation [1] *on* D*;*
(3) β_2 *is an* involution [2] *on* D*.*

When two darts d_1 and d_2 are such that $\beta_i(d_1) = d_2$ $(1 \leq i \leq n)$, we say that d_1 is i-sewn to d_2. We speak about i-sewing (resp. i-unsewing) for the operation that puts two darts in relation for β_i (resp. that removes an existing β_i relation). We note β_0 for β_1^{-1}, and β_{ji} for $\beta_i \circ \beta_j$ ($\beta_i \circ \beta_j(d) = \beta_i(\beta_j(d))$, we first apply β_j then β_i, the permutations are applied in the same order as they are read in the notation β_{ji}).

We can see in Fig. 2(a) the 2-map which represents the object shown in Fig. 1. The β_1 relation connects an edge and the following edge in the same face, and the β_2 relation connects the two faces incident to the same edge. In order to simplify the figures, we use the graphical representation presented in Fig. 2(b) where the β_i are not explicitly drawn. Each dart is represented by an arrow that shows the face orientation. With this orientation, we can retrieve, for each dart, the following dart in the same face and so deduce the β_1 permutation. Moreover, two darts 2-sewn are drawn near and parallel and the β_2

[1] A *permutation* on a set S is a one to one mapping from S onto S.
[2] An *involution* f on a set S is a one to one mapping from S onto S such that $f = f^{-1}$.



	1	2	3	4	5	6	7	8	9	10	11	12
β_1	2	3	4	5	1	7	8	9	6	11	12	10
β_2	6	9	12	11	7	1	5	10	2	8	4	3

Fig. 2. Two different representations of the same combinatorial map. (a) Full representation. Darts are represented by numbered black segments, β_1 relations are represented by grey arrows and β_2 by thick black arrows. (b) Implicit representation where the β_i are not explicitly drawn. Darts are represented by numbered black arrows (to represent the orientation). Two darts 1-sewn are drawn consecutively, and two darts 2-sewn are concurrently drawn and in reverse orientation.

permutation can also be deduced from the graphical representation.

Within the combinatorial map framework, all space cells are represented implicitly using the notion of *orbit*:

Definition 2 (orbit) *Let* $\Phi = \{f_1, \ldots, f_k\}$ *some permutations on* D. *We note* $< \Phi >$ *the permutation group generated by* Φ. *This is the set of permutations obtained by any composition and inversion of permutations contained in* Φ. *The* orbit *of a dart* d *relatively to* Φ *is defined by* $< \Phi > (d) = \{\phi(d) | \phi \in \Phi\}$.

Intuitively, an orbit $< f_1, \ldots, f_k > (d)$ is the set of darts that we can reach with a traversal starting with d and using all combinations of all the f_i or f_i^{-1} permutations. Given a 2-map and a dart d, we can retrieve all the cells incident to d by using particular orbits. The vertex incident to d is defined by $< \beta_{21} > (d)$, the edge by $< \beta_2 > (d)$ and the face by $< \beta_1 > (d)$. A dart is said to be incident to a cell if it belongs to the set of darts that represents the cell. Moreover, two cells are incident if the two corresponding sets have a non-empty intersection. In the 2-map shown in Fig. 2, the vertex incident to the dart *8* is the orbit $< \beta_{21} > (8) = \{5, 8, 11\}$, the edge incident to the same dart is $< \beta_2 > (8) = \{8, 10\}$ and the face is $< \beta_1 > (8) = \{6, 7, 8, 9\}$.

Combinatorial maps only represent object topology, and not their geometry. But it is easy to add some geometric elements to some (even all) orbits of

the combinatorial map: this operation is called *embedding* [3] . The separation of topological and geometrical models is one of the main advantages of combinatorial maps. Indeed, such a separation allows us to classify each operation in three different categories:

- Topological operations, which exclusively modify the topological model so the combinatorial map;
- Geometrical operations, which exclusively modify the geometrical model, the combinatorial map remaining unchanged;
- Mixed operations, which modify both models. But even in this case, we can usually decompose this operation in two distinct steps: first the topological modifications then the geometrical ones.

The separation of these two models allows us to modify either the embedding only or the topological model only. Moreover, operations become easier to define when the modifications of the topological and the geometrical model are separated.

3 Using Combinatorial Maps for Image Representation

Let us here recall some usual notations. A pixel is a point of discrete plan $\mathbb{Z}^2$ associated with a value which could be a color, a grey level.... A two dimensional image is a finite set of pixels. We use the classical notion of 4-connectivity because combinatorial maps can not represent non-manifold and so the 8-connectivity can not be considered without some tricks. In this work, we use combinatorial maps to represent pixel sets of *labeled images* having same values and which are 4-connected.

Definition 3 (labeled image) *A* labeled image *is a set of labeled pixels such as two pixels with the same label belong to the same 4-connected component.*

Note that we can represent any type of image simply by labeling all the 4-connected pixel sets that have the same value, for example with a growing algorithm. Considering only labeled images is an optimization that allows us to immediately retrieve, given the label of a pixel, the connected component its belongs to. But this work can be extended in order to consider any types of images. We speak about *region* for a set of same labeled pixels. Two pixels with same label belong to the same region, and two different regions have two different labels. We use this property only for the inclusion tree (cf. Sec. 8) in order to use the label of regions as a unique identifier, and so it is possible to

[3] One embedding example of a combinatorial map consists in linking to each topological vertex of the map the coordinates of an Euclidean space point.

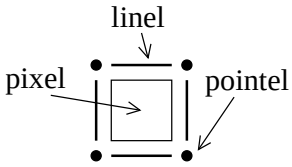


Fig. 3. All the cells of the interpixel 2-dimensional space.

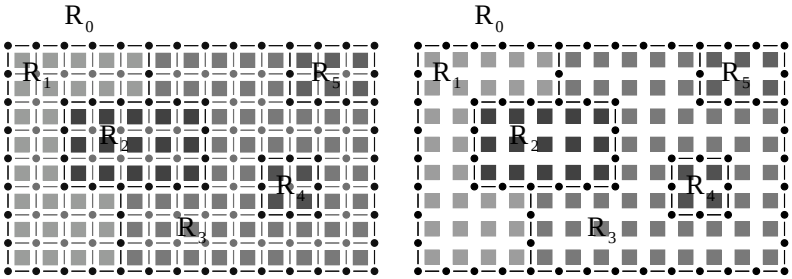
extend this work to consider regions made of several 4-connected components. To avoid particular processes of the image border pixels, we consider an infinite region R_0 that contains all the pixels that do not belong to the image. With this region, each pixel has exactly 4 neighbors (for the 4-adjacency). Moreover this infinite region allows us to process any type of image, not only the ones rectangular and without holes.

We say that a region R_i is *included* into a region R_j if and only if any 4-connected path going from a pixel of R_i to a pixel of R_0 (the infinite region) has at least one pixel belonging to region R_j. Intuitively, this inclusion notion corresponds to the fact that a region is around another one without any constraint on the way this surrounding is done. We can notice that each region is at least included into the infinite region, and that this relation is a partial order relation.

Combinatorial maps represent the borders of the regions contained in the image. Several works have been done on the notion of boundary in a discrete image and have shown that using a topology based on the *interpixel* notion [28,25,26,23,21] enables to define these borders so that they verify classical topological properties, as in the Jordan theorem [27].

In the interpixel framework, an image is not considered only as a matrix of pixels, but as a subdivision of a 2-dimensional space in a set of cells: *pixels* (the 2-cells called sometimes *faces*), *linels*, the 1-cells in between two 2-cells (also called *cracks*), and *pointels*, the 0-cells in between 1-cells (or *points*). These different types of cells can be seen in Fig. 3. With these cells, the notion of curve is topologically correctly defined: a *curve* is a list of alternated pointels and linels (with a pointel on each end) such as two consecutive cells in the list are incident. A curve is said *simple* when all the cells of the curve are distinct (a curve is simple if it is not self-intersected), and *closed* if the first cell is the same as the last one.

The *interpixel boundary* between two regions R_i and R_j is the set of simple curves two by two disjoined, such as each linel of these curves is incident to exactly one pixel of R_i and one pixel of R_j. We call *boundary curve* a curve belonging to one of the image boundaries. These boundary curves are maximal: each linel of the image incident to one pixel of R_i and one pixel of R_j belongs to one boundary curve, and two adjacent linels belonging to the same boundary belongs to the same boundary curve (they are separated in



(a) (b)

Fig. 4. Interpixel and Boundaries of a Labeled Image. (a) An image with all the interpixel elements. (b) Interpixel boundaries of the image regions.

this curve by the pointel incident to these two linels).

Figure 4 represents a labeled image and its interpixel boundaries. The boundary between regions R_1 and R_3 is composed of two distinct curves, the one between R_3 and R_4 is composed of one closed curve, and the boundary between R_1 and R_4 is empty since these two regions are not adjacent. Note that this definition of interpixel boundary is valid for any region, due to the existence of the infinite region.

4 Topological Map: Last level of Several Maps Definition

Combinatorial maps are well suited to represent interpixel boundaries since they represent space subdivisions and all the incidence relations between the different cells. For this reason they were used in several previous works [22,12,35,7,9,5]. But different combinatorial maps can represent the same image, and these works have used different map variants according to their needs.

In this paper we introduce the notion of *simplification levels* which allows us to give a simple and constructive definition of the 2D topological map. These levels can be formally defined with simple constructive definitions, and can be extended in higher dimension. The topological map is the last simplification level since it is minimal; other levels are only intermediate concepts. But it is possible to use an intermediate level in a particular application where we have no memory constraint (for example [4] uses a map equivalent to our level 2 in a 3D-modeler). First, we only present the topological part of this work without looking at the link with a geometrical model. It allows us to help with the understanding of our model and not to mix the two parts up. The link with a geometrical model is the purpose of Sec. 5.

The main idea of our approach is first to build a complete combinatorial map, that represents all the interpixel cells of the image, and then to progressively

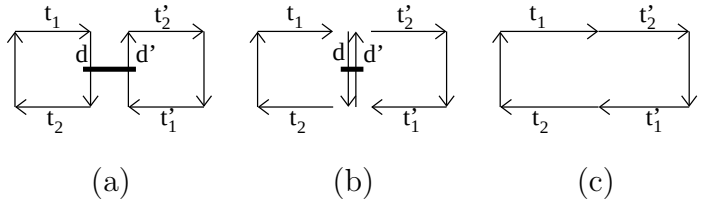


Fig. 5. 1-removal of the edge incident to the dart *d*. (a) Initial configuration. (b) 0 and 1-unsewing of *d* and *d'*. (c) 1-sewing of t_1 with t'_2 and t'_1 with t_2.

simplify it as long as no topological information is lost. This construction scheme allows us, in the end, to define the minimal map that represents the interpixel boundaries. At every step of the process, we only need to verify that there is no topological information loss in order to ensure the validity of the new map. To perform the successive simplifications, we use the *removal operations*.

4.1 Removal Operations in Map

The i-dimensional removal operation (noted i-removal) consists in removing a i-cell. This leads to the merging of the *(i+1)*-cells incident to the removed cell. In 2D, we can remove an edge or 1-cell (1-removal, see an example in Fig. 5) or remove a vertex or 0-cell (0-removal, see an example in Fig. 6). We only present here the main notions of these operations. A more complete description can be found in [17] where we give general definitions of removal and contraction [4] operations.

We can remove any edge in a 2-map without constraint (see example in Fig. 5 where we remove the edge incident to dart d). In order to perform this operation, we first 0 and 1-unsew the edge incident to the dart d (d and *d'* in Fig. 5(b)). Then we sew again "correctly" the darts which were previously sewn to this edge (t_1 with t'_2 and t'_1 with t_2, see Fig. 5(c)) and remove the free darts d and *d'*. We can prove this operation is correct whatever the initial configuration and the edge to remove (even in degenerated cases, as for example the removal of an isthmus, see [17]).

The 0-unsewing operation of a dart d consists only in removing the existing relation between d and $\beta_0(d)$. For that, we can use a particular dart (called for example NIL [5]) and define the $0-unsew(d)$ by the two affectations:

[4] Contraction is the dual operation of the removal. It consists in contracting a i-cell into a *(i-1)*-cell.

[5] If we want to formally define this particular value, we can modify the combinatorial map definition (Def. 1) by adding a particular dart called NIL such as $\forall i, \beta_i(NIL) = NIL$. This modified definition can also be used to represent objects with boundaries.

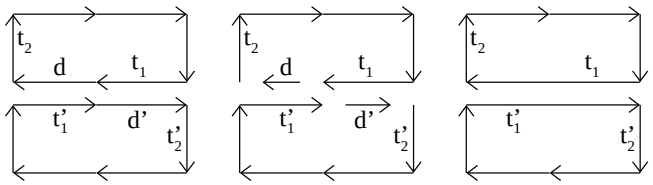


Fig. 6. 0-removal of the vertex incident to the dart d. (a) Initial configuration. (b) 0, 1 and 2-unsewing of d and d'. (c) 1-sewing of t_1 with t_2 and t'_1 with t'_2. 2-sewing of t_1 with t'_1.

$\beta_1(\beta_0(d)) = NIL$ and $\beta_0(d) = NIL$, the $1-unsew(d)$ by $\beta_0(\beta_1(d)) = NIL$ and $\beta_1(d) = NIL$ and the $2 - unsew(d)$ by $\beta_2(\beta_2(d)) = NIL$ and $\beta_2(d) = NIL$. We need to modify two values in order to preserve the properties of combinatorial maps ($\beta_0 = \beta_1^{-1}$ and β_2 is an involution). Using the 0-unsewing to define the 0-removal allows us to be sure that we always preserve these properties. Indeed, without unsewing, we can for example 1-sew a dart d, already 1-sewn to d_2, to another dart d_3 and thus obtain an object which is no longer a combinatorial map since $\beta_0(d_2)$ is equal to d and $\beta_0(d_3)$ too. Moreover, 0-unsewing is also useful to update eventual embeddings (see Sec. 5).

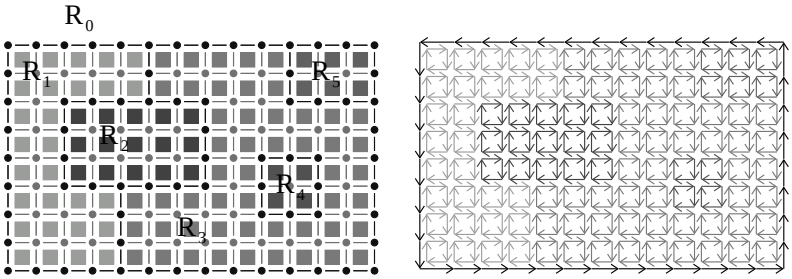
The vertex removal is possible only for degree two vertices (the degree of a i-cell c is the number of distinct *(i+1)*-cells incident to c). Otherwise, it is not possible to automatically decide how to connect the different edges around the removed vertex. We can see an example in Fig. 6 where we remove the vertex incident to dart d. The operation is performed with a similar algorithm to the edge removal. First we unsew the darts incident to the vertex (d and d' in Fig. 6(b)) then we sew the free darts (t_1 with t_2 and t'_1 with t'_2, see Fig. 6(c)) and remove free darts d and d' [6].

4.2 Level 0: Complete Map

Level 0 map is the starting point of our process and represents all the interpixel cells of a labeled image.

Definition 4 (level 0 map) *Level 0 map corresponding to an $n_1 \times n_2$ pixels labeled image, is the map having $n_1 \times n_2$ square faces 2-sewn between them,*

[6] Vertex removal can be achieved with edge contraction. Indeed, in the example shown in Fig. 6, removing the vertex incident to d is equivalent to contracting the edge incident to d (and also equivalent to contracting the edge incident to d'). The difference between vertex removal and edge contraction is the precondition of the operation: edge contraction can be achieved without constraint, and vertex removal can be done only for degree two vertices.



(a) (b)

Fig. 7. (a) An image and its interpixel elements. (b) The corresponding level 0 map.

each face corresponding to a pixel, plus an enclosing face which represents the infinite region.

Figure 7(b) shows the level 0 map of the image shown in Fig. 7(a). For an $n_1 \times n_2$ image, this map is composed of $(n_1 \times n_2)+1$ faces. $n_1 \times n_2$ square faces, each one representing a pixel of the image, made of 4 darts, and an additional face that represents the infinite region, made of $2 \times (n_1 + n_2)$ darts. Two darts are 1-sewn when they represent two consecutive linels of the same face, and are 2-sewn when they represent the same linel.

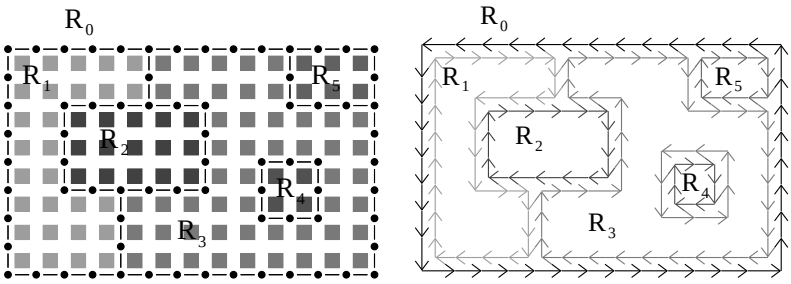
4.3 Level 1: Linel Map

Level 0 map represents all the interpixel cells of an image. In order to obtain a combinatorial map which represents only the interpixel boundaries, it is necessary to remove all linels corresponding to inner boundaries. This operation is done with the edge removal.

Definition 5 (level 1 map) *Level 1 map is the map obtained from level 0 map by removing each edge between two pixels having the same label.*

We can see in Fig. 8(a) the image of our example and in Fig. 8(b) the corresponding level 1 map that represents all the interpixel boundaries of the image. Each edge of this map corresponds exactly to one linel of a boundary curve.

It should be noticed that each region is represented in level 1 map by an external border and zero or several internal borders. A region R has internal borders when there are some regions included into R. In our example in Fig. 8(b), only region R_3 has one internal border (without considering the infinite region that is always represented by an internal border). The map is disconnected into several connected components (in the example, two connected components), and we have lost the topological information necessary to place and link the connected components. Without this information, it is not possible to dis-



(a) (b)

Fig. 8. (a) Interpixel boundaries. (b) The corresponding level 1 map.

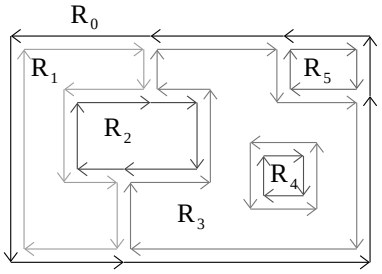


Fig. 9. Level 2 map.

tinguish two non-adjacent regions and one region included into another one. In order to keep this information, we introduce a region inclusion tree of the (more precisely described in Sec. 8). This tree has one node for each image region of the. Its root is the infinite region, and the set of regions included in a given region R defines the sons of R in the tree. This tree allows us to retrieve the inclusion information and so to retrieve all the boundaries of a given region.

This level 1 is the first map that represents interpixel boundaries of a labeled image. But it is clear that this structure is not optimal, neither for space, nor for time complexity. Indeed, due to the number of darts having to be traverse to retrieve adjacency information, operations on this map require heavy computations. This is why we simplify this map and define level 2.

4.4 Level 2: Border Map

Level 2 map is the most intuitive map encoding interpixel boundaries of an image. Indeed, each edge of this map represents a straight part of a boundary (this map is called *border map* in [5]). We can see level 2 map of our image in Fig. 9.

Definition 6 (level 2 map) *Level 2 map is the map obtained from level 1 map by removing each degree two vertex between two aligned edges.*

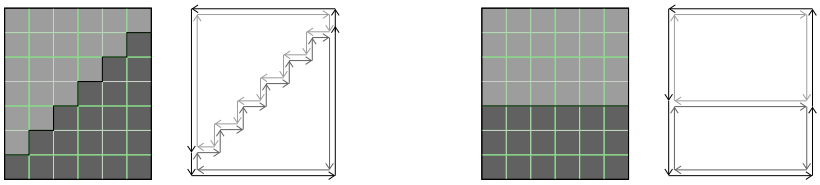


Fig. 10. Two topological equivalent partitions and their corresponding level 2 maps.

Each edge of level 2 map corresponds to a maximal series of aligned linels of a boundary curve. But these edges are now of any length, contrarily to level 1 map where edges are of length one. We can prove that level 2 map is topologically equivalent to level 1 map. The only modification achieved on the level 1 map is the removal of each degree two vertex between two aligned edges. Each couple of aligned edges incident to a degree two vertex is necessary between the same two regions R_i and R_j. So the two edges represent the same topological adjacency information and deleting one of the two does not lead to any loss of topological information. Moreover, the vertex removal can not lead to the disconnection of the map in several connected components.

The main problem of level 2 map is its non-unicity. Indeed, the number of darts used to represent a boundary curve depends on the geometry of this boundary. In the worst case, when a boundary does not have any consecutive aligned linel (cf. the stair case in Fig. 10), the number of darts of level 2 map is exactly twice the number of linels of the boundary. This non-unicity implies that two topologically equivalent partitions can have two completely different level 2 maps.

4.5 Level 3: Topological Map

To solve the non-unicity problem of level 2 map, we need to perform the same simplifications achieved to define level 2 map but for non-aligned edges. Indeed, after these simplifications, we obtain a model which do not depend on the geometry of the image boundaries but only on its topology. Level 3 map of our example is shown in Fig. 11.

Definition 7 (level 3 map) *Level 3 map is the map obtained from level 2 map by removing each degree two vertex.*

Each edge of level 3 map corresponds exactly to a boundary curve of the image. This gives the final model the name of "topological map": level 3 map plus the inclusion tree. This map is minimal, we can not do any additional removal without modifying the topology. To ensure that level 3 map contains the same topological information as level 2 map (and so as level 1 map), we use the same proof as the previous map and we show that two edges incident to a degree two vertex represent the same adjacency information.

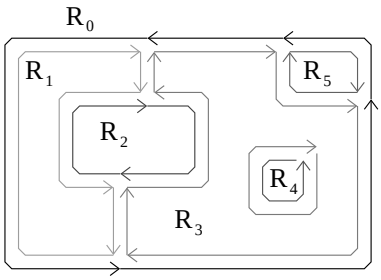


Fig. 11. Level 3 map.

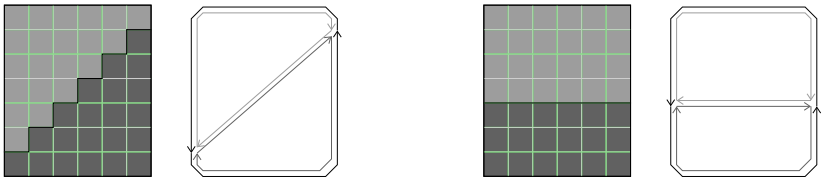


Fig. 12. Two topologically equivalent partitions and their corresponding level 3 maps.

Level 3 map only depends on the topology of the represented image. That is why two topologically equivalent partitions are represented by two isomorphic maps as we can see in Fig. 12. Level 3 map represents each boundary curve on a minimal way (one edge) and thus we can no longer simplify the map. This map needs much less memory than the others, it is consistent with the topology of the represented objects and it is stable for geometric transformations of the corresponding image. Moreover, image processing algorithms need less computational time when they are defined on level 3 map because of the direct access of the adjacency information.

5 What Geometrical Model for Each Map Level

Combinatorial maps only represent the topological part of images. But in most applications, it is necessary to represent also geometry. There are several ways to associate a geometrical model to a combinatorial map. We present here one solution for each map level and show how these models have to be modified during our main operations: sewings, unsewings and removals.

5.1 Level 1 and Level 2 Maps

For level 1 and level 2 maps, we associate a 2D geometrical point (given by its coordinates) to each topological vertex of the map. Each topological vertex is represented in the map by a set of darts, but only one of these darts is linked

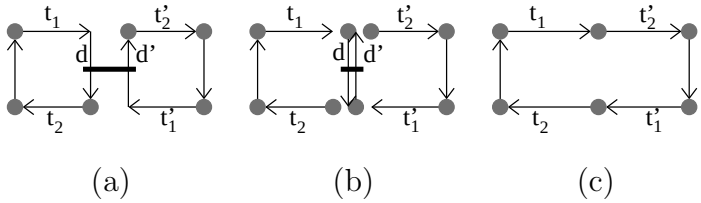


Fig. 13. 1-removal of the edge incident to the dart d with embedding modification. (a) Initial configuration. (b) 0 and 1-unsewing of d and d'. The two vertices incident to d and d' are duplicated. (c) 1-sewing of free darts: t_1 with t'_2 and t'_1 with t_2. Geometrical points linked with d and d' are removed.

with the geometrical point. Given a dart, we have to traverse the vertex orbit in order to find this particular dart and so the geometrical point.

We use the same geometrical model for level 1 and level 2 maps in order to simplify algorithms. Indeed, we do not need a conversion of the geometrical model between these two levels. But this leads to a more important complexity in memory space for level 1 since we do not use its particular properties.

This embedding is simple since only one type of cell is embedded. Moreover, retrieving the embedding of the edges or the faces can easily be achieved since it is sufficient to traverse topological cells and to keep the coordinates of each vertex. To ensure the validity of this embedding, we must verify that each topological vertex has exactly one dart that is linked with a geometrical point.

This property can be preserved automatically by modifying the sewing and unsewing operations. Indeed, we test when a sewing operation groups together two initial distinct vertex orbits and then remove one [7] of the two geometrical points when it is the case. We also check when an unsewing operation breaks an initial topological vertex into two distinct orbits, and then we duplicate the geometrical point. We can see in Fig. 13 the 1-removal operation (already presented in the previous section) with the embedding modifications. After the 0 and 1-unsewing of d and d', the two incident geometrical points have been duplicated as the corresponding vertices have been broken into two orbits (Fig. 13(b)). Then, after the 1-sewing of free darts (Fig. 13(c)), the two geometrical points associated with d and d' are removed along with the deletion of the two darts.

Note that this is not the optimal embedding updating. Indeed, if we know performing a 1-removal operation (as the example shown in Fig. 13), we know that darts d and d' are going to be deleted and thus it is not required to duplicate the two geometrical points. We can sometimes choose to use the

[7] When we i-sew(a,b), we keep the geometrical point associated with a. For this reason, 0-sew(a,b) or 1-sew(b,a) give the same topological result but not the same geometrical one.

general sewing and unsewing operations that check and update the vertex embeddings, and some other times the basic sewing and unsewing operations (for example when we perform a set of sewing operations and know exactly how the different orbits are going to be modified).

But this is only an optimization and in the following we do not detail when we can use basic operations instead of general ones. Moreover, the additional cost in complexity of general sewing and unsewing operations is not important. Indeed in the worst case there are 4 darts incident to the same geometrical point (one dart by linel incidents to the corresponding pointel) and thus the test on the vertex orbit can be performed in 4 elementary operations that is to say in constant time.

5.2 Level 3 Map

For level 3 map it is not possible to use the same embedding as for the two other levels. Indeed, we can not retrieve the embedding of an edge with only the embedding of its end vertices because edges can now have any geometrical shape.

In order to embed this map, we associate a 2D geometrical point to each topological vertex (as for the previous levels), and we associate to each topological edge the 1D curve corresponding to the interpixel boundary without its two extremities (we call these curves *open curves*). These curves are represented by 1-dimensional combinatorial maps [8] . These 1-maps represent maximal straight line segments in order to minimize the number of elements. Only one dart of the two that compose an edge is linked to the 1-map, and the orientation of the 1-map is the same as the one of this dart.

The main advantage of this embedding is to not encode twice the coordinates of the vertices. The main drawback is that we do not have immediately the embedding of a boundary curve but we need to reconstruct it from one open curve and two geometrical points. But this can be performed in linear complexity for the number of points of the boundary curve.

We can see in Fig. 14 the embedding of level 3 map of our example image used in this paper. This map has seven topological vertices (embedded by geometrical points a, d, e, h, k, m and o) and ten topological edges. Among them, three do not have an edge embedding (edges $\{2, 8\}$, $\{4, 12\}$ and $\{9, 18\}$). Indeed, when a topological edge corresponds to a straight line segment, the

[8] A 1-map is equivalent to a double-linked list but this allows us to consider the extension in higher dimension where each i-dimensional topological cell could be embedded with a i-dimensional combinatorial map.

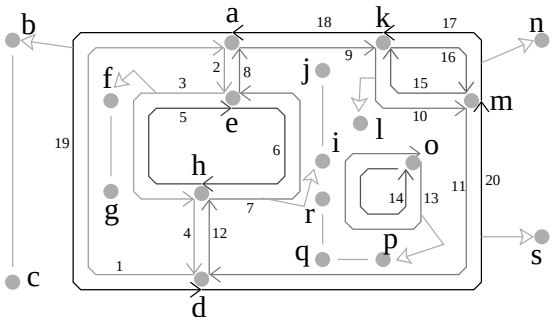


Fig. 14. Topological map with open curves and vertices embedding.

1D open curve is empty and the two geometrical end points are enough to reconstruct the edge geometry.

The method to reconstruct the embedding of an edge incident to a dart, for example dart numbered 19 in the figure, consists in retrieving in a first time the embedding of the vertex incident to 19 (geometrical point a), then in retrieving the embedding of the edge (geometrical points b,c), and at last in retrieving the embedding of the dart $\beta_2(19)$ (geometrical point d). The 1D curve made from the concatenation of these three embeddings is the embedding of the edge (list of geometrical points a,b,c,d).

During modification operations, we need to update both embeddings (vertex and edge embeddings). For the vertex embedding, we can exactly perform the same modifications of the sewing and unsewing operations as for level 1 and level 2 maps. The edge embedding has to be updated only for the 0-removal operation. Indeed, when we remove an edge, the corresponding open curve is simply destroyed. But when we remove a vertex, we need to keep the geometry of the two merged edges in the unique embedding of the resulting new edge.

In order to do that, we process in two steps (shown in Fig. 15). So as to remove the vertex incident to dart d, we first add at the beginning of the 1-map associated to d the geometrical point of the removed vertex (depending on the dart that is linked with the 1-map, this is equivalent to adding the geometrical point at the end of the 1-map associated to $\beta_2(d)$). Note that this creates such a map if the edge incidents to d does not already have an edge embedding (case of straight line segments).

In a second step we merge the edge embedding of $\beta_0(d)$ and the edge embedding of d (with a particular case to take into account when one edge does not have an edge embedding). When the two 1-maps do not have the same orientation, we need to reverse one of the two before merging them in a unique 1-map. Moreover, in order to represent maximal straight line segments, we check during this merging if the three geometrical points around the removed vertex are aligned, and when it is the case we remove the geometrical point inserted in the previous step.

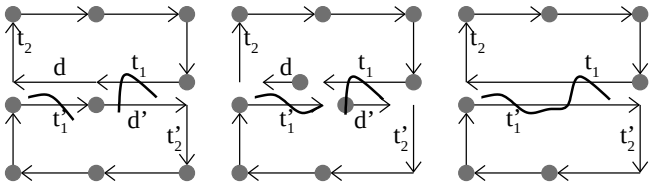


Fig. 15. 0-removal of the vertex incident to the dart d with embedding modification. (a) Initial configuration. (b) 0 and 1-unsewing of d and d'. The geometrical point incident to d is added in the beginning of the 1-map associated to the edge incident to d. (c) 1-sewing of free darts and merging of the two 1-maps associated with the edges incident to d and d'.

As for the embedding of level 1 and level 2 maps, the updating of the embedding can automatically be achieved by the sewing and unsewing operations and the 0-removal operation, but it can also be achieved manually for particular operations in order for example to optimize and avoid several tests of the same orbit. But in this last case, we must make sure that the embedding is correctly updated in order to avoid a loss of geometrical information.

6 Basic Generic Extraction Algorithm of Any Map Level

The successive definitions of each map level seen in Sec. 4 immediately give a first extraction algorithm presented in Algo. 1. The main advantage of this algorithm is its simplicity. Indeed, it follows the definitions of the different map levels. Moreover, it is generic: the same algorithm can extract any map level. First, this algorithm builds level 0 map of the labeled image (line 0). Then, depending on l, the level we want to extract, we progressively simplify the map by following the level definitions. Each line numbered i of the algorithm constructs level i map of the image. At last (line 4) we compute the inclusion tree with the algorithm presented in Sec. 8.

The embedding modifications are achieved during the removal operations by the way we have explained in the previous section. For level 1 and level 2 maps only for the vertex embedding, and for level 3 by adding edge embeddings (when we remove a vertex between two non-aligned edges).

This algorithm is linear in number of image pixels (both in time and memory). Indeed, level 0 map has a dart number which is linear in pixel number (see Sec. 4.2) and other levels have less (or equal) darts. In order to construct each level, we only test each cell exactly once. But in order to construct level 3 map, we traverse each dart three times, and even if the complexity is linear it is not optimal. Moreover, we begin for level 0 map by creating many darts that are going to be eventually destroyed during the simplifications. This leads

Algorithm 1: Basic extraction of level l map.

Input: A labeled image I of $n_1 \times n_2$ pixels.
l the level of the map we want to extract ($l \in \{1, 2, 3\}$).

Output: The level l map corresponding to I.

```
0 Build a 2-map M of n1 × n2 squares sewn between them
1 Remove each edge between two same labeled regions
  if l > 1 then
2 |  Remove each degree two vertex between two aligned edges
  |  if l> 2 then
3 |  |  Remove each degree two vertex
4 Compute the region inclusion tree
```

to an important and useless time consumption due to memory allocation and de-allocation. Moreover, memory space required for the construction of level 0 map can be too important, and can prevent the construction of other map levels. To solve these drawbacks, we now propose an improvement of this basic algorithm.

7 Precodes to Define an Optimal Extraction Algorithm

The basic principle of precodes [13,21,22] consists in scanning the image from top to bottom and from left to right, with a 2×2 pixels window, and in executing an algorithm which depends on the local configuration of pixels in this window. We have used this principle, adapted it to our different map levels, and precisely studied what configurations have to be treated for each level. The main idea of our approach is to compute the map by an incremental and direct way, with only one image scan. Furthermore, precode algorithms depend only on the local pixel configuration, and create only the necessary darts.

7.1 Precodes

We scan the image from top to bottom and from left to right by using a 2×2 pixels window. During this scan, the *current pixel* is the bottom right pixel of the 2×2 window. A *precode* is a local configuration of pixels.

Definition 8 (precode) *A* precode *is a partition of the set of the four pixels of the 2×2 window.*

A precode fixes the pixels that have the same label and those that have different labels. Given a partition $\{p_1, \ldots, p_k\}$ of the 2×2 window, two pixels

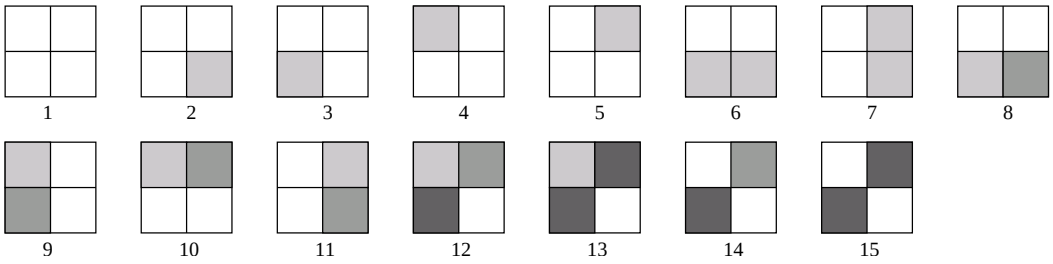


Fig. 16. The 15 different precodes.

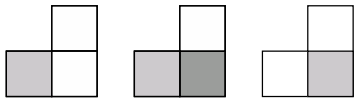


Fig. 17. Two partial precodes (a and b) and a counter example (c).

belonging to the same p_i have the same label, and reciprocally two pixels belonging to distinct p_i and p_j have different labels. There exist 15 different precodes in 2D shown in Fig. 16. We draw a precode by affecting a different color to each partition element. Precodes number 13, 14 and 15 in Fig. 16 are *non-manifold* precodes.

Definition 9 (non-manifold precode) *A precode is called* non-manifold *if and only if it exists an element of the partition which is not 4-connected. A precode which is not a non-manifold precode is called* manifold.

The precode notion allows to define the operations to perform for each configuration met during the image scan. But we need another notion, which allows us to group several precodes in order to factorize similar processings: this is the *partial precode* notion.

Definition 10 (partial precode) *A* partial precode *is a partition of any subset of the four pixels of the 2×2 window, where each element of the partition is a 4-connected set of pixels.*

We can see in Fig. 17(a) and Fig. 17(b) two examples of partial precodes and a counter example in Fig. 17(c). This last partition is not a partial precode, since one element of this partition (white pixels) is not a 4-connected set of pixels.

Precodes and partial precodes do not have exactly the same interpretation. Given a partition $\{p_1, \ldots, p_k\}$ representing a partial precode, pixels not present in the partition can belong to any region. Two pixels belonging to the same p_i have the same label (as for a precode). But two pixels belonging to distinct p_i and p_j can have the same label or not, depending if $p_i \cup p_j$ is a 4-connected set or not:

- $p_i \cup p_j$ is 4-connected: the two pixels have the same label;
- $p_i \cup p_j$ is not 4-connected: the two pixels can have the same label or not.

Intuitively, a partial precode fixes the pixels that have the same label or those that have different labels, but only for 4-neighbor pixels. This is why a partial precode regroups several precodes (and also because pixels not present in the partition can belong to any region).

The partial precode shown in Fig. 17(b) only fixes that the bottom right pixel has a different label from its left pixel neighbor, and a different label from its top pixel neighbor. But it fixes no constraint between the top right pixel and the bottom left pixel, because the union of the two partition elements is not a 4-connected set. On the contrary, the two pixel sets of the partial precode shown in Fig. 17(a) can not belong to the same region, because their union is a 4-connected set.

A partial precode regroups the precodes possible to be obtained by labeling pixels not present in the partial precode, and by possibly grouping non 4-connected element partitions in a same element. For example, partial precode shown in Fig. 17(a) regroups precodes 3, 7 and 9, and partial precode Fig. 17(b) regroups precodes 2, 8, 11, 12, 13, 14 and 15.

7.2 Optimal and Generic Extraction Algorithm

The extraction algorithm based on the precode notion is presented in Algo. 2. It is generic since it can extract any map level, only by considering different precodes, and it is optimal since the map is extracted in a single image scan (we process each pixel exactly once) and the processings performed for each precode is done in a minimal number of operations. Moreover, this algorithm is simple because it can be summarized by an image scan where we execute the code corresponding to the current precode.

Algorithm 2: Optimal extraction of the level l map.

Input: A labeled image I of $n_1 \times n_2$ pixels.
l the level of the map we want to extract.

Output: The level l map corresponding to I.

```
1 last ← Build the upper left border of the image
2 for j=1 to n2+1 do
      for i=1 to n1+1 do
          last ← Process the code corresponding to precode (i,j)
3 Compute the inclusion tree
```

The image contains pixels having coordinates between $1 \dots n_1$ in x and $1 \dots n_2$

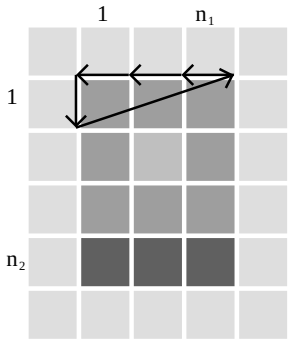


Fig. 18. The initial upper left border of an image with 3×4 pixels.

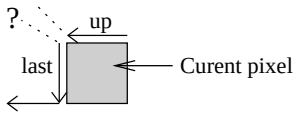


Fig. 19. Darts *last* and *up* by relation with the current pixel.

in y (other pixels are in the infinite region). Precode (i,j) is the precode having as current pixel (the bottom right pixel of the 2×2 window) pixel (i,j). During the image scan, we overflow the image by one pixel in any direction. Indeed, when we process precodes (1,j) (resp. (n_1+1,j), (i,1), (i,n_2+1)), that processes the pixels on the left (resp. right, up and bottom) of the image.

The first step of the optimal algorithm consists in building the upper left border of the image. Indeed, we have this invariant: the map corresponding to the pixels already scanned, is already built. This invariant allows us to be sure, at any time, that it exists a dart on the left and at the top of the current pixel. To satisfy this invariant for the first pixel of the image (pixel (1,1)), we begin by building the upper left border of the image, as we can see in Fig. 18. With this initial map, each pixel in the first line of the image has an upper dart, and the first pixel of this first line has also a left dart.

During the image scan, we need to know the darts on the left and at the top of the current pixel, in order to locally modify the map already built. We respectively call these darts *last* and *up* (or l and u in figures), as we can see in Fig. 19. In this figure, we don't know if the darts *last* and *up* are sewn or if another darts between these two darts exist (this is represented by the dash lines and the question mark in the figure). But this has no influence on our algorithms since the map is only locally modified. We only keep the *last* dart since the *up* dart can be computed from this dart, starting from the *last* dart, and turning over the vertex until we find a dart not 2-sewn (note that during the extraction, the map is not closed, the darts in the border of the map are not 2-sewn, but after the extraction we have a correct map where all darts are sewn).

The *last* dart is initialized with the unique vertical dart of the initial border. Then, during the extraction, we execute the algorithm corresponding to the current precode. Each precode algorithm locally modifies the map, and return the vertical dart, to the bottom right of the current precode, which is the *last* dart for the next precode. Note that we do not need particular processings for

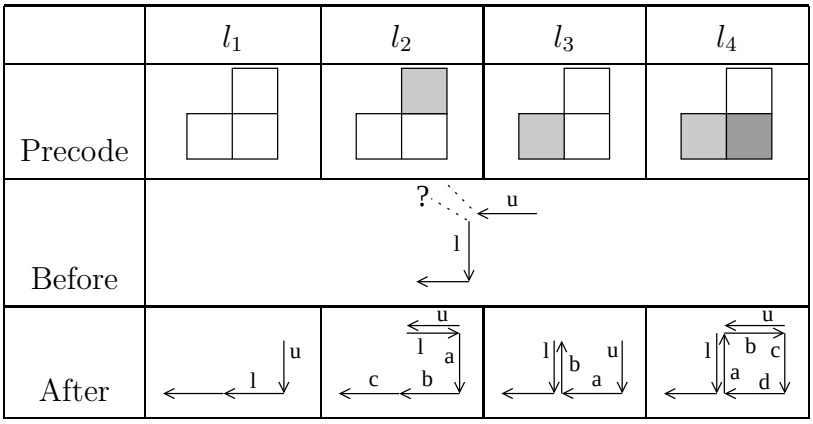


Fig. 20. The four precodes to extract level 1 map.

the border of the image since it is plated on a cylinder (the last pixel of each line is the same as the first pixel of the next line). At the end of the algorithm, we have to compute the inclusion tree, as for the basic algorithm, by using the algorithm given in Sec. 8.

7.3 Precodes for Level 1 Map

The optimal extraction is generic. Indeed, to extract a particular map level, we only need to define what precodes we have to consider, and give the operations to perform for each of them. To find the precodes to consider for a particular level, we have to look at the operations to be achieved to obtain this level and see the consequences of these operations on the local precodes. To extract level 1 map, we need to remove each edge of the complete map between two pixels having the same label.

Considering the current pixel and the map already built, we need to remove the left edge when the left pixel has the same label as the current pixel and we also need to remove the upper edge when the upper pixel has the same label as the current one. So there are four precodes to consider: they are all the ways of removing zero, one or two of these edges. These precodes are shown in Fig. 20.

In this figure, the first line gives the precode name, the second the precode itself, the third shows the current configuration of the map before executing the current precode algorithm, and the last line shows for each precode, the map we want to obtain after this execution. This map corresponds to the map obtained locally by the basic algorithm when we create the complete map and then we perform 1-removals. Maps represented in the figures are partial representation of the local configuration. For example, for the map after precode l_2 in Fig. 20, there are necessarily a dart 0-sewn to l, another 0-sewn and 1-sewn to u, but since this precode is local, we do not know where

these darts are and we can not draw them.

In order to write the algorithm corresponding to each precode, we only have to look at the map before, compare it with the map we want to obtain, and write the sequence of operations which transform the first map into the second. We can see in Algo. 3, which gives the algorithm of precode l_2, that this code is easy to write. Indeed it is mainly composed of successive unsewn and sewn operations. We can verify in Fig. 20, that this algorithm actually transforms the current map into the map we want to obtain for the corresponding precode.

Algorithm 3: Code corresponding to precode l_2.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow$ a new dart; $b \leftarrow$ a new dart
$c \leftarrow \beta_1(last)$; 1-unsew(last)
1-sew(last,a); 1-sew(a,b)
1-sew(b,c); 2-sew(up,last)
Embedding of the vertex incident to $b \leftarrow$ (x,y)
return *a*

The initial map is modified in order to include the current pixel. For that, we can modify all the darts of the initial map, even *last* and *up* darts. We only need at the end of the algorithm to return the dart which will be the next *last* dart for the next precode. The definition of the four precode algorithms (given in annex) totally defines the optimal extraction of level 1 map, which during the image scan, tests what is the current precode, and only executes the corresponding algorithm.

7.4 Precodes for Level 2 Map

To extract level 2 map, we have to remove each degree 2 vertex between two aligned edges. But since we construct the map in a single scan, we also need to remove each edge between two pixels having the same label (simplification to obtain level 1 map). It we look at the 15 different possible precodes, we can see that there are only two cases with aligned edges that are incident to a degree two vertex: the two precodes given in Fig. 21.

To extract level 2 map, we need to find what is the current precode between these two precodes plus the four level 1 precodes. We can note that these two level 2 precodes are included in precodes l_2 and l_3. To find what is the current precode, we first test if it is a level 2 precode (f_1 or f_2), and when it is not the case we search in the level 1 precodes. We can see in Fig. 21 these two

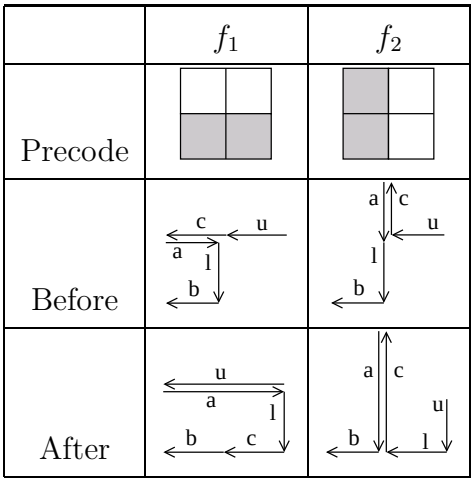


Fig. 21. The two additional precodes to extract level 2 map.

precodes, the initial maps and the maps we want to obtain. As for level 1 map, the algorithms corresponding to these two precodes are simple as we can see for precode f_1 in Algo. 4.

Algorithm 4: Code corresponding to precode f_1.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$
$c \leftarrow \beta_1(up)$; $d \leftarrow \beta_1(c)$
1-unsew(last); 2-unsew(a)
0-unsew(c); 1-unsew(c)
1-sew(last,c); 1-sew(c,b)
1-sew(up,d); 2-sew(up,a)
Embedding of the vertex incident to $c \leftarrow$ (x,y)
return *last*

7.5 Precodes for Level 3 Map

For this level, we have to remove each degree 2 vertex. By looking at the 15 precodes, we can note that there are only four cases where such a removal has to be achieved. These precodes are given in Fig. 22. We have not represented the edge embeddings in this figure, but they are updated during the sewing, unsewing and 0-removal operations, by the way explained in Sec. 5. We can see in Algo. 5 and Algo. 6 the two algorithms that correspond to precodes t_1 and t_3.

Algorithm 5 is similar to previous algorithms. The unique difference is the explicit embedding modification of the edge incident to *a*. Indeed, since we remove the vertex incident to dart *c*, we need to keep the corresponding ge-

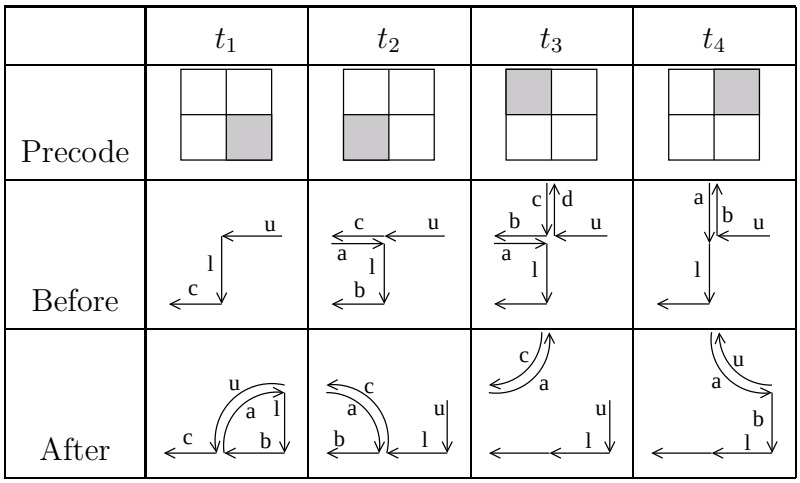


Fig. 22. The four additional precodes to extract level 3 map.

Algorithm 5: Code corresponding to precode t_1.

Input: *last* and *up*, (x,y) coordinates of the new vertex.

Output: The "next" *last*.

$a \leftarrow$ a new dart; $b \leftarrow$ a new dart
$c \leftarrow \beta_1(last)$; 2-sew(up,a)
Add geometrical point incident to *last* at the end of the edge embedding of a
0-unsew(last); 1-unsew(last)
1-sew(a,last); 1-sew(last,b)
1-sew(b,a); 0-sew(c,up)
Embedding of the vertex incident to $b \leftarrow$ (x,y)
return *last*

ometrical point in the edge embedding. Note that the edge incident to a has initially no embedding because only closed edges (for β_2) can have such an embedding. So when we add the geometrical point incident to *last* at the end of the edge embedding of a, this creates an edge embedding made of this unique geometrical point.

Algorithm 6: Code corresponding to precod t_3.

Input: *last* and *up*, (x,y) coordinates of the new vertex.

Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_2(a)$; $c \leftarrow \beta_0(b)$
$d \leftarrow \beta_2(c)$; $t_1 \leftarrow \beta_1(b)$; $t_2 \leftarrow \beta_1(d)$
Add geometrical point incident to *last* at the end of the edge embedding of a
Merge edge embedding of d at the end of edge embedding of a
0-unsew(b); 1-unsew(b); 2-unsew(b); delete b
0-unsew(d); 1-unsew(d); 2-unsew(d); delete d
0-unsew(last); 1-sew(up,last)
2-sew(a,c); 1-sew(a,t_2); 1-sew(c,t_1)
Embedding of the vertex incident to $last \leftarrow$ (x,y)
return *up*

The algorithm of precode t_3 is a little more complex since this is the unique case where the two edges along the removed vertex (edges incident to a and b) can both have an edge embedding. In order to keep the geometry, we first add the geometrical point incident to *last* at the end of the edge embedding of a, then we merge the edge embedding of d at the end of edge embedding of a. This merging has to consider the special case where one embedding is empty, and also the case where the two embeddings have not the same orientation. In this case, we need to reverse one of the two embeddings before the merging (see Sec. 5 for more explanations).

The definition of the four algorithms, corresponding to the four level 3 precodes, allows us to extract level 3 map in a single image scan, by the way of the generic extraction algorithm. First we check if the current precode is a level 3 precode, if not if it is a level 2 precode and otherwise we find the current precode in the level 1 precodes. Note that we only have ten cases to consider amongst the 15 possibles precodes. This is due to our simplification levels that allows us to factorize immediately similar processings.

8 Inclusion Tree

The computation of the inclusion tree is the last step of our two extraction algorithms, after the extraction of a level l map. This tree is necessary to keep a relation between each different connected components of the map. The inclusion relation used here is the one presented in Sec. 3, but we only represent direct inclusions (relations we can not retrieve by transitivity). The inclusion tree is always rooted with the infinite region. We keep relations between a node and its sons and with its father, in order to be able to go through the tree from root to leaves and from leaves to root.

There are many possibilities to represent inclusion relations: we can give for each region the list of regions adjacent to an internal border; we can add fictive edges between all the internal and external borders of each region; we can give for each region one region per connected set of included regions (one region for each hole)... Our choice allows us to obtain an inclusion tree and not a forest, where each region is present. This is interesting when we need to traverse each region: so it is enough to traverse the inclusion tree without considering the map. Otherwise, this solution is not a good one to easily check if two regions can be merged or not, only by looking at the inclusion tree. But of course, each solution has its own advantages and disadvantages, and our proposed solution need to be reconsidered for each application.

To compute the inclusion tree, we traverse all the darts of the map connected component by connected component. Indeed, a connected component is com-

posed of regions that all have the same father. In order to retrieve the different inclusions we need some particular information:

- Each dart is linked with its belonging region noted *region(d)*: $\forall$ *d* and *d'*, *region(d)=region(d')* if and only if *d* and *d'* belong to the same region;
- Each region R knows one dart of the map, belonging to this region. This dart has to be the dart with the corresponding vertex in the upper left of this region. This property ensures us we can easily retrieve the region containing R. We call such a dart the *representative* of its region;
- We also have to know the list of all the image regions, without the infinite region. This list must be sorted so that a region R_i is smaller than another region R_j if and only if R_i is met before R_j during the image traversal from top to bottom and from left to right.

These properties are given by our extraction algorithms. For the basic algorithm, we need to perform a first image scan in order to initialize the list of sorted regions. The dart labeling is achieved during the construction of level 0 map, since simplification operations do not lead to modification of a dart label. The representative dart of each region is fixed during the initial image scan, and possibly modified during the simplifications. Indeed, when the representative dart of a region is removed, we choose as new representative a neighboring dart of the former one. All these operations can be achieved in linear complexity, but lead nevertheless to additional time for the initial image scan.

For optimal algorithm, the list can be computed at the same time as the extraction since we use the same image scan. During this scan we can fix the representative dart of each region when we meet a region for the first time. The dart labeling is realized at the same time. When we meet a new region, we label the new darts with this new region (this case can occur only for precodes l_4 and t_1). For the other cases, we copy the label of the same region existing darts (we can verify that for each precode except l_4 and t_1, the current pixel has a neighbor with the same label).

The computation of the inclusion tree is presented in Algo. 7. This algorithm's input is a map, of any level, and a list of regions, sorted as explained above, and its return is the corresponding inclusion tree. This algorithm is made of two loops. The first loop traverses unmarked regions of list L. Indeed, when a region is marked, this is because it was already considered and placed in the inclusion tree. For an unmarked region, we initialize the first dart of the traversal of the second loop with the representative of this region (called d_{init}). The region which contains this region, is inevitably the region of dart β_2 sewn to dart d_{init}. Indeed, list L is sorted so that, when we meet the first region of a connected component, we know that this region has no region belonging to the same connected component at its top and on its left. So $\beta_2(d_{init})$ is obviously

Algorithm 7: Inclusion tree computation.

Input: The map M of a labeled image I.
The list L of regions of the image I.

Output: An inclusion tree T.

Unmark each region of L
Add a node corresponding to the infinite region as root of T
1 **foreach** *region* R *of* L *not marked* **do**
  $d_{init} \leftarrow$ representative(R)
  father $\leftarrow$ the node corresponding to $region(\beta_2(d_{init}))$
2 **foreach** *dart* d *of the connected component incident to* d_{init} **do**
    if region(d) *not marked* **then**
      Add a node corresponding to *region(d)* in T
      Set this node as son that *father* in T
      Mark *region(d)*

return T

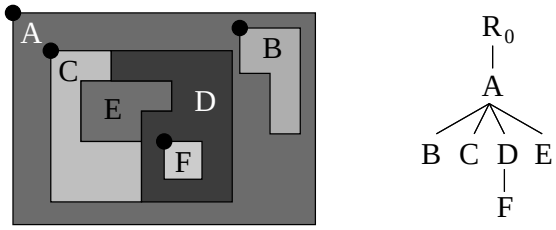


Fig. 23. A labeled image and its inclusion tree.

a dart of the region containing the current region, and each region belonging to the same connected component is also included in this same region.

The second loop of the algorithm traverses each dart of the connected component incident to d_{init}, and for each region not already met, it puts this region as son of *father*. Since we traverse each dart of the connected component, we are going to consider exactly all regions of this component, and not the included regions which will be treated during a next round of the first loop.

We can see in Fig. 23 a labeled image and its inclusion tree. The sorted list of regions is *A, B, C, D, E, F* (first sorted by y coordinate then by x). So the first region considered is A which has for representative the dart that contains the point at the top left corner of this region. d_{init} is initialized to this dart and so *father* to the infinite region. Then, the second loop of the algorithm traverses the connected component of A but here there is no region adjacent to A (not included) so we only put A as son of R_0.

The second region considered is B, its representative dart contains the point at the top left of this region (d_{init}) and so $\beta_2(d_{init})$ belongs to region A. Region B is put as son of A in the inclusion tree. As for region A, there is no region

adjacent to B so the second loop marks only darts of B. The next region is C, d_{init} is initialized to the dart that contains the top left point of C. Here, the second loop traverses regions C, D and E because they are in the same connected component. Each of these regions is put as son of A in the inclusion tree (that corresponds to the notion of inclusion used in this work). The last region considered is F (because others regions of the list are already marked) which is put as son of D in the tree.

The complexity of this algorithm is linear in number of darts of the map, and in number of regions of the image. Since the number of regions is lower that the number of darts, the complexity is so linear in number of darts of the map. Indeed, we traverse each dart exactly once, because we traverse the connected components one by one, and two different connected components are inevitably disjointed. We also traverse each region only once because we traverse the list of regions and mark each treated region. This algorithm works for every map level. Moreover, its execution time decreases when the level increases, even if the global complexity doesn't change, because the number of darts to traverse decreases more and more for higher levels.

9 Comparison With Similar Existing Approaches

In order to compare our solution with existing approaches, we are going to describe more precisely the two that are very near to our solution: the topological map of J.P. Domenger et Al. [19,11] and the topological graph of frontier (TGF) of C. Fiorio et Al. [21,22]. These two solutions are both based on combinatorial maps, and both define the minimal model that represents an image partition. For these reasons, the three methods (solution of J.P. Domenger et Al., C. Fiorio et Al. plus the new method presented in this paper) have the same topological representation. The main differences are about the embedding and about the algorithms used to extract each model from an image.

9.1 The Topological Map of J.P. Domenger et Al.

We present here the definition given in [10]. Combinatorial maps used in this work are not exactly the same as the ones presented in this paper (cf. Sec. 2). Differences are only about the name of the applications. They use a combinatorial map with σ a permutation that gives for a dart the next dart of the same vertex, and an involution α which connects the two darts incident to the same edge. α if equal to our β_2, and σ can be defined by $\beta_2 \circ \beta_1$. Indeed, to represent a 2D combinatorial map, we can choose to encode the permutation around vertices or the one around faces. Our notation can be extended more

easily in higher dimension that their notation since we can add any β_i that puts in relation two i-cells.

In order to be addressed, each face is associated with a unique label. But contrary to our label, two darts have the same label if and only if they belong to the same face orbit. So an internal contour does not have the same label as its corresponding external contour. These labels are used to represent the inclusion relations by using two functions called *prnt* for father and *chld* for children. An external contour has no father (which is represented by the particular value 0 for the function *prnt*), and an internal contour has no child. At last, another function called β gives a canonical dart for each label. This last function is equivalent to our representative dart.

Comparing this solution to our inclusion tree, we can see that the two solutions are very near. We use only a label for each region that allows us to add other information about region (mean grey level, number of pixels...). But we can not say in constant time if there is a boundary between a region and its father region, since we need to traverse the face orbit. This information is given immediately by the solution of J.P. Domenger et Al. But there are some cases where their solution need more operations than ours. For example when an edge removal leads to the creation of an internal border, we only need to modify the position of one region in the inclusion tree, but they need to add a new label and to traverse the face orbit in order to put this new label onto all the darts of the new internal contour.

They define the topological map by introducing the notion of *boundary points* which are pointels with more than two incident linels that belong to a boundary. The boundary points are then connected with edges when there is a contour linking them in the image. We can see that this leads to the same map as our solution since we remove all degree two vertices. At the end, we thus obtain only points that have more than two incident linels (if we except the cases of loops).

Their solution for the embedding of the topological map consists in a matrix that represents all interpixels cells (pointels and linels) that belong to the image boundaries, and which is called *boundary image*. They use an efficient coding in order to compress the memory necessary to encode this matrix. But they also need to keep some additional information in order to be able to retrieve a dart that contains a given pointel, and to retrieve, given a dart, the corresponding 1D curve in the boundary image. This is made with two additional data structures that are correspondence tables in the both ways. There are so many additional structures that need to be updated according to each operation that modifies the map.

This leads to time consumption since we have more structures to update, and

leads to more complex algorithms since we need to study how an operation on the map possibly modifies other structures. Comparing to our solution, we only have the map plus an inclusion tree with an unique labelling function that gives for each dart its belonging region. Moreover, our embedding can be updated automatically by the sewing and unsewing operations since it is quite independent of the map, which is not the case for their solution.

The last difference is about the algorithm used to extract the topological map. They use a contour tracking algorithm that follows the boundaries in the image. They start from an initial loop that represents the boundary between the image and the infinite region. Then, they select boundary points by looking at the number of pixels having different labels around each pointel. These points are inserted into the boundary images by splitting existing edges. Then, edges are inserted between boundary points by following the outer contours. Last, it is necessary to scan each region to look for possible new included connected components. Compared with our approach, this algorithm is more complex since it needs a more complex image scan and several distinct steps. Another advantage of our solution is that we can choose to use the optimal algorithm with precodes in order to optimize time, or to use the basic algorithm to give a simple algorithm easy to develop. Moreover, our optimal solution with precodes is more efficient as we can see in Sec. 10. At last, our model and our extraction algorithm can be extended more easily in higher dimension since we do not have many different structures and our algorithm is only based on the removal operations (which are defined in any dimension), and since we have a solution which is a lot simpler than their extraction algorithm.

9.2 The Topological Graph Of Frontier of C. Fiorio et Al.

We present here the definition that we can find for example in [22]. They see their model as an extension of the RAG (region adjacency graph) with multi-adjacency relations encoded by a 2D combinatorial map. This leads to a difference for the representation of the structure, where edges are drawn between the barycenter of regions instead of our representation where edges are drawn on the image boundaries. But this is only a graphical representation and not a main difference between the two approaches. They use the same combinatorial maps notations as J.P. Domenger et Al. with α and σ.

This is the definition of the FTG that we can find in [22]:

Definition 11 *A FTG is an extended combinatorial map $G(V, D, \alpha, \sigma)$ where V is the set of vertices of the graph, D the set or darts, α a permutation on D and σ an involution on D such that:*

(1) each region is represented by one and only one vertex;

(2) a dart e is always incident to a vertex R of V. The notation e^R denotes the incident vertex of e;
(3) an edge of the graph is a non-ordered pair $(e, \sigma(e))$ where e belongs to D. Let E be the set of edges, $(e_1^R, e_2^{R'}) \in E$ if and only if there exists a frontier between R and R'. Thus an edge represents a frontier. A dart symbolizes the frontier "as seen from" the region to which it is incident;
(4) the cycles of permutation α correspond to a contour of a given region and respect the order induced by the sequence of frontiers making the contour;
(5) inf *is a particular vertex of the graph symbolizing the exterior of the image;*
(6) to each vertex is associated the list of the cycles of α related to the contours of the region represented by this vertex. By convention, the exterior contour will always be the first of the list.

This definition defines directly the minimal combinatorial map that represents an image (item 3, each edge corresponds exactly to a frontier). Moreover, it fixes the relation between regions (the vertices V) and the combinatorial map (item 2). Item 4 is necessary in order to guaranty that the order of the edges in the map is the same as the order of frontiers in the image. At last, item 6 is equivalent to our inclusion tree. Indeed, each region has an external contour (the first element of the list), then possibly some internal contours that are holes in the regions. This implicit encoding is equivalent to the explicit function that puts in relation the contours in the work of J.P. Domenger et Al. We can note that this definition is more complex than ours, since the minimal model is directly defined. Moreover, we can do the same criticism as for the previous approach. The fact that the combinatorial map is linked with other additional structures, leads to more complex algorithms when we need to update the model, and thus leads more important complexity. Moreover, this TGF is only a topological model and does not keep the region geometry.

To extract the TGF, they use an algorithm based on precodes (this algorithm was our initial inspiration to our algorithm). But since TGF is a unique structure, they need to build at the same time the map and the relations between frontiers (which are equivalent to our inclusion tree). This leads to more complex algorithms since they do not only need to give operations that modify the map, but also operations that update the inclusion tree. They keep a list of pending contours (contours that touch the current line of the image scan) and updating the type of contour (internal or external contour) depending of the current precode and a test to see if darts belong to the same orbit or not.

For these reasons, the TGF is difficult to extend in higher dimension. First the definition is complex to extend since it is based on the order of boundaries in the image, and this order is not the same in 3D for surfaces. Moreover the extraction algorithm needs to consider all precodes (15 in 2D and 4140 in 3D) to build the inclusion tree, contrary to our approach with simplification

levels that allows us to factorize many cases (only 10 precodes in 2D and 129 precodes in 3D). Moreover, our computer software which extracts our model is more efficient than their software since we made less operations (see Sec. 10).

10 Experiments and Analysis

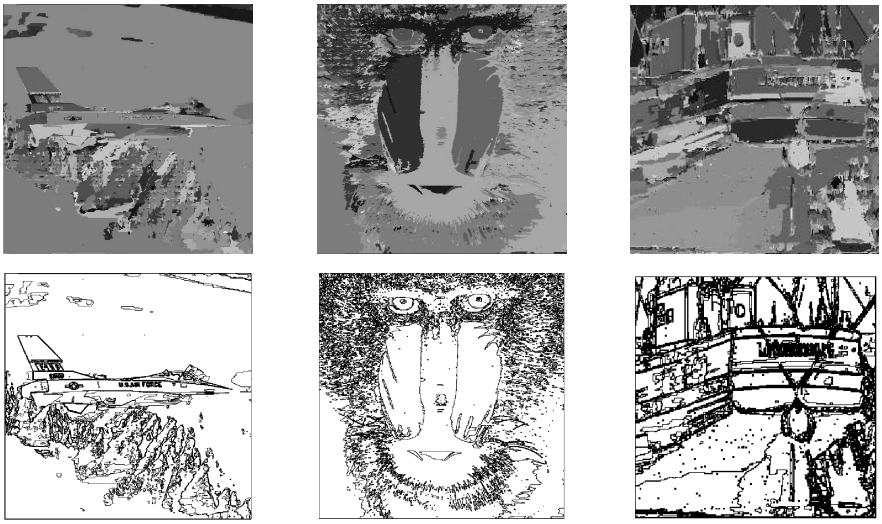
We have implemented our optimal extraction algorithm based on precodes for the three different map levels. We have used the embedding presented in Sec. 5 (for level 1 and 2 a vertex embedding, and for level 3 a vertex embedding plus an open edge embedding). Moreover, we have compared our software with the software of L. Brun [9] which implements the method first proposed by J.P. Domenger et Al. [19,11], and the software of C. Fiorio which constructs the TGF [21,22]. Experiments were made on a personal computer with a 1500MHz CPU with 256Mb of memory and a Linux system.

We choose to represent each dart by a structure where each β_i relation is simply a pointer to the dart i-sewn. There are many different ways to represent maps in computer memory, and the choice of one of them depends on what we would like to prioritize: memory cost or time consumption. Our choice clearly prioritize time since we access to each β_i of a given dart in direct access, and we can also easily modify all the β_i relation without constraint. But of course this is to the detriment of memory space occupation. This choice need to be studied more precisely for each particular need of a given application, but this question could not be answered in a general way. Moreover, our software is develop in C++ without particular optimization.

We have used for our experiments the six labeled images well known in image processing shown in Fig. 24 and Fig. 25. These figures show, firstly, the labeled image (one color by different label), then the maps obtained which represent the interpixel boundaries. Note that the segmentation algorithm used to obtain the label image is very simple (only based on the mean grey value of regions). We do not try to obtain here a good segmentation but only to compare the different map levels on the same images. We can find in the legend of the figures the size of each image in pixels and the number of regions obtained by the segmentation.

First we only compare the three simplification levels. We can see in Fig. 26 the evolution of memory space, number of darts and number of vertices for level 1, 2 and 3 maps, in level 1 percentage. The memory space evolution shows that the gain is very significant. It is about 36% between level 1 and

[9] Thank you very much to Luc Brun for his help which allowed us to use his software and to realize this comparison.



(a) (b) (c)

Fig. 24. (a) Airplane, 512×512 pixels, 6361 regions. (b) Baboon, 512×512 pixels, 6705 regions. (c) Cornouaille, 256×256 pixels, 5410 regions.

(a) (b) (c)

Fig. 25. (a) Goldhill, 720×576 pixels, 8006 regions. (b) Lena, 512×512 pixels, 6198 regions. (c) Peppers, 512×512 pixels, 5765 regions.

level 2, and about 34% between level 2 and level 3 (this is a mean between the six considered images). We thus obtain that the memory gain is about 58% between level 1 and level 3. This is not a surprise since a lot of darts are removed during the successive simplifications.

But the memory gain in not the unique advantage of level 3 maps. Indeed, if we look at the number of darts, we can observe that level 3 map has only about 30% of darts of level 1 map. This significant difference leads to decrease

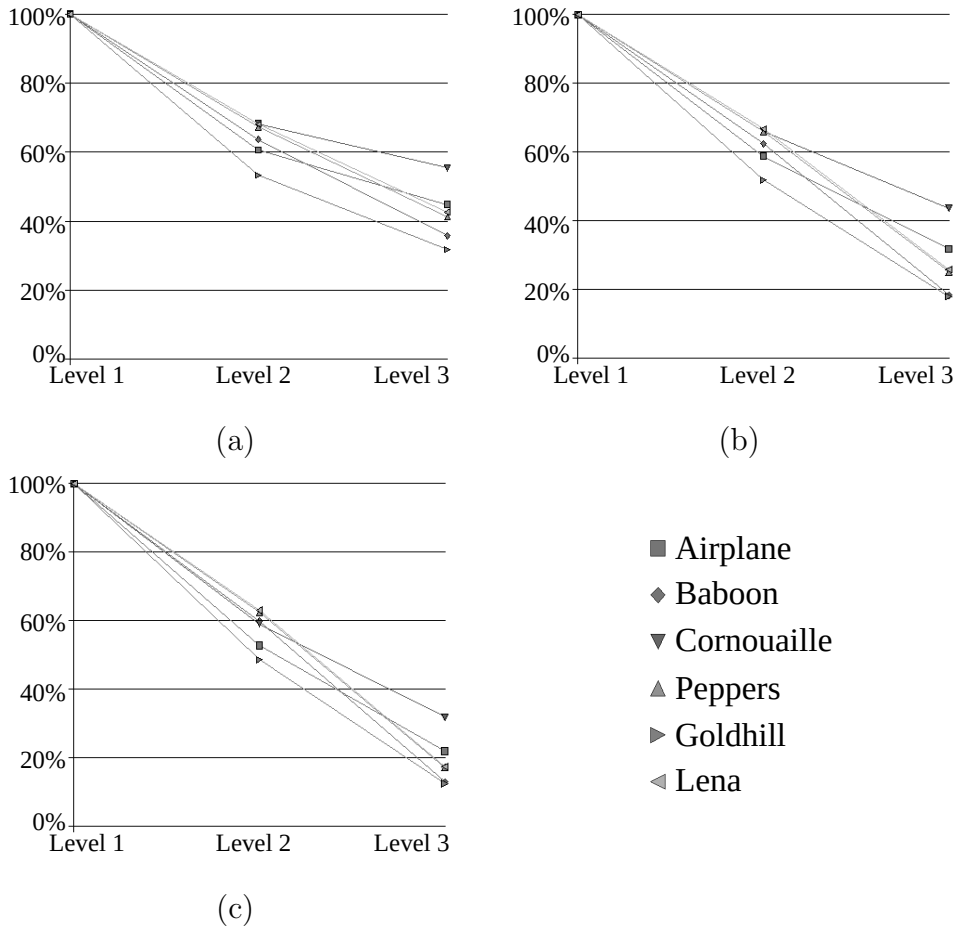


Fig. 26. Characteristics evolution for the three different map levels, in level 1 percentage. (a) Memory space. (b) Number of darts. (c) Number of Vertices.

the computational times for algorithms define on level 3 map due to a more direct access to the adjacency information.

We can see in Tab. 1 the extraction time in seconds, and the memory space occupation in mega-bytes for levels 1, 2 and 3 by using our method, and for the two other softwares of L. Brun and C. Fiorio. First, we can notice, on these different extraction examples, that the execution time of our algorithm is about the same to extract any simplification level. Indeed, all precode algorithms are similar (mainly made of some sewing and unsewing operations) and so take about the same time. The execution time mainly depends on the image size and on the number of regions. Indeed, when there are less regions, there are also less darts and the computation of the inclusion tree requires less time. We can also see that for some images (for example airplane or goldhill) the execution time decreases for higher levels. This is due to the less significant number of memory allocation and disallocation for higher levels.

Note that these times are the full execution times including initialization, image reading, topological and geometrical part and inclusion tree computation.

Table 1
Comparison of extraction time (in seconds) and memory space occupation (in megabytes) for each image.

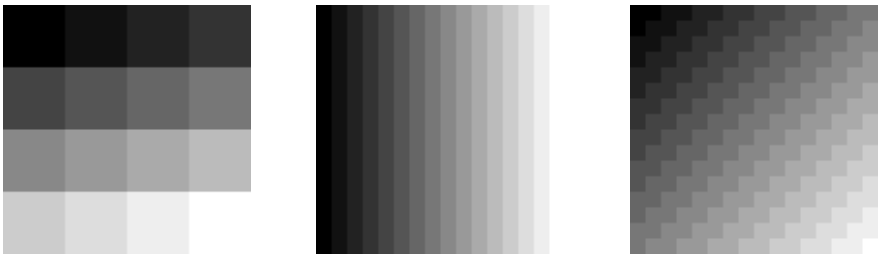
Image	Level 1	Level 2	Level 3	L. Brun	C. Fiorio
Airplane	0.08s	0.06s	0.05s	0.71s	0.25s
	3.85Mb	2.34Mb	1.6Mb	1.64Mb	1.49Mb
Baboon	0.11s	0.08s	0.09s	6.96s	0.35s
	6.11Mb	3.9Mb	1.89Mb	2,75Mb	1.41Mb
Cornouaille	0.05s	0.03s	0.04s	0.27s	0.20s
	2.46Mb	1.69Mb	1.3Mb	1,36Mb	1.3Mb
Goldhill	0.14s	0.11s	0.11s	2.7s	0.43s
	7.97Mb	4.26Mb	2.24Mb	2,76Mb	3.62Mb
Lena	0.08s	0.08s	0.08s	0.97s	0.35s
	4.36Mb	2.98Mb	1.66Mb	1,79Mb	1.4Mb
Peppers	0.09s	0.07s	0.08s	1.5s	0.31s
	4.36Mb	2.94Mb	1.61Mb	1,63Mb	1.37Mb

This time is divided in about 7% for the inclusion tree computation, 26% for the geometrical part and 37% for the topological one. The remaining 30% are for the image scan and the current precode test. These percentages are approximative since they depend on the images.

Now if we compare our solution with the two existing ones, we can see that we are really more efficient in computation times, and that the three methods are quite similar for memory space occupation (of course without considering level 1 and level 2 maps). For memory space, this is due to the fact that the three maps are similar, and differences only concern the embedding. The computation time is larger for the method of L. Brun since its extraction algorithm is a contour tracking which makes much more operations than our method. The difference with the method of C. Fiorio is probably due to the inclusion tree computation which is made in the same time as the map extraction (cf. Sec. 9).

These results are a good surprise since our first goal was not to improve existing methods but only to revisit these methods to propose a definition that could be easily extended in higher dimension. Our works made in order to simplify algorithms lead to simpler computer software and thus also more efficient. This shows another advantage of our approach.

In order to compare more precisely our simplification levels, we have made



(a) (b) (c)

Fig. 27. (a) Square image, 128×128 pixels with $l = 32$ so with 16 regions. (b) Column image, 128×128 pixels with $l = 8$ so with 16 regions. (c) Stairs image, 128×128 pixels with $l = 8$ so with 16 regions.

some experiments with artificial regular images. This allows to show the extreme cases, for example when all regions are made of squares or otherwise when all regions have only stair boundaries. For that, we have made four types of images:

(1) square images where each region is a square of length l;
(2) line images where each region is a line, indeed a rectangle with its height equal to the height of the image and its width equal to l;
(3) column images which are the same as line images but with a 90 degree rotation;
(4) stair images: each region is a 45 degree oriented rectangle. So its borders are made of successive stairs of length l.

In order to obtain regular images, we have worked with a 128×128 pixels image, and only use for regions the lengths which are a power of two. For each type of image, we generate each image starting with l equal to 1 and multiply by two until length equal to 128. Thus we can observe the relation between the length of regions and the memory space occupied by each level. We can see in Fig. 27 some examples of obtained images. Since images are regular, we can compute, given the length of each region, the number of regions. For example, when we have a square image with 128×128 pixels, the number of regions is $(128 \times 128)/l^2$

We give in Tab. 2 the memory space required for square images for each size of squares between 1 and 128. We can observe that level 2 and level 3 are very near since almost all region boundaries are straight line segments (in fact all boundaries except for the 4 corners of the image).

In Tab. 3, we make the same experiment with column images, for each length between 1 and 128. Again, level 2 and level 3 are very near since almost all region boundaries are straight line segments. We obtain exactly the same results for line images since the maps are exactly the same in both cases.

Table 2
Comparison of memory space occupation (in kilo-bytes) for each length of squares between 1 and 128.

Length	1	2	4	8	16	32	64	128
Nb regions	16384	4096	1024	256	64	16	4	1
Level 1	2462	1278	662	348	182	100	60.1	40.1
Level 2	2462	628	167	50.5	13.3	3.67	1.10	0.39
Level 3	2461	627	166	50.3	13.1	3.45	0.89	0.21

Table 3
Comparison of memory space occupation (in kilo-bytes) for each length of columns between 1 and 128 (exactly the same results for line images).

Length	1	2	4	8	16	32	64	128
Nb regions	128	64	32	16	8	4	2	1
Level 1	1314	672	351	190	110	70.2	50.1	40.1
Level 2	34.1	17.1	8.62	4.37	2.25	1.18	0.65	0.39
Level 3	33.9	16.9	8.39	4.14	2.01	0.95	0.42	0.21

Table 4
Comparison of memory space occupation (in kilo-bytes) for each length of step stairs between 1 and 128.

Length	1	2	4	8	16	32	64	128
Nb regions	128	64	32	16	8	4	2	1
Level 1	1314	672	351	190	110	70.2	50.1	40.1
Level 2	1294	327	83.7	21.9	6.07	1.89	0.73	0.39
Level 3	286	79.4	23.7	7.78	2.85	1.13	0.44	0.21

But if we look in Tab. 4 where we make the same experiment with stair images, for each length between 1 and 128, we can see that it is the opposite. Indeed, it is now level 1 and level 2 that are mainly near, since almost all region boundaries are not straight line segments. These boundaries become straighter and straighter when l becomes larger, and so level 2 becomes closer to level 3 when l is large.

We can see in Fig. 28 three curves that show the memory space evolution for each type of images in function of the length of regions. On each figure, there are two curves that show the memory space occupation for level 2 and level 3 map, in percentage of the memory of level 1 map. Curves shown in Fig. 28(a) show that level 2 and level 3 are very near for square images. It is also the case for line images (see Fig. 28(b)), even if we can see differences between the two curves, but this is due to the y-axis which is very precise (between 0 and

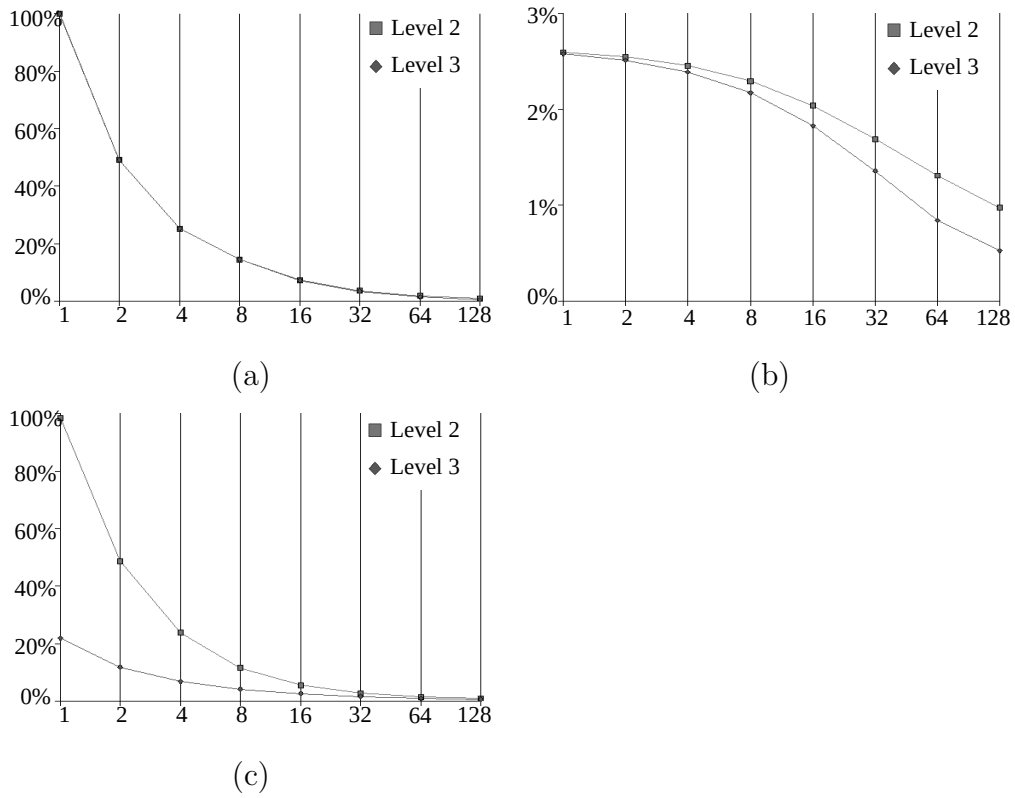


Fig. 28. Memory space evolution for level 2 and level 3 maps, in level 1 percentage. (a) Square images. (b) Line (resp. column) images. (c) Stair images.

3%). Indeed, even when the length of lines is equal to one, regions have very large straight boundaries and so the memory occupied by level 2 and level 3 is much smaller than the memory required by level 1. Otherwise, we can see in Fig. 28(c) that memory occupation of level 3 is much smaller than the one of level 2, even if the difference decreases when the length of the stairs increases.

These four experiments with regular images show that the difference between level 2 and level 3 can be very significant or not, depending on the images content. But in general, if we consider "real images", boundaries are often not straight and the difference between level 2 and level 3 memory occupation is significant. This is confirmed by our experiments made with real images (see Tab. 1).

11 Conclusion

In this paper we have presented a model for 2D image representation, and have given an optimal extraction algorithm which computes this model from a labeled image.

Several works have already studied the use of combinatorial maps to represent interpixel boundaries of 2D images. But our approach is original due to the new notion of simplification level which allows to give a formal and constructive definition of the topological map. Moreover, these levels can be extended easily in higher dimension [2,16], whereas for other approaches this extension is much more difficult, even not possible.

Of course, some problems still remain in higher dimension since we do not know the topological classification of objects ([30] discusses about these problems and a possible solution). Nevertheless, we can define the topological map in any dimension and use this tool in order to try to characterize discrete objects. For that, we are studying how to compute homology groups on topological map. Another idea is to add fictive cells in the topological map in order to keep each i-cell homeomorphic to an i-sphere. This can be achieved without particular problems, thanks again to our progressive simplification levels, where we can control if a removal operation leads to a disconnection. Then we need to study how obtained objects can be interpreted and how modification algorithms have to process these particular elements.

The topological map is the last simplification level. We have proved that this map is *complete* by showing that no information is lost during the different simplifications, *minimal* because we can not remove anything in this map without changing the topology, and *unique* because this map only depends on the topology of the image and so two topological equivalent partitions have the same level 3 map. For these reasons, topological map is a good model for image processings. Indeed, it allows to retrieve most of the information which may be required by an image processing algorithm with a low computational cost. Thus we can consider many different image processing algorithms that only work on topological map and that use many kind of information.

We also have presented in this paper an optimal extraction algorithm which uses the notion of precode. It can extract any map level in one image scan. Moreover this algorithm is simple to implement since it is based on the definition of a limited number of operations to do, according to a limited number of local configurations. We have given for each simplification level the precodes required to extract the corresponding map and the algorithms associated to each of them. This has allowed us to show that there is a correspondence between the levels and the number of precodes required. At last, we have shown in different experiments that the topological map needs less memory than other levels and we have computed the gain observed comparatively to each level. Moreover, we have compared our solution with other existing approaches and showed that our software is more efficient to construct the topological map than other ones. This is due to our optimal extraction algorithm and to our effort to simplify the definition of our model.

We are currently working on the use of our model to propose new segmentation algorithms or to improve existing techniques [6]. The first results are encouraging and let us hope to achieve interesting solutions. But our main goal is to extend this work in 3D and so to propose new 3D segmentation algorithms based on the 3D topological map [18]. This work is quite advanced and we already obtained first results [2,16]. At last, we are also working on operations allowing to modify the topological map. Indeed by mixing the segmentation part of our work, the extraction of the map, and the different interactive or automatic operations, we could define a complete image analysis software based on the topological map that takes into account topological information of the analyzed objects.

Acknowledgements

The authors are grateful to reviewers for their useful comments and suggestions which greatly improved the paper. We also would like to especially thank all Gral family for his careful reading and correction of this paper.

References

[1] E. Ahronovitz, C. Fiorio, and S. Glaize. Topological operators on the topological graph of frontiers. In *Discrete Geometry for Computer Imagery*, number 1568 in Lecture Notes in Computer Science, pages 207–217, Marne-la-Vallée, France, 1999.

[2] Y. Bertrand, G. Damiand, and C. Fiorio. Topological encoding of 3d segmented images. In *Discrete Geometry for Computer Imagery*, number 1953 in Lecture Notes in Computer Science, pages 311–324, Uppsala, Sweden, december 2000.

[3] Y. Bertrand, G. Damiand, and C. Fiorio. Topological map: Minimal encoding of 3d segmented images. In *Workshop on Graph Based Representations*, pages 64–73, Ischia, Italy, may 2001. IAPR-TC15.

[4] Y. Bertrand and J.F. Dufourd. Algebraic specification of a 3d-modeler based on hypermaps. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 56(1):29–60, january 1994.

[5] Y. Bertrand, C. Fiorio, and Y. Pennaneach. Border map: a topological representation for nd image analysis. In *Discrete Geometry for Computer Imagery*, number 1568 in Lecture Notes in Computer Science, pages 242–257, Marne-la-Vallée, France, 1999.

[6] P. Bourdon, O. Alata, G. Damiand, C. Olivier, and Y. Bertrand. Geometrical and topological informations for image segmentation with monte carlo markov

chain implementation. In *Vision Interface*, pages 413–420, Calgary, Canada, may 2002.

[7] J.-P. Braquelaire and L. Brun. Image segmentation with topological maps and inter-pixel representation. *Journal of Visual Communication and Image Representation*, 9(1):62–79, march 1998.

[8] J.-P. Braquelaire, P. Desbarats, and J.-P. Domenger. 3d split and merge with 3-maps. In *Workshop on Graph Based Representations*, pages 32–43, Ischia, Italy, may 2001. IAPR-TC15.

[9] J.-P. Braquelaire, P. Desbarats, J.-P. Domenger, and C.A. Wüthrich. A topological structuring for aggregates of 3d discrete objects. In *Workshop on Graph Based Representations*, pages 193–202, Austria, may 1999. IAPR-TC15.

[10] J.-P. Braquelaire and J.-P. Domenger. Representation of segmented images with discrete geometric maps. *Image and Vision Computing*, 17(10):715–735, 1999.

[11] L. Brun and J.-P. Domenger. A new split and merge algorithm with topological maps and inter-pixel boundaries. In *The fifth International Conference in Central Europe on Computer Graphics and Visualization*, february 1997.

[12] L. Brun, J.-P. Domenger, and J.-P. Braquelaire. Discrete maps : a framework for region segmentation algorithms. In *Workshop on Graph Based Representations*, Lyon, april 1997. IAPR-TC15. published in Advances in Computing (Springer).

[13] P. Charnier. *Outils algorithmiques pour le codage interpixel et ses applications.* Thèse de doctorat, Université Montpellier II, janvier 1995.

[14] R. Cori. *Un code pour les graphes planaires et ses applications.* PhD thesis, Universit Paris VII, 1973.

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

[16] G. Damiand. *Définition et étude d'un modèle topologique minimal de représentation d'images 2D et 3D.* Thèse de doctorat, Université Montpellier II, décembre 2001.

[17] G. Damiand and P. Lienhardt. Removal and contraction for n-dimensional generalized maps. In *Computer Vision Winter Workshop*, pages 208–221, Bad Aussee, Austria, february 2002.

[18] G. Damiand and P. Resch. Topological map based algorithms for 3d image segmentation. In *Discrete Geometry for Computer Imagery*, number 2301 in Lecture Notes in Computer Science, pages 220–231, Bordeaux, France, april 2002.

[19] J.P. Domenger. *Conception et implémentation du noyeau graphique d'un environnement 2D*1/2 *d'édition d'images discrètes.* Thèse de doctorat, Université Bordeaux I, avril 1992.

[20] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, 7, 1960.

[21] C. Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation.* Thèse de doctorat, Université Montpellier II, novembre 1995.

[22] C. Fiorio. A topologically consistent representation for image analysis: the frontiers topological graph. In *Discrete Geometry for Computer Imagery*, number 1176 in Lecture Notes in Computer Science, pages 151–162, Lyon, France, november 1996.

[23] J. Françon. Topologie de khalimsi et kovalevski et algorithmique graphique. Rapport de recherche 91-10, Université Louis-Pasteur, Centre de Recherche en Informatique, Strasbourg, France, 1991.

[24] A. Jacques. Constellations et graphes topologiques. In *Combinatorial Theory and Applications*, volume 2, pages 657–673, 1970.

[25] E. Khalimsky, R. Kopperman, and P.R. Meyer. Boundaries in digital planes. *Journal of Applied Mathematics and Stochastic Analysis*, 3(1):27–55, 1990.

[26] T.Y. Kong, R. Kopperman, and P.R. Meyer. A topological approach to digital topology. *American Mathematical Monthly*, 98(10):901–917, 1991.

[27] T.Y. Kong and A. Rosenfeld. Digital topology: introduction and survey. *Computer Vision, Graphics, and Image Processing*, 48(3):357–393, 1989.

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

[29] W.G. Kropatsch. Building irregular pyramids by dual-graph contraction. *Vision, Image and Signal Processing*, 142(6):366–374, december 1995.

[30] W.G. Kropatsch. Abstraction pyramids on discrete representations. In *Discrete Geometry for Computer Imagery*, number 2301 in LNCS, pages 1–21, Bordeaux, France, april 2002.

[31] W.G. Kropatsch and H. Macho. Finding the structure of connected components using dual irregular pyramids. In *Discrete Geometry for Computer Imagery*, pages 147–158, *invited lecture*, september 1995.

[32] 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.

[33] P. Lienhardt. Topological models for boundary representation: a comparison with n-dimensional generalized maps. *Commputer Aided Design*, 23(1), 1991.

[34] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324, 1994.

[35] J.-G. Pailloncy and J.-M. Jolion. The frontier-region graph. In *Workshop on Graph Based Representations*, volume 12 of *Computing Supplementum*, pages 123–134. Springer, april 1997.

[36] A. Rosenfeld. Adjacency in digital pictures. *Information and Control*, 26(1):24–33, 1974.

[37] W.T. Tutte. A census of planar maps. *Canad. J. Math.*, 15:249–271, 1963.

A All the precodes algorithms

Algorithm 8: Code corresponding to precode l_1.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$
0-unsew(last); 1-unsew(up)
1-sew(up,last); 1-sew(a,b)
Embedding of the vertex incident to $last \leftarrow$ (x,y)
return *up*

Algorithm 9: Code corresponding to precode l_2.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow$ a new dart; $b \leftarrow$ a new dart
$c \leftarrow \beta_1(last)$; 1-unsew(last)
1-sew(last,a); 1-sew(a,b)
1-sew(b,c); 2-sew(up,last)
Embedding of the vertex incident to $b \leftarrow$ (x,y)
return *a*

Algorithm 10: Code corresponding to precode l_3.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow$ a new dart; $b \leftarrow$ a new dart
$c \leftarrow \beta_1(up)$; 1-unsew(up)
1-sew(up,a); 1-sew(a,b)
1-sew(b,c); 2-sew(last,b)
Embedding of the vertex incident to $a \leftarrow$ (x,y)
return *up*

Algorithm 11: Code corresponding to precode l_4.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow$ a new dart; $b \leftarrow$ a new dart
$c \leftarrow$ a new dart; $d \leftarrow$ a new dart
1-sew(a,b); 1-sew(b,c)
1-sew(c,d); 1-sew(d,a)
2-sew(last,a); 2-sew(up,b)
Embedding of the vertex incident to $d \leftarrow$ (x,y)
return *c*

Algorithm 12: Code corresponding to precode f_1.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$
$c \leftarrow \beta_1(up)$; $d \leftarrow \beta_1(c)$
1-unsew(last); 2-unsew(a)
0-unsew(c); 1-unsew(c)
1-sew(last,c); 1-sew(c,b)
1-sew(up,d); 2-sew(up,a)
Embedding of the vertex incident to $c \leftarrow$ (x,y)
return *last*

Algorithm 13: Code corresponding to precode f_2.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$; $c \leftarrow \beta_1(up)$
0-unsew(last); 1-unsew(last); 1-unsew(up)
0-sew(b,a); 1-sew(up,last); 1-sew(last,c)
Embedding of the vertex incident to $last \leftarrow$ (x,y)
return *up*

Algorithm 14: Code corresponding to precode t_1.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow$ a new dart; $b \leftarrow$ a new dart
$c \leftarrow \beta_1(last)$; 2-sew(up,a)
Add geometrical point incident to *last* at the end of the edge embedding of a
0-unsew(last); 1-unsew(last)
1-sew(a,last); 1-sew(last,b)
1-sew(b,a); 0-sew(c,up)
Embedding of the vertex incident to $b \leftarrow$ (x,y)
return *last*

Algorithm 15: Code corresponding to precod t_2.

Input: *last* and *up*, (x,y) coordinates of the new vertex.
Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(last)$; $c \leftarrow \beta_1(up)$
Add geometrical point incident to *last* at the end of the edge embedding of a
0-unsew(last); 1-unsew(last); 1-unsew(up)
0-sew(b,a); 1-sew(up,last); 0-sew(c,last)
Embedding of the vertex incident to $last \leftarrow$ (x,y)
return *up*

Algorithm 16: Code corresponding to precod t_3.

Input: *last* and *up*, (x,y) coordinates of the new vertex.

Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_2(a)$; $c \leftarrow \beta_0(b)$
$d \leftarrow \beta_2(c)$; $t_1 \leftarrow \beta_1(b)$; $t_2 \leftarrow \beta_1(d)$
Add geometrical point incident to *last* at the end of the edge embedding of a
Merge edge embedding of b at the end of edge embedding of a
0-unsew(b); 1-unsew(b); 2-unsew(b); delete b
0-unsew(d); 1-unsew(d); 2-unsew(d); delete d
0-unsew(last); 1-sew(up,last)
2-sew(a,c); 1-sew(a,t_2); 1-sew(c,t_1)
Embedding of the vertex incident to $last \leftarrow$ (x,y)
return *up*

Algorithm 17: Code corresponding to precod t_4.

Input: *last* and *up*, (x,y) coordinates of the new vertex.

Output: The "next" *last*.

$a \leftarrow \beta_0(last)$; $b \leftarrow \beta_1(up)$; $c \leftarrow \beta_1(b)$
Add geometrical point incident to *last* at the end of the edge embedding of a
0-unsew(last); 0-unsew(b)
1-unsew(b); 2-unsew(b)
2-sew(up,a); 1-sew(up,c)
1-sew(a,b); 1-sew(b,last)
Embedding of the vertex incident to $last \leftarrow$ (x,y)
return b