Topological Model for 3D Image Representation: Definition and Incremental Extraction Algorithm

Guillaume Damiand

SIC, bât SP2MI, BP 30179, 86962 Chasseneuil Cedex, France

Abstract

In this paper, we define the three-dimensional *topological map*, a model which represents both the topological and geometrical information of a three-dimensional labeled image. Since this model describes the image's topology in a minimal way, we can use it to define efficient image processing algorithms. The topological map is the last level of map hierarchy. Each level represents the region boundaries of the image and is defined from the previous level in the hierarchy, thus giving a simple constructive definition. This model is an extension of the similar model defined for 2D images. Progressive definition based on successive map levels allows us to extend this model to higher dimension. Moreover, with progressive definition, we can study each level separately. This simplifies the study of disconnection cases and the proofs of topological map properties. Finally, we provide an incremental extraction algorithm which extracts any map of the hierarchy in a single image scan. Moreover, we show that this algorithm is very efficient by giving the results of our experiments made on artificial images.

Key words: topological model, 3D image representation, intervoxel boundaries, combinatorial map, structure for image processing

1 Introduction

In this paper we present a combinatorial structure which describes intervoxel boundaries of a three-dimensional labeled image: the *topological map*. First we give a formal definition of the topological map, then we propose an incremental

⋆ Article published in Computer Vision and Image Understanding, 109(3), pp. 260-289, March 2008, doi:10.1016/j.cviu.2007.09.007

Email address: damiand@sic.univ-poitiers.fr (Guillaume Damiand).

algorithm which builds this structure from a labeled image by a single image scan.

The goal of such structuring is to propose efficient algorithms for image processing. Indeed, the efficiency of such algorithms depends on the data structure used and depends on the way that information can be retrieved. It is indeed necessary to be able to efficiently compute the geometrical or topological features that are used during algorithms. This is the case in many different applications, such as image segmentation, feature extraction, or indexing techniques based on matching algorithms...

There are many approaches that have studied the definition of such a structure to describe 2D images [44,35,24,12,42,7,9,4]. Topological data structures describe the image as a set of elements and their relations. The most famous example is the Region Adjacency Graph (RAG) [44] which is a graph which describes each region by a vertex, and where neighboring regions are connected by an edge. This neighboring information is crucial for example to implement an algorithm of image segmentation based on region growing. Indeed, with a RAG, one step of the segmentation algorithm can be computed in $O(n+m)$ (with n the number of vertices and m the number of edges), while the same algorithm has $O(n^2)$ complexity without RAG [1].

Moreover, topological data structures can also be used for matching algorithms, for recognition purposes. In this type of application, we need to characterize objects in order to recognize them. For that, many different features need to be combined: some geometrical features (like the shape of the object, its size, its color...) and some topological features (like the number of faces which compose the object, its number of neighbors, the Euler characteristic of each surface...).

In order to characterize each object as precisely as possible and to facilitate the distinction between objects, the topological data structure must contain as much information as possible. It is not the case with RAG which does not describe multiple adjacencies (so there is no difference between a region adjacent once or twice to another region). It does not make the difference between adjacency and inclusion, does not describe the order of the edges around a vertex. It does not describe the faces but only vertices and edges. For all of these reasons, it is not unique, i.e. two different images could be described by the same RAG [34].

To solve these problems, the RAG model has been extended, for instance in the dual-graph structure [35,37]. This structure is composed of two multi-graphs describing inclusion relations. The first graph is equivalent to the RAG, but

[1] With a RAG, given one region, we can run through its neighboring regions in linear time, while we need to consider all regions without a RAG.

it has got multi-edges and self-loops in order to describe 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 between relations of inclusion and relations of adjacency. Moreover, dual graphs allow, in general cases, to retrieve the order of regions around a given region. But this is not the case for particular configurations, and thus two images that are not topologically equivalent can have the same dual graphs. Another drawback of this structure is that each operation has to be applied twice (once to the primal graph and a second time to the dual one) in order to maintain the correspondence between both graphs. Finally, this structure is only defined in 2D and its extension in 3D is not straightforward. We can cite also approaches based on Reeb graph [43], a graph which allows to represent the topology of a surface. But these approaches cannot be used to represent the topology of labeled images, only to deal with the surface of a single 3D object. Indeed, a Reeb graph represents the topological skeleton of the 3D object. Moreover, Reeb graph depends also on the shape of the object and thus does not depend only on its topology.

There are many different approaches [21,23,24,11,12,7,1,10,17] that have been proposed in order to give a solution to the problem of defining a structure, describing all the information which results from region segmentation by using combinatorial maps. The basic principle of these approaches is to use a combinatorial map to describe the topology of the image. Indeed, the combinatorial map is a good model to describe a space subdivision. It is defined in any dimension and describes all the cells of the subdivision and all the relations of adjacency. Moreover, the combinatorial map can be linked to a geometrical model in order to describe the object's geometry. Lastly, it is an efficient model for retrieving and for updating information contained in the image.

These different structures have been first defined in 2D, but the need to work with images of higher dimension and specially in 3D has led to study how to extend previous works to higher dimensions. There are mainly two approaches that have been proposed to solve this problem. The first one [9,8] proposes a data structure based on a combinatorial map, that allows to describe a labeled image. This approach has different drawbacks: the first one is that it uses an implicit description of the map, which induces a topological model which is not minimal in the number of cells. This is an important drawback since the map is not representative of a given object. Another drawback is that some specific configurations of voxels cannot be encoded (if two voxels with same label are adjacent by an edge) and must be rearranged before extracting the topological map. This requires heavy pre-processing which is both time-consuming and unsatisfactory as it modifies the initial image.

The second approach is the one that we present in this paper, and which was already partially presented in [2,3]. This approach is also based on the combinatorial map, but we use an explicit description which allows to obtain the minimal model in the number of cells that we call a *topological map*. Moreover, this model does not depend on the geometry of described objects. This is important in order to characterize objects by topological invariant. At last, our model can describe any type of image without any pre-processing. The only limit used in this work is to describe images with 6-connected labeled regions. But this is only an optimization used to retrieve directly, from a given voxel, its belonging region without additional structure. This limit can be removed and we have already extended this work in order to describe any type of image. Moreover, given an image, it is possible to relabel each 6-connected region with a distinct label without modifying the initial image.

This paper is organized as follows. Firstly, in Section 2, we give a brief presentation of the 3D combinatorial map which is the basic model used in this work, and we also introduce the notations used in this paper. Then, in Section 3, we present the removal operations which are the basic operations used to define the topological map. This definition is given in Section 4 by using the notion of simplification level. This model is mainly a topological model, but we also want to keep the description of the shape of regions. We show in Section 5 how the topological map is linked with a geometrical model, here a matrix of intervoxel elements. Then we present an incremental extraction algorithm in Section 6 which computes the topological map with only one image scan. We present some experimental results in Section 7. Finally we conclude this paper in Section 8 and give some perspectives for future works.

2 Combinatorial Maps and Images

2.1 Combinatorial Maps

The subdivision of a 3D topological space is a partition of the space into 4 subsets whose elements are 0D, 1D, 2D and 3D *cells* (respectively called vertices, edges, faces and volumes, and noted i-cell for a i-dimensional cell). Boundary relations are defined between these cells, where the boundary of a i-cell is a set of $(j<i)$-cells. Two cells are *incident* if one cell belongs to the boundary of the other cell, and two i-cells are *adjacent* if they are both incident to the same $(i-1)$-cell.

A combinatorial map is a mathematical model describing the subdivision of a space, based on a planar map [22,48,29,13,14]. A combinatorial map encodes all the cells of the subdivision and all the relations of incidence and

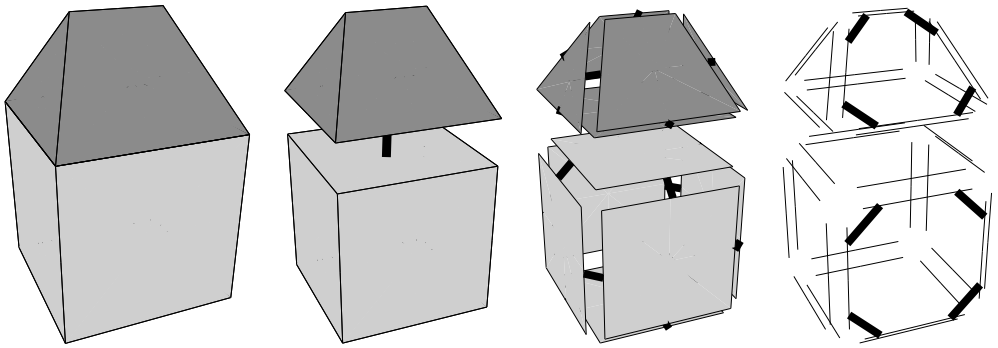


Fig. 1. The successive decompositions of a 3D space subdivision to obtain the corresponding 3-map. (A) A 3D space subdivision. (B) Disjoined volumes. (C) Disjoined faces. (D) Disjoined edges.

adjacency between the different cells, and so describes the topology of this space. A combinatorial map can be defined formally for any dimension, and we call *n-map* an n-dimensional combinatorial map. An n-map can encode an orientable quasi-manifold [2] subdivision of an n-dimensional space without boundary. Combinatorial maps were generalized in [38,40] in order to encode all n-dimensional subdivisions whether they are orientable or not and whether they are with or without boundary ([39] established a connection between maps and several other models).

A combinatorial map can be obtained intuitively by successive breakdowns as we can see in Fig. 1. To describe the 3D space subdivision shown in Fig. 1A, we first decompose the volumes of this subdivision (Fig. 1B) then the faces of these volumes (Fig. 1C) and then the edges of these faces (Fig. 1D). At each step, we keep the adjacency information between the broken down cells (drawn by black segments, only partially for the last step). The elements obtained after the last decomposition are called *darts* and are the basic only elements used in the definition of combinatorial map. In order to obtain the map, each adjacency relation is reported onto darts. β_i is the relation between two darts which describes an adjacency between two i-dimensional cells. Let us see now the formal definition of a 3D combinatorial map that we can find for example in [39]:

Definition 1 (3D combinatorial map) *A* 3D combinatorial map*, (or 3-map) is a 4-tuple* $M = (D, \beta_1, \beta_2, \beta_3)$ *where:*

(1) D *is a finite set of darts;*

[2] Intuitively, a n-dimensional quasi-manifold, called sometimes n-pseudomanifold, is an nD space subdivision which can be obtained by gluing together n-dimensional cells along *(n-1)*-dimensional cells. In such subdivision, an *(n-1)*-cell cannot belong to the boundary of more than two n-cells.

(2) β_1 *is a* permutation[3] *on* D*;*
(3) β_2 *and* β_3 *are two* involutions[4] *on* D.
(4) $\beta_1 \circ \beta_3$ *is an involution*[5] *.*

The different constraints of the 3-map definition (β_1 is a *permutation*, other β_i are involutions and $\beta_1 \circ \beta_3$ is an involution) ensures the quasi-manifold property of a described subdivision. For example, intuitively the last constraint says that two volumes cannot be partially adjacent. If two volumes are adjacent with regard to a face, then they are also adjacent with regard to each edge of the face.

Two darts d_1 and d_2 are i-sewn iff $\beta_i(d_1) = d_2$ $(1 \leq i \leq n)$. The i-sewing operation puts two darts d_1 and d_2 in relation to β_i by keeping the property of involution for $i > 1$. Indeed, in this case, i-sewing operation involves two modifications : $\beta_i(d_1) = d_2$ and $\beta_i(d_2) = d_1$, while for $i = 1$ there is only the first modification since β_1 is a permutation.

Note 1 *In the following, we denote:*

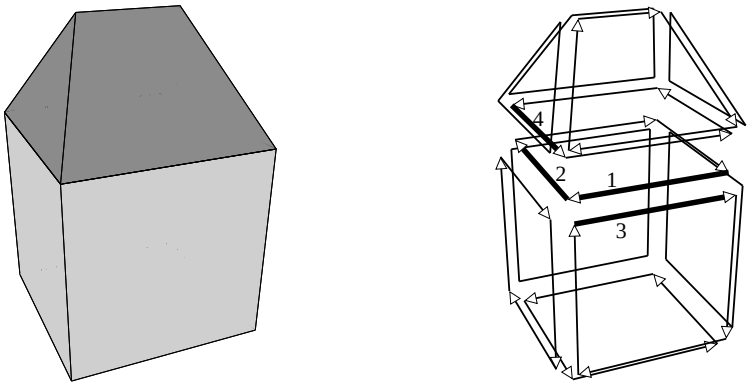
(1) β_0 *for* β_1^{-1}*;*
(2) β_{ji} *for* $\beta_i \circ \beta_j$ *(we first apply* β_j *then* β_i*, the permutations are applied in the same order as read in the notation* β_{ji}*).*

We present an example of a 3-map in Fig. 2B, and the corresponding subdivision in Fig. 2A. The β_1 relation connects an oriented edge and the following oriented edge incident to the same face and the same volume, the β_2 relation connects the two faces incident to the same edge and the same volume, and the β_3 relation connects the two volumes incident to the same edge and the same face. In order to simplify the figures, we use the graphical convention presented in Fig. 2B where the β_i are not explicitly drawn. Each dart is drawn by an arrow that shows the face orientation. With this orientation we can retrieve for each dart, the following dart on the same face and so deduce the β_1 permutation. Moreover, two darts 2-sewn or 3-sewn are drawn near and parallel and in reverse orientation, and thus the involutions can be deduced from the graphical convention.

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

Definition 2 (orbit) *Let* $\Phi = \{f_1, \ldots, f_k\}$ *be a finite set of permutations on* D*. We denote* $< \Phi >$ *the permutation group generated by* Φ*. This is the set*

[3] A *permutation* on a set S is a one to one mapping from S onto S.
[4] An *involution* f on a set S is a one to one mapping from S onto S such that $f = f^{-1}$.
[5] $\beta_1 \circ \beta_3$ is the composition of both permutations: $(\beta_1 \circ \beta_3)(x) = \beta_1(\beta_3(x))$.



A B

Fig. 2. A 3D combinatorial map example. (A) A 3D subdivision. (B) Implicit drawing of the corresponding combinatorial map. β_i applications are not explicitly drawn but can be (generally) deduced from the shape of the darts. Darts are drawn by black arrows which give the faces' orientation (arrows are only partially drawn in this figure). Two darts 1-sewn are drawn consecutively, two darts 2-sewn or 3-sewn are drawn near, parallel and in reverse orientation. If necessary, β_i applications are explicitly drawn (for example if an application cannot be deduced from the shape of darts). In this example, without the explicit description, we can retrieve that $\beta_1(1) = 2$, $\beta_2(1) = 3$ and $\beta_3(2) = 4$.

of permutations obtained by any composition and inversion of permutations contained in Φ. *The* orbit *of a dart* d *with respect 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 can be reached, starting with d, and using all combinations of all the f_i permutations. Given a 3-map and a dart d, we can retrieve all the cells incident to d by using particular orbits. The vertex (0-cell) incident to d is defined by $< \beta_{21}, \beta_{31} > (d)$, the edge (1-cell) by $< \beta_2, \beta_3 > (d)$, the face (2-cell) by $< \beta_1, \beta_3 > (d)$, and the volume (3-cell) by $< \beta_1, \beta_2 > (d)$. A dart is said to be incident to a cell if it belongs to the set of darts of the corresponding orbit. Moreover, two cells are incident if the two corresponding orbits have a non-empty intersection.

Based on the cell definition, we can define the classical *cell degree* notion which will be useful during this work.

Definition 3 (cell degree) *The* degree *of a* i*-cell* c *is the number of distinct* (i+1)*-cells incident to* c.

Note that in a n-dimensional space, the degree is only defined for i-cells with $0 \leq i < n$. Indeed, *(n+1)*-cells do not exist in such a space. In this work, we need another notion which is more specific to combinatorial models with multiple incidence: the notion of *local degree*.

Definition 4 (local cell degree) *The local degree of a* i*-cell* c*, is the sum of the number of times of each distinct* (i+1)*-cell* c' *incident to* c*, is incident to* c*.*

Intuitively, this notion corresponds to the classical notion of cell degree, but considering cells locally, that is to say only in a small neighborhood around the cell. To compute the degree of a cell, we count the number of distinct *(i+1)*-cells incident to c, while to compute the local degree we count also the number of times each distinct *(i+1)*-cell is incident to c.

The local degree of a i-cell c is always greater or equal to the degree of the same i-cell, and both values are equal if there is no *(i+1)*-cell several times incident to c. For example, if a vertex v is incident to a loop (a loop is an edge incident twice to the same vertex, by using the notion already given of i-cell and incidence relations in combinatorial map), the degree of v is 1 while the local degree of v is 2.

A combinatorial map only describes the topology of the subdivision, and not its geometry. But it is easy to add some geometric elements to some (even all) orbits of the combinatorial map [6]. The separation of topological and geometrical models is one of the main advantages of the combinatorial map. Indeed, operations become easier to define if the modifications of the topological and the geometrical model are separated. Indeed, we can usually breakdown an operation into two distinct steps: first the topological modifications then the geometrical ones.

2.2 Using Combinatorial Maps for Image Description

Let us here recall some notions. A voxel is a point of discrete space $\mathbb{Z}^3$ associated with a value which could be a color, a grey level.... A three-dimensional image is a finite set of voxels. In this work, we use combinatorial maps to describe voxel sets of *labeled images* having the same values and which are 6-connected. We use the classical notion of 6-connectivity because combinatorial maps cannot describe non-manifold and so the 18 or 26-connectivity cannot be considered without tricks. The label of a voxel is given by a label function $l : \mathbb{Z}^3 \Rightarrow L$ which gives for each voxel its label (a value in the finite set L).

Definition 5 (labeled image) *A* labeled image *is a set of labeled voxels such as two voxels with the same label l belong to the same 6-connected component of voxels with label l.*

[6] One example of a geometrical model associated with a combinatorial map consists in linking to each dart of each topological vertex of the map, the coordinates of an Euclidean space point.

Note that we can describe any type of image simply by labeling all the sets of 6-connected voxels that have the same value, for example with a classical connected component labeling algorithm. Considering only labeled images is an optimization that allows us to retrieve immediately, given the label of a voxel, the connected component it belongs to (but this work can be extended without any problem in order to consider other types of images, and we have already done this in some other works in 2D as for example in [16]).

We speak about *region* for a maximal set of voxels with the same labels.

Definition 6 (region) *A* region *in a labeled image is a maximal set of voxels having the same label.*

Since we consider only labeled images, each region is a 6-connected set of voxels. Two voxels with the same label belong to the same region, and two different regions have two different labels. Note that we use only this property for the inclusion tree (cf. Section 6.2) in order to use the label of regions as a unique identifier.

To avoid any specific processing of voxels in the image border, we consider an infinite region R_0 that surrounds the image. Intuitively, this region contains all the voxels of $\mathbb{Z}^3$ that do not belong to the image. This infinite region allows us to process any type of image, not only rectangular ones without holes. This infinite region is also useful to define the notion of inclusion.

Definition 7 (inclusion) *A region R_i is* included *in a region R_j if and only if any 26-connected path going from a voxel of R_i to a voxel of R_0 pass through R_j.*

Note that each region is at least included in the infinite region, and that this relation is a partial order relation. We need to use the 26-connectivity to define the inclusion relation since it is well known that in order to study the topology of a region R with 6-connectivity, it is necessary to consider the complement of R with 26-connectivity. This is necessary to get a correspondence between the topology of R and the topology of its complement (see for example [33]).

A combinatorial map describes the boundaries of the regions contained in an image. Several studies have been carried out on the notion of boundaries in a discrete image and have shown that using a topology based on the *intervoxel* notion [34,30,32,26,23] enables to define these boundaries so that they verify classical topological properties, such as the Jordan theorem [33].

With the intervoxel framework, an image is not considered only as a matrix of voxels, but as a subdivision of a 3-dimensional space in a set of unit elements: *voxels* are unit cubes of the image, *surfels* the unit squares in between two voxels, *linels* the unit segments in between two surfels (also called *cracks*),

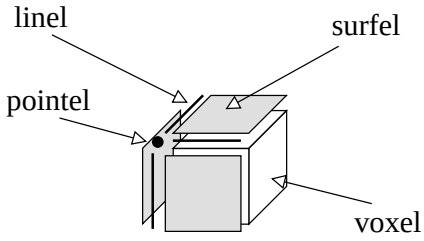


Fig. 3. All the type of elements in the intervoxel 3-dimensional space.

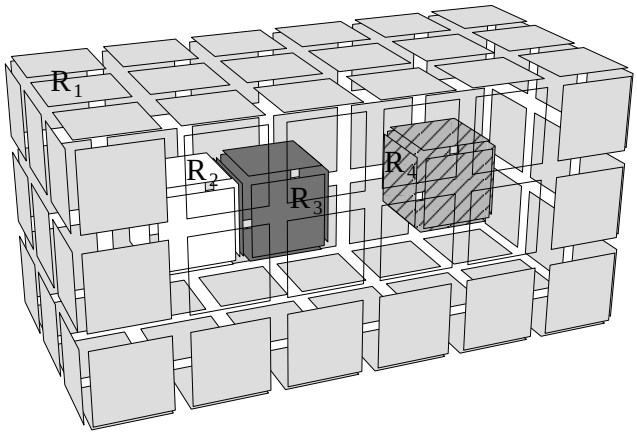


Fig. 4. Intervoxel boundaries. Region R_1 is composed of 3 intervoxel boundaries: an external boundary (light grey surfels, some of them are drawn in wire-frame in order to see the interior of the region) and two internal boundaries: the first one which is around regions R_2 and R_3 and the second one which is around region R_4.

and *pointels* the point in between linels (or *points*). These different types of elements can be seen in Fig. 3.

With these intervoxel elements, the boundary of a region is defined as a set of adjacent surfels (see [34,30,45,27] for more precisions about intervoxel spaces and about discrete surfaces). :

Definition 8 (intervoxel boundary) *An* intervoxel boundary *B of a region R is a maximal connected*[7] *set of surfels such that each surfel of B is between a voxel that belong to region R and a voxel that does not belong to R.*

With this definition, each region R is composed of at least one intervoxel boundary which corresponds to the exterior surface of R, and eventually some intervoxel boundaries which correspond to inner surfaces of R, each one describes a cavity of R (see example in Fig. 4).

[7] A set of surfels S is connected iff for any surfel s_1 and s_2 of the set, there is a path of surfels, all in S, such that each couple of successive surfels of the path are adjacent.

These intervoxel boundaries allow to link combinatorial maps and labeled images.

Definition 9 (Combinatorial map describing a labeled image) *A combinatorial map* M describes *a labeled image* I *iff:*

(1) there is exactly one volume in M *corresponding to each intervoxel boundary of* I*;*
(2) if two regions R_1 and R_2 of I *are adjacent, there is at least one face in* M *which describes this adjacency relation;*
(3) each volume of M *is subdivided in faces, edges and vertices. Let $\#f$ (resp. $\#e$ and $\#v$) the number of faces (resp. edges and vertices), $\#v - \#e + \#f = \chi$ gives the Euler characteristic of the corresponding intervoxel boundary.*

Intuitively, this definition allows to describe the topology of the labeled image. The first item links the volumes of the map with the intervoxel boundaries of the image. Given a region R, it is composed of one volume for its external boundary, and one volume for each cavity. The second item ensures that each adjacency relation is described in the map. The last item guarantees that the topology of each surface is correctly described in the map: this gives the number of tunnels for each volume.

Given a labeled image I, there are several maps which describe I. We want to distinguish a particular combinatorial map among these different maps: this is the *minimal map in number of cells*.

Definition 10 (Minimal combinatorial map describing a labeled image) *A combinatorial map* M *which describes a labeled image* I *is* minimal *iff there is no combinatorial map which describes* I *with a smaller number of cells.*

The main interests of this map is first to allow to retrieve topological information on regions by looking at only a few elements in the map, since the map is minimal. Another advantage concerns the memory space which is much less important for the minimal map than for other maps. The last advantage is to be more representative of the labeled image. Indeed, given a labeled image, the corresponding minimal map does not depend on the geometry of the image and thus is characteristic of the topology of the subdivision.

3 Removal Operations for Combinatorial Maps

The i-dimensional removal operation (noted i-removal) consists in removing an i-cell. This leads to the merging of the two *(i+1)*-cells incident to the

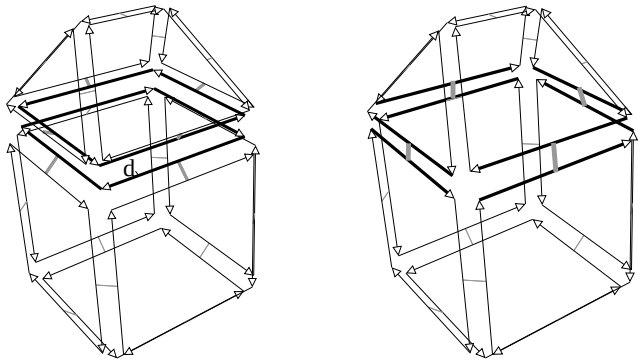


Fig. 5. 2-removal of the face incident to dart d. (A) Initial configuration with two adjacent volumes incident to d. (B) Map obtained after face removal: the two initial volumes are merged.

removed cell. This operation can be applied only if the local degree of the i-cell is two [8] . Indeed, it is not possible otherwise to automatically decide how to connect the different cells around the removed cell.

In 3D, we can remove a face or 2-cell (2-removal, see an example in Fig. 5), an edge or 1-cell (1-removal, see an example in Fig. 6) or remove a vertex or 0-cell (0-removal, see an example in Fig. 7). We only present here the main notions of these operations. A more complete description can be found in [18] where we give general definitions of removal and contraction [9] operations in nD.

In this section, operations are presented in the general framework of combinatorial maps, without any link with images. In this general framework, a combinatorial map can be empty, composed of only one face, or can be composed of several connected components. In the following sections of this paper, we will use these operations to define the topological map and add some additional constraints in order to keep new properties necessary to the new specific framework of image representation (see Section 4).

3.1 Face Removal

We can remove any face in a 3-map without constraint (see example in Fig. 5 where we remove the face incident to dart d). Indeed, in a 3D space, any

[8] If the local degree of an i-cell c is two, c is either a degree two cell with two distinct *(i+1)*-cells, each one incident once to c, either a degree one cell with one *(i+1)*-cell incident twice to c.

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

2-cell is a local degree two cell (by definition of quasi-manifold). Algorithm 1 performs this operation for a given dart d that identifies the face to remove.

Algorithm 1: Face removal

Input: A map M;
A dart d.
Result: M is modified: the face incident to d is removed.

foreach *dart cur of the orbit* $< \beta_1 > (d)$ **do**
  2-sew($\beta_2(cur)$, $\beta_{32}(cur)$);
  Remove darts cur and $\beta_3(cur)$ from M;

This algorithm is local because each dart of the orbit $< \beta_1 > (d)$ is processed independently and without any order. For each dart cur of this orbit, we just 2-sew the darts d_1 and d_2 that were previously 2-sewn to cur and $\beta_3(cur)$. Then, darts cur and $\beta_3(cur)$ can be removed from the combinatorial map since they are now useless. At the end of this algorithm, the two volumes around the removed face are merged into a single volume, and the face is destroyed. We can prove this operation is correct whatever the initial configuration and the face to remove (even in degenerated cases, as for instance the removal of a dangling face adjacent to an unique volume, see [18]).

3.2 Edge Removal

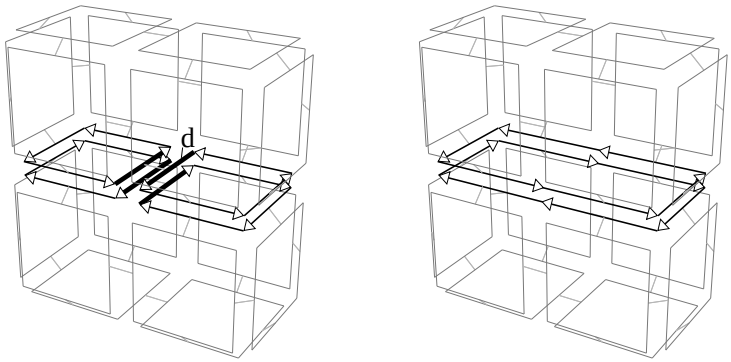
The 1-removal (removal of an edge) is possible only for local degree two edges. We present in Algorithm 2 the edge removal algorithm.

Algorithm 2: Edge removal

Input: A map M;
A dart d incident to a local degree two edge.
Result: M is modified: the edge incident to d is removed.

$d_0 \leftarrow \beta_0(d)$; $d_1 \leftarrow \beta_{21}(d)$; $d_2 \leftarrow \beta_{20}(d)$; $d_3 \leftarrow \beta_1(d)$;
1-sew(d_0, d_1); 1-sew($\beta_3(d_1)$, $\beta_3(d_0)$);
1-sew(d_2, d_3); 1-sew($\beta_3(d_3)$, $\beta_3(d_2)$);
Remove darts d, $\beta_2(d)$, $\beta_3(d)$ and $\beta_{23}(d)$ from M;

We can see in Fig. 6 an example of edge removal. We first sew together the darts which were previously sewn to the edge incident to dart d (the edge is composed of 4 darts, two for each volume). Then, darts d, $\beta_2(d)$, $\beta_3(d)$ and $\beta_{23}(d)$ can be removed from the combinatorial map since they are now useless. As with face removal, we can prove that this operation is correct whatever the



A B

Fig. 6. 1-removal of the edge incident to dart d. (A) Initial configuration with two adjacent faces incident to d. (B) Map obtained after edge removal: the two initial faces are merged.

initial configuration and the edge to remove (even in degenerated cases, as for instance the removal of a loop or a dangling edge, see [18]).

3.3 Vertex Removal

The last possible removal operation is vertex removal. As for edge removal, this operation is only possible for local degree two vertices. We give in Algorithm 3 the vertex removal algorithm.

Algorithm 3: Vertex removal

Input: A map M;
A dart d incident to a local degree two vertex.
Result: M is modified: the vertex incident to d is removed.

$inv \leftarrow 2$; $cur \leftarrow d$; $end \leftarrow \beta_1(d)$;
repeat
  $next \leftarrow \beta_1(\beta_{inv}(cur))$; $d_0 \leftarrow \beta_0(cur)$; $d_1 \leftarrow \beta_1(cur)$;
  1-sew(d_0, d_1); (inv)-sew(d_0,$\beta_0(next)$);
  if $inv = 2$ **then** $inv \leftarrow 3$;
  else $inv \leftarrow 2$;
  Remove dart cur from M;
  $cur \leftarrow next$;
until $cur = end$;

We can see an example in Fig. 7 where we remove the vertex incident to dart d. The operation is performed with a similar algorithm to the one for face removal. But we can see that this algorithm is slightly more complicated. This is due to the inhomogeneous definition of combinatorial maps, where β_1

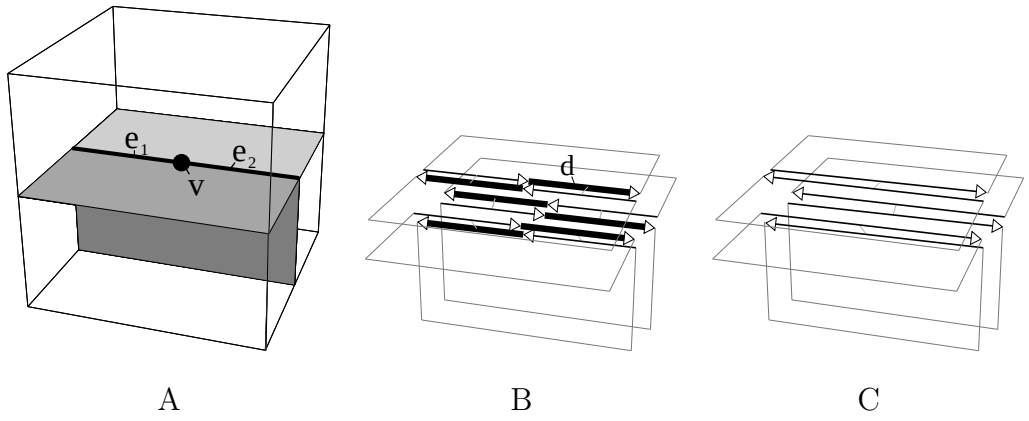


Fig. 7. 0-removal of the vertex incident to dart d. (A) Drawing of the cellular subdivision. We want to remove vertex v incident to edges e_1 and e_2. This vertex is incident to the three faces drawn in grey. (B) Initial configuration with two adjacent edges incident to the vertex identified by d. This figure shows only the three faces incident to the removed vertex. (C) Map obtained after vertex removal: the two initial edges are merged.

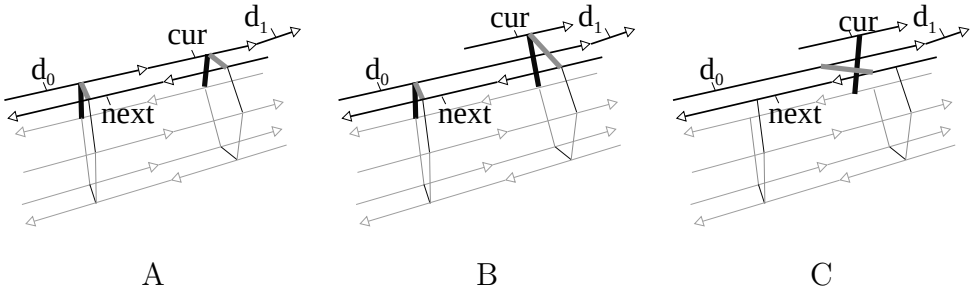


Fig. 8. One step of 0-removal operation where the current dart is labeled *cur*. (A) Initial configuration. (B) Map obtained after 1-sewing(d_0, d_1). (C) Map obtained after 2-sewing(d_0, $\beta_0(next)$).

is a permutation while the others β_i for $i \neq 1$ are involutions. This algorithm is local and processes each dart incident to the removed vertex by following the orbit $< \beta_{21}, \beta_{31} > (d)$ (we apply successively β_{21} then β_{31}). For each dart *cur*, we first 1-sew the dart which is 0-sewn to *cur* (called d_0 in the algorithm) with the dart which is 1-sewn to *cur* (called d_1). This sewing removes locally the vertex incident to d.

But we need also to update the β_2 and β_3 relations. Indeed, the darts that are 2 and 3-sewn to *cur* do not belong to the vertex incident to d, and so they are not removed during the algorithm. But since they were sewn to *cur*, their relations become invalid after the removal of *cur*. We can see in Fig. 8 a zoom on the removed vertex of Fig. 7. This edge is like a bundle of darts, consecutively sewn by β_2 (drawn in grey in this figure) then β_3 (in black on this figure).

In Fig. 8A, we can see the current dart of the algorithm (called *cur*), and

its neighbors for β_0 and β_1 (called d_0 and d_1). First, we 1-sew(d_0, d_1). We obtain the map shown in Fig. 8B. We can observe the problem raised by the two relations β_2 and β_3 if we remove the dart *cur*. We obtain a map where a dart is sewn with a dart that has disappeared (for example $\beta_{02}(next) = cur$). To solve this problem, we 2-sew(d_0, $\beta_0(next)$). We obtain the map shown in Fig. 8C. In the next step of the algorithm, we process the dart *next* and now modify the β_3 relation.

4 Topological Map Definition and Properties

Combinatorial maps are well suited to describe intervoxel boundaries since they describe space subdivisions and all the incidence relations between the different cells. For this reason they were used in 2D in several previous works [24,12,42,7,9,4]. In this paper we extend to 3D the notion of *simplification levels* introduced in 2D in [15,17]. These levels allow us to give a simple and constructive definition of the 3D topological map. The topological map is the last simplification level since it is minimal in number of cells; other levels are only intermediate steps. Note that it is possible to use additional intermediate levels, depending on the particular needs of each application, without modifying the final level (see for example [3,15] where we use five different levels).

Firstly, we only present the topological part of this work without considering the link to a geometrical model. This helps to understand our model and allows us not to mix up the two parts. The link with a geometrical model is the purpose of Section 5. The main idea of our approach is firstly to build a combinatorial map that describes all the intervoxel elements of the image, and then progressively to simplify it as much as possible, while keeping a map which describes the image (in the sense of Def. 9). This construction scheme allows us, in the end, to define the minimal map that describes the image (in the sense of Def. 10). This map describes all the intervoxel boundaries and all the relations of adjacency and incidence between regions. At every step of the process, we only need to verify that each item of Def. 9 is preserved in order to ensure the validity of the new map. To perform the successive simplifications, we use the removal operations presented in Section 3.

4.1 Level 0: Maximal Map

The map of level 0 is the starting point of our process and describes all the intervoxel elements of a given labeled image. Note that this map does not really depend on the labeled image since it encodes the whole set of voxels of

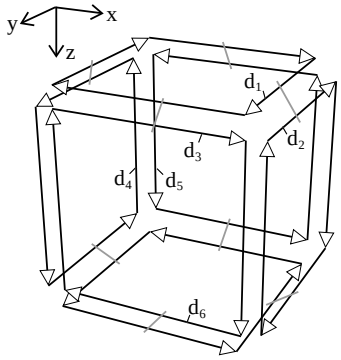
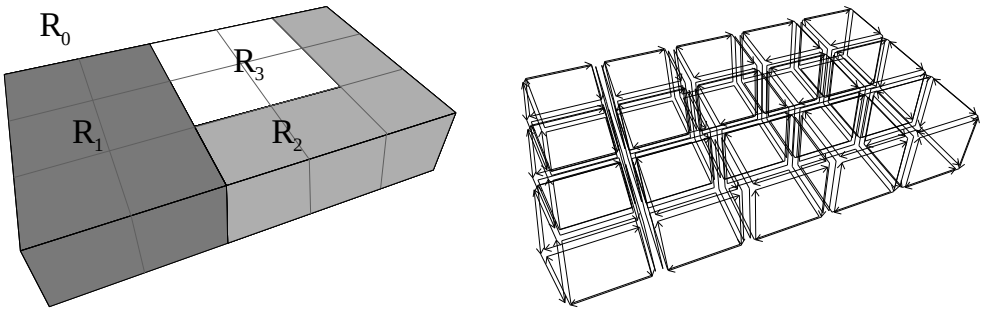


Fig. 9. Creation of a cube in a combinatorial map. A cube is a volume, made of 6 square faces, each one made of 4 darts. The 4 darts of each face are successively 1-sewn, and faces are 2-sewn (represented by little grey segments in the figure). For example, d_1 is 2-sewn with d_2, ans $\beta_1(d_1)$ is 2-sewn with d_3. By using the cartesian coordinate system (x,y,z), the face incident to d_1 (resp. d_2, d_3, d_4, d_5 and d_6) is called the upper (resp. right, front, left, back and bottom) face of the cube.



A B

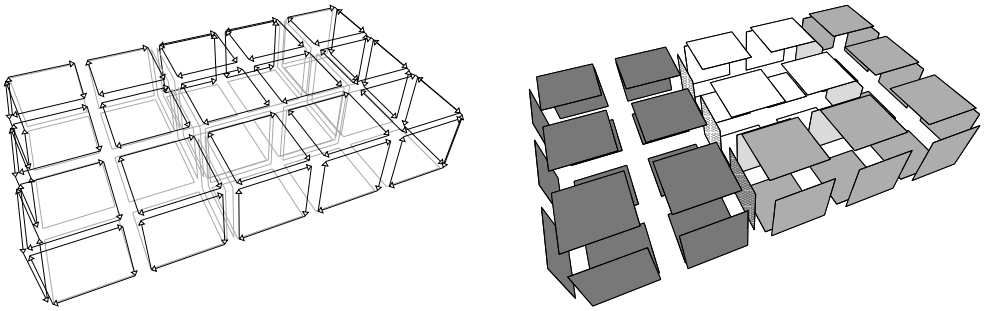
Fig. 10. (A) A 3D image. (B) The corresponding level 0 map (partial drawing without the infinite region).

the image domain.

Definition 11 (level 0 map) *The* level 0 map *corresponding to an* $n_1 \times n_2 \times n_3$ *voxel labeled image, is the map having* $n_1 \times n_2 \times n_3$ *cubes 3-sewn between them, each cube corresponding to a voxel, plus an enclosing volume which describes the infinite region.*

We can see in Fig. 9 the way a cube is created in the map in order to represent a voxel.

Fig. 10B shows the level 0 map of the labeled image shown in Fig. 10A (in this image, different labels are drawn with different grey levels). For an $n_1 \times n_2 \times n_3$ image, this map is composed of $(n_1 \times n_2 \times n_3) + 1$ volumes. $n_1 \times n_2 \times n_3$ cubes, each one describing one voxel of the image, plus an additional volume that describes the infinite region. In this paper, most of the time, we do not draw the infinite region in order to make figures clearer and more understandable.



A B

Fig. 11. (A) Level 1 map. (B) The corresponding intervoxel boundaries.

4.2 Level 1 Map

The level 0 map describes all the intervoxel elements of an image. In order to obtain a combinatorial map which describes the image, it is necessary to remove all the surfels that are between two voxels having the same label. This operation is achieved by using the face removal operation for each face corresponding to such surfels (these faces are called *inner faces*, i.e. faces incident to the same region twice).

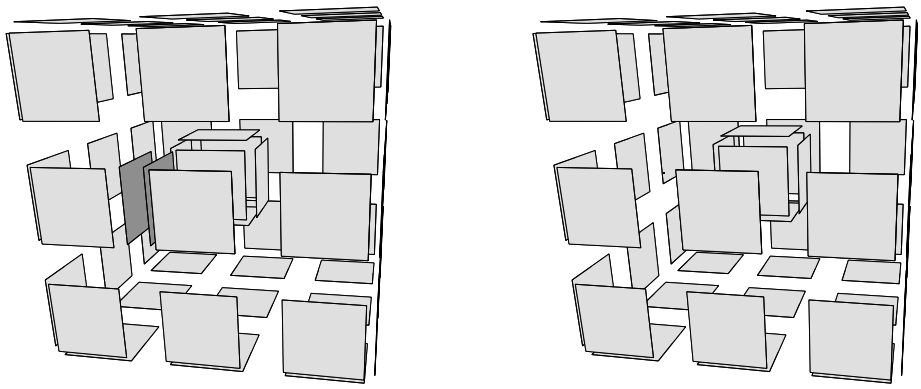
Definition 12 (level 1 map) *The* level 1 map *is the map obtained from the level 0 map by removing each face between two voxels having the same label.*

We can see in Fig. 11A, the level 1 map of our 3D image, and in Fig. 11B, the corresponding intervoxel boundaries. We can verify that each face of the level 1 map corresponds exactly to one surfel of an intervoxel boundary. Indeed, each surfel between two voxels having different labels belongs to an intervoxel boundary (see Def. 8) and the level 1 map describes all of these surfels since level 0 describes all surfels of the subdivision and between level 0 and level 1, we have only removed inner faces.

4.3 Inclusion Tree

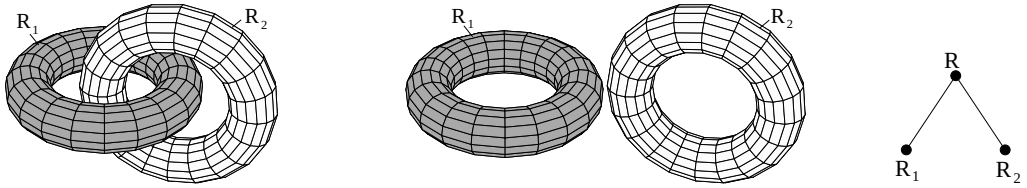
The level 0 map describes each intervoxel element of the image. This map is obviously connected, but can be disconnected during the construction of level 1 (see example in Fig. 12).

This type of disconnection occurs during face removal, if a volume is included in another one. That is why we call this kind of disconnection a *volume disconnection*. In this case, we have lost the topological information that allows to position each connected component relatively to each other. Indeed, each region is described in the level 1 map by an external boundary and zero of



A B

Fig. 12. (A) Level 1 map during its construction. At this step, this map is connected. (B) After the 2-removal of the dark face, the map is now made up of two distinct connected components.



A B C

Fig. 13. Problem of interlaced rings. Both configurations (A) and (B) are described with the same inclusion tree shown in (C) (R being the region which contains both regions R_1 and R_2). The inclusion tree does not contain information on interlacing.

several internal boundaries.

In the example shown in Fig. 12, the $1 \times 1 \times 1$ cube is included in the $3 \times 3 \times 3$ cube. During the level 1 map construction, we remove progressively inner faces. After some removals, the current map can be the one shown in Fig. 12A which is connected. After the removal of the dark grey face, we obtain the map shown in Fig. 12B which is made up of two distinct connected components (one for the external boundary of the $3 \times 3 \times 3$ cube, and a second one for the internal boundary which corresponds to the $1 \times 1 \times 1$ cube).

We need to keep additional information in order to place the different connected components. For that, we introduce a region inclusion tree (more precisely described in Section 6.2). This tree has one node for each region of the image. Its root is the infinite region R_0, 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 interior boundaries of a given region.

Note that the problem of interlaced rings (e.g. Fig. 13) is not taken into

account by the inclusion tree. Indeed, if two rings R_1 and R_2 are interlaced, and included in a region R, the inclusion tree does not contain information about the interlacing. Thus, we cannot differentiate this case (Fig. 13A) and the same case where the two rings are not interlaced (Fig. 13B). Note that even if this information is not contained in the topological map, it can be retrieved by using the shape of the volumes and intersection algorithms (by using for example [25]).

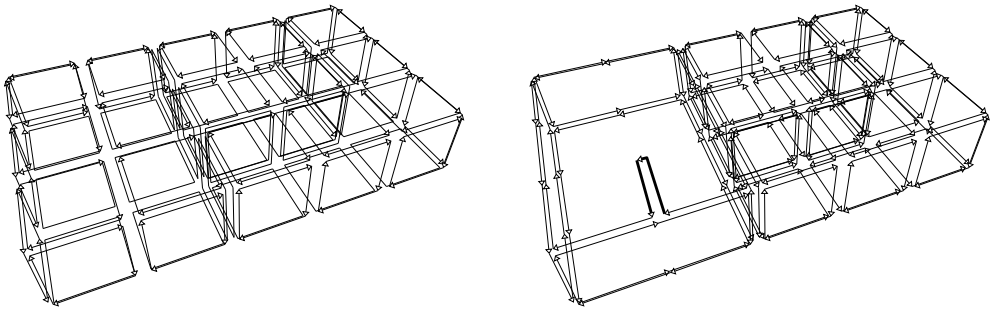
4.4 Level 2 Map

The level 1 is the first map of our constructive definition that describes the labeled image [10] . But this level contains too many darts which are not useful to describe adjacency relations between regions. To obtain the minimal map, we decrease the number of cells that describe the intervoxel boundaries. Firstly, we merge faces by using edge removal. For that, we need to remove local degree two edges, and for two reasons: firstly because it is not possible to remove edges with local degree higher than two due to the precondition of the 1-removal operation. Secondly, this is in order not to depend on the order of removal operations.

When we remove a degree two edge, this merges the two incident faces which describe the same adjacency information between the same two regions R_i and R_j (otherwise the degree of the edge is not equal to two). But this is not enough to obtain the minimal model. Indeed, a degree two edge can become a degree one edge after some edge removals. We can see, in Fig. 14A, the level 1 which is the starting point of the level 2 map, and in Fig. 14B, the level 2 map during its construction. In this map, we have removed some local degree 2 edges but not all of them yet. The bold edge in this figure is a degree one edge (incident twice to the same face), but was a degree two edge in the level 1 map. We need to remove this edge in order not to depend on the order of removal operations. Moreover, this edge does not describe an adjacency relation and so has to be removed to obtain the minimal map.

Edge removal can involve a disconnection (called *face disconnection*) when we remove a degree one edge (indeed, the 1-removal of a degree two edge cannot lead to a disconnection). In such a case, before the removal, a face is described in the map by an orbit $< \beta_1 > (d)$, and this orbit is cut into two different orbits $< \beta_1 >$ after the removal. This is a problem because this disconnects the region into two connected components and thus the first point of Def. 9 is no longer verified (there is exactly one volume in M corresponding to each intervoxel boundary of I) since there are two volumes for the same intervoxel boundary.

[10] Indeed, the level 0 map describes all the intervoxel elements of the image.



A B

Fig. 14. (A) Level 1 map. (B) Level 2 map during its construction.

In order to solve this problem of disconnection, we keep an edge which links the two distinct orbits. This edge is a degree one edge which is normally removed during level 2 building, and which is only kept now to avoid a disconnection. Moreover this edge does not describe the boundary of a face unlike the other edges of the level 2 map. For this reason, we call this edge a *fictive edge* and on the contrary we call the other edges *real edges*.

Definition 13 (fictive edge) *A* fictive edge *is a degree one edge whose removal involves a face disconnection or completely removes a face.*

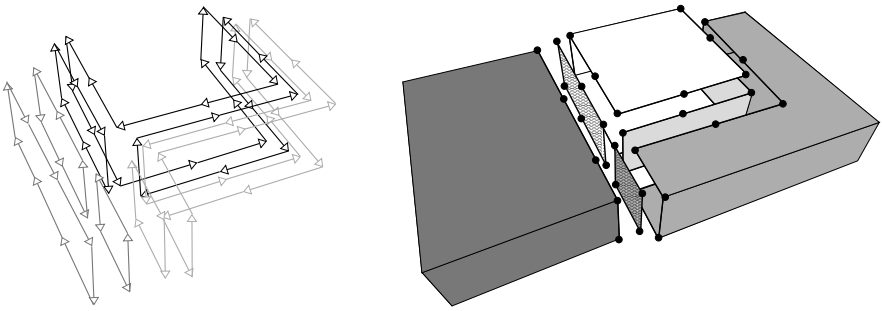
We need to add the second condition (completely removes a face) to avoid the entire disappearance of a face composed of a single edge (which is the case of a minimal subdivision of the sphere). Indeed, in such a case, this involves losing an adjacency relation by completely removing the face.

With these fictive edges, each face is always connected. This can be proved by showing that each face is homeomorphic to the unit disk and thus the problem of face disconnection is now solved (see Section 4.7).

Definition 14 (level 2 map) *The* level 2 map *is the map obtained from the level 1 map by removing successively each local degree two edge that does not involve a face disconnection and that does not completely remove a face.*

With this definition, we are sure that a fictive edge is never removed, thanks to the additional conditions "does not involve a face disconnection" and "does not completely remove a face".

We can see the level 2 map of our image in Fig. 15. In this map, the number of faces is minimal (because the topological map is minimal in number of cells, and the number of faces is constant between level 2 and level 3 maps, see Section 4.7 for proofs). Indeed, we cannot remove an additional edge, since the local degree of each real edge is higher than two, and it is not possible to remove a fictive edge since this involves a face disconnection or completely removes a face. Moreover, we can show that this map still represents the



A B

Fig. 15. (A) Level 2 map. (B) The corresponding subdivision.

labeled image, which proves that no information is lost by our simplifications (see Section 4.7 for proofs).

4.5 Level 3 Map

The level 2 map has the minimal number of faces. To obtain the minimal map in number of cells, we decrease the number of edges (and therefore the number of vertices since these two numbers are linked [11]) by using vertex removal.

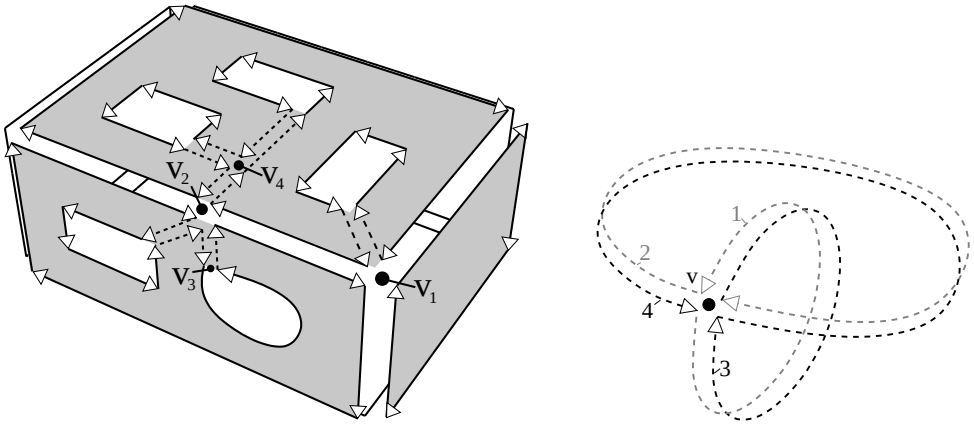
We can remove each degree two vertex incident to two non-loop edges without modifying the topology of described regions. Indeed, given such a vertex, both incident edges describe the same adjacency information and they can be merged into a single edge. But this is not enough to obtain the minimal map. Indeed, keeping a fictive edge during level 2 construction modifies the degree of the incident vertex, and can prevent its removal during level 3 construction. That is unsatisfactory because we do not obtain the minimal map: we keep some vertices that could eventually be removed depending on the configuration of the fictive edges.

In order to solve this problem, we need to consider the degree of each vertex without taking into account fictive edges. This is the notion of *real degree*.

Definition 15 (real degree of a vertex) *The* real degree *of a vertex* v *is the number of distinct real edges incident to* v.

Note that the real degree of a vertex is always less or equal to the degree of the same vertex, and that both values are equal if there is no fictive edge incident to the vertex.

[11] When we remove a degree two vertex, this decreases the number of vertices by one, and decreases also the number of edges by 1 since the two incident edges are merged.



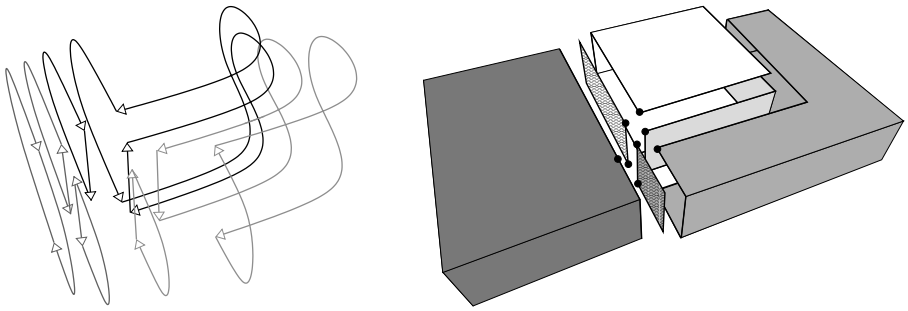
A B

Fig. 16. Different types of vertices depending on their real degrees (fictive edges are drawn with dotted lines). (A) There are, for example, two vertices to remove: v_2 which is a real degree two vertex and v_4 which is a real degree zero vertex, and two vertices to keep: v_1 which is a real degree three vertex and v_3 which is a real degree one vertex. (B) In this map, the only vertex v is incident to 2 loops which are fictive edges. This vertex must not be removed. We can verify that each edge of this map is fictive by simulating edge removal. For example, if we remove the edge $\{1, 3\}$, we obtain the map made up of the two darts $\{2, 4\}$ with $\beta_1(2) = 2$ and $\beta_1(4) = 4$. Before edge removal, we have only one orbit $< \beta_1 >$ which is cut into two orbits after removal.

Now we can consider each vertex v depending on its real degree d:

- $d > 2$ (there are at least 3 real edges incident to v): v cannot be removed (case of vertex v_1 in Fig. 16A) due to the precondition of 0-removal;
- $d = 2$ (there are two real edges and eventually some fictive edges incident to v): v needs to be removed, if both incident real edges are not loops, in order not to depend on the fictive edge position. But vertex removal is defined only for local degree two vertex and cannot be applied for a real degree two vertex v if some fictive edges are incident to v. In order to use vertex removal, firstly we shift all the fictive edges incident to v on a neighbor vertex of the same face. After this operation, the local degree of v is now two and we can apply the vertex removal operation (case of vertex v_2 in Fig. 16A).

  If one real edge incident to v is a loop, v must not be removed since the loop corresponds to a face and thus describes adjacency information. If we remove v, this involves the disappearance of the face described by the loop and thus this means losing some topological information.
- $d = 1$ (there is only one real edge and eventually some fictive edges incident to v): the real edge is a loop and v has to be kept (case of vertex v_3 in Fig. 16A) for the same reason as the previous case.
- $d = 0$ (there is no real edge incident to v, so there are only fictive edges): v needs to be removed if at least one incident fictive edge is not a loop. Indeed,



A B

Fig. 17. (A) Level 3 map. (B) The corresponding subdivision.

such a vertex does not describe topological information, since a fictive edge is only necessary in order to solve the problem of face disconnection. If we shift all the fictive edges incident to v except the one which is not a loop, we keep the connectivity. Then the degree of v is now one and it can be removed without loss of any topological information (case of vertex v_4 in Fig. 16A).

If v is incident to only one edge, or if v is only incident to $2k$ loops (case of vertex v in Fig. 16B), v is not removed since this case corresponds either to the case of the sphere, or to the minimal representation of a torus with k holes. If we remove v, this involves the removal of a face and thus means losing topological information.

Note that we can shift fictive edges only if there is a non-loop edge which is not shifted (see the cases above for real degree zero or two vertices). This edge gives the direction to shift the fictive edges. The fact that this edge is not a loop guarantees that fictive edges are moved into a different vertex.

Now we can define the level 3 map by using fictive edge shifting to remove real degree two vertices:

Definition 16 (level 3 map) *The* level 3 map *is the map obtained from the level 2 map by removing successively:*

- *each real degree two vertex* v *incident to two non-loop edges, after having shifted all fictive edges incident to* v*;*
- *each real degree zero vertex* v *incident to at least one non-loop edge* e*, after having shifted all fictive edges incident to* v*, except* e*.*

With this definition, the level 3 map is the minimal map which describes the initial image (see Section 4.7 for proof). We can see the level 3 map of our image in Fig. 17. Now, this map also has the minimal number of edges because it is impossible to remove additional vertex without losing any information. This map is composed of 2 vertices, 4 edges, 6 faces and 4 volumes (if we count the infinite region).

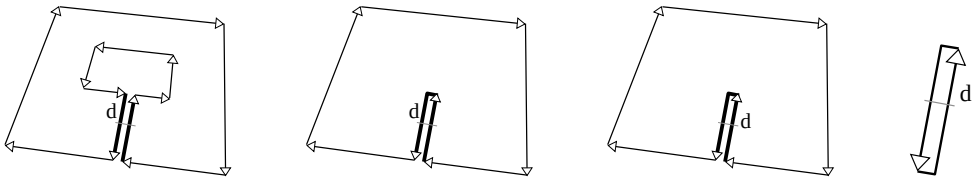


Fig. 18. The four possible configurations of a face around a degree one edge by considering the degree of the two vertices incident to the edge. Only the first case involves a face disconnection when we remove the edge incident to d. (A) Both degrees are greater than 1. $\beta_0(d) \neq \beta_2(d)$ and $\beta_1(d) \neq \beta_2(d)$. (B) and (C) are the same configuration but with different positions of dart d. One degree is greater than 1 and the other is equal to 1. (B) $\beta_0(d) = \beta_2(d)$ and $\beta_1(d) \neq \beta_2(d)$. (C) $\beta_1(d) = \beta_2(d)$ and $\beta_0(d) \neq \beta_2(d)$. (D) Both degrees are equal to 1. $\beta_0(d) = \beta_2(d)$ and $\beta_1(d) = \beta_2(d)$.

The *topological map* of a labeled image is composed of the level 3 map of the image, plus the inclusion tree of regions contained in the image.

Definition 17 (topological map) *The* topological map *corresponding to a labeled image* I*, is composed of:*

(1) the level 3 map corresponding to I*;*
(2) the inclusion tree of regions contained in I.

This map is called a *topological map* since it describes all the topological information in a minimal way (minimal in number of cells). Moreover, this map depends only on the topology of the described image and not on its geometry.

4.6 Fictive Edge Management

A fictive edge is a degree one edge. We can see in Fig. 18 all the possible configurations of a degree one edge, considering the degree of its both extremities. For each vertex v incident to the edge, there are two cases to consider: if the degree of v is one or if it is greater than 1. This gives four possible cases. In the first case (Fig. 18A), the removal of the edge incident to d involves a face disconnection. The two next cases cannot disconnect the face and so the edge can be removed. The last case (Fig. 18D) does not involve a face disconnection but the entire disappearance of the face since it was composed of one edge only.

Therefore, the only case of face disconnection is if we remove a degree one edge with its two incident vertices having a degree greater than one (case shown in Fig. 18A). Detecting this case can be achieved by testing if d (a dart of the edge to remove) and $\beta_2(d)$ belong to the same face (for example by exploring

the face, starting from d and searching for $\beta_2(d)$). Moreover, testing the degree of both vertices can be achieved locally by verifying if $\beta_0(d) \neq \beta_2(d)$ and if $\beta_1(d) \neq \beta_2(d)$ (see Fig. 18A).

Algorithm 4 gives the algorithm that tests if an edge has to be removed or not. The first test of this algorithm ($\beta_{23}(d) \neq \beta_{32}(d)$) allows to test if the local degree of the edge incident to d is greater than two. In that case, the edge is not removed and so the algorithm returns false.

Algorithm 4: Test if an edge has to be removed

Input: A map M;
A dart d.
Output: true iff the edge incident to d has to be removed.

```
if β23(d) ≠ β32(d) then return false;
if β0(d) = β2(d) then
    if β1(d) = β2(d) then return false;
    else return true;
if β1(d) = β2(d) then return true;
if β2(d) belongs to the orbit < β1 > (d) then
    return false;
return true;
```

Then, we know that we have a local degree two edge. The three following tests identify the three configurations shown in Fig. 18B, Fig. 18C and Fig. 18D. For the cases of Fig. 18B and Fig. 18C, we return true since dangling edges must be removed, and for the last case we return false to avoid the entire disappearance of a face.

After these tests, we have either a degree two edge that needs to be removed, or a degree one edge with a configuration similar to Fig. 18A that has to be kept and which is a fictive edge. In order to distinguish these two cases, we have to test if d and $\beta_2(d)$ belong to the same $< \beta_1 >$ orbit or not (this is the test which allows to find the fictive edges. For example, in Fig. 16B, the edge $\{1, 3\}$ is fictive because $\beta_2(3) = 1$ and $1 \in < \beta_1 > (3) = \{1, 2, 3, 4\}$, and thus Algorithm 4 return false: the edge cannot be removed).

To distinguish the two cases, it can be possible to check all the darts of the orbit $< \beta_1 > (d)$. If one of them is equal to $\beta_2(d)$, then the edge is a degree one. Otherwise, it is a degree two. Another possibility (which is the one chosen in this work) is to use union-find trees [47] to improve the complexity. We link each face of the map to a union-find tree which describes the face (the tree is created and linked to each surfel in the level 0 map). When we remove an edge, we just merge both corresponding trees. With this technique, testing if d and $\beta_2(d)$ belong to the same $< \beta_1 >$ orbit is simply achieved by testing if

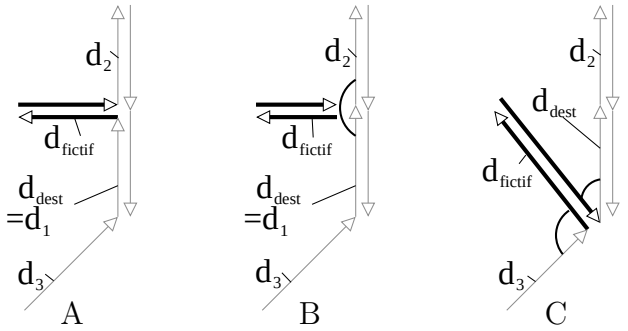


Fig. 19. The two steps of the algorithm that shift one fictive edge. (A) The initial configuration. (B) 1-sewing of d_1 and d_2. (C) Sewing of the edge incident to d_{fictif}.

find(d) equals *find($\beta_2(d)$)*.

With union-find trees, the complexity of this algorithm can be bounded by 5 (indeed, it is well known that the amortized cost of a series of m union-find operations on n elements can be done in time $O(n.\alpha(m,n))$ with $\alpha(m,n)$ being the inverse Ackermann function, which is less than 5 in practical cases. See [47] for the demonstration about the complexities). Moreover, in several cases (edge with a local degree greater than two or edges in particular configurations) our algorithm has $O(1)$ complexity since we do not need to make the last test in Algorithm 4.

Algorithm 5 allows to shift the fictive edge incident to the dart d_{fictif}. This edge is shifted between the darts $\beta_0(d_{dest})$ and d_{dest}. The edge is shifted in two steps as we can see in Fig. 19.

Algorithm 5: shiftOneFictiveEdge

Input: A map M;
A dart d_{fictif} incident to a fictive edge;
A dart d_{dest} incident to a non-loop edge.

Result: M is modified: the edge incident to d_{fictif} is shifted between $\beta_0(d_{dest})$ and d_{dest}.

$d_1 \leftarrow \beta_0(d_{fictif})$; $d_2 \leftarrow \beta_{21}(d_{fictif})$; $d_3 \leftarrow \beta_0(d_{dest})$;

1-sew(d_1, d_2); 1-sew$(\beta_3(d_2), \beta_3(d_1))$;

1-sew(d_3, d_{fictif}); 1-sew$(\beta_2(d_{fictif}), d_{dest})$;
1-sew$(\beta_3(d_{fictif}), \beta_3(d_3))$; 1-sew$(\beta_3(d_{dest}), \beta_{23}(d_{fictif}))$;

In order to shift all the fictive edges incident to a given vertex v, it is only necessary to apply Algorithm 5, which shifts a given edge, for each fictive edge incident v. This is achieved by Algorithm 6.

Proposition 18 *Given a map* M *and a dart* d *incident to a non-loop edge* e*, and to vertex* v*, at the end of Algorithm 6, there is no fictive edge incident to*

Algorithm 6: Shifting of all the fictive edges incident to a given vertex

Input: A map M;
A dart d incident to a non-loop edge e, and to vertex v.
Result: M is modified: all the fictive edges incident to v, except e, are shifted on the vertex incident to $\beta_2(d)$.

```
actu ← d ;
repeat
    toShift ← β02(actu) ;
    while isFictiveEdge(toShift) and toShift ≠ actu do
        next ← β02(toShift);
        shiftOneFictiveEdge(toShift, β1(actu)) ;
        toShift ← next;
    actu ← β23(actu) ;
until actu = d ;
```

v, e *expected.*

PROOF. Algorithm 6 walks through the darts of the edge incident to the given dart d, and for each dart to shift all the fictive edges incident to the considered vertex by using *shiftOneFictiveEdge* algorithm.

Given a dart *actu* of the edge e, the second loop starts from the dart $\beta_{02}(actu)$, in order not to shift the initial edge incident to *actu*. If the incident edge is fictive, this edge is shifted on the second vertex of the non-loop edge e. This decreases the number of fictive edges incident to e by one.

At the end of the second loop, there is no fictive edge incident to v (e excepted) in the face incident to *actu*. Indeed, this loop is stopped when we obtain either a non-fictive edge, or go back to the starting dart. Note that the last case only occurs when the considered vertex has a real degree equal to zero. Indeed, otherwise, we are sure that there are two real edges incident to v.

Since the first loop runs through all the darts incident to edge e, we apply the second loop on each face incident to e. Thus, at the end of the algorithm, there is no fictive edge incident to v, e excepted. □

Note that we can shift all the fictive edges incident to a vertex without involving any topological modification, since the position of a fictive edge is not important. Indeed, the role of a fictive edge is only to keep each face homeomorphic to the unit disk, and fictive edge shifting keeps this property.

In this section, we study and prove the main properties of the topological map. The first property is that topological map describes the labeled image (in the sense of Def. 9). For that, we prove each part of Def. 9: there is exactly one volume in the map corresponding to each intervoxel boundary of the image; when two regions are adjacent, there is at least one face in the map which describes this adjacency relation; and each volume of the map is subdivided into faces, edges and vertices in such a way that the Euler formula gives the Euler characteristic of the corresponding surface. The second property is that the topological map is minimal in number of cells, according to Def. 10. We also prove that each face of the map is homeomorphic to the unit disk. This proves that the face disconnection has not occurred as if a face is disconnected into two orbits $< \beta_1 >$, it is not homeomorphic to the unit disk because one of the two orbits describes a hole in the face. Lastly, we prove that the topological map does not depend on the shape of regions contained in the image. This last property is particularly interesting if we want to use a topological map to characterize the topology of regions. To prove these properties, we often use the fact that the volumes are constant between level 1 and level 3 maps and that the faces are constant between level 2 and level 3 maps. Thus we start by proving these two properties.

Proposition 19 *Given a labeled image* I*, there is a bijection between faces of the level 2 map corresponding to* I *and faces of the level 3 map.*

PROOF. To prove this proposition, we have to prove that between level 2 and level 3, there is no face creation, nor face destruction. To do so, we show firstly that for each face of the level 2 map, there is no face disconnection. Indeed, such disconnection involves breaking an orbit face in two, and thus a face of the level 2 map becomes two faces in the level 3 map. Then, we show that no pair of faces are merged between the two levels. Lastly, we show that no face disappeared directly. Due to the level 3 map definition, we are sure that no face is directly created and thus this case cannot occur.

Firstly, there is no face disconnection. Indeed, to build the level 3 map, we remove each real degree zero and two vertex (see Def. 16). This cannot involve a disconnection, since in both cases we remove a degree one or two vertex, an operation that cannot involve a disconnection. Moreover, the fictive edge shifting operation cannot involve a disconnection either, since each edge is only shifted along a non-loop edge.

Secondly, no pair of faces of the level 2 map can be merged in the level 3 map. This can be directly proved since to build the level 3 map, the only

operations used are vertex removal and fictive edge shifting, and neither of these operations can involve merging two faces.

Lastly, no face of the level 2 map can disappear in the level 3 map. Indeed, we remove vertices and thus a face can disappear only if it is composed of a vertex only. In such a case, the incident edge is a loop and the vertex is not removed by level 3 map definition and the additional condition (see Def. 16). □

Proposition 20 *Given a labeled image* I*, there is a bijection between volumes of the level 1 map corresponding to* I *and volumes of the level 2 map (resp. level 3 map).*

PROOF. The proof is similar to the previous one but now for volumes. We have to prove that between level 1 and the two following levels, there is no volume creation, nor volume destruction. To do so, we show firstly that for each volume v of the level 1 map, there is no volume disconnection. Indeed, this would involve breaking an orbit volume in two, and thus a volume of the level 1 map becomes two volumes in following levels. Then, we show that no pair of volumes are merged. Lastly, we show that no volume disappeared directly.

Firstly, there is no volume disconnection. A volume can be disconnected in two orbits, even directly, or by cutting a face in two parts. Firstly, the face disconnection is avoided in the level 2 map construction by definition of this map (see Def. 14) and this is not possible for level 3 maps as we have seen in the previous proof. Secondly, the volume disconnection can occur directly only by using face removal, an operation which is not used to build level 2 and level 3 maps.

Secondly, no pair of volumes of the level 1 map can be merged in the following levels. This can be directly proved since to build level 2 and level 3 maps, the operations used are edge removal, vertex removal and fictive edge shifting, and none of these three operations can involve merging two volumes.

Lastly, no volume of the level 1 map can disappear in the following levels. For the level 2 map, we remove edges sequentially and thus, a volume can disappear only if it is composed of an edge only, and this case is avoided by the level 2 definition (see Def. 14). For level 3, we remove vertices and thus a volume can disappear only if it is composed of just a vertex. In such a case, there is at least one edge of the volume, and this edge is a loop. But in such a case, the vertex is not removed by the level 3 map definition (see Def. 16). □

Proposition 21 *In the topological map corresponding to a labeled image* I*, each volume corresponds to an intervoxel boundary of* I*.*

PROOF. The proof is achieved in two steps: firstly we prove that the property is true for the level 1 map, and secondly that it is kept for other levels.

Let us take the level 1 map corresponding to I, and let us suppose that the property is not true. We show that this leads to a contradiction with the level 1 map definition. There are two possible cases in which the property is not true: firstly if two different volumes v_1 and v_2 describe a same intervoxel boundary b of a region R, and secondly if a volume v describes two different intervoxel boundaries b_1 and b_2 of a region R. The case where a volume describes two different intervoxel boundaries of two different regions is impossible since each volume of a level 1 map is obtained by merging adjacent cubes of a same region from the level 0 map.

Let us suppose that two different volumes v_1 and v_2 describe a same intervoxel boundary b of a region R. These two volumes are necessarily adjacent since they describe the same intervoxel boundary, which is a connected set of surfels. Moreover, each face between these two volumes is an inner face (i.e. a face incident twice to the same region) since volumes v_1 and v_2 are obtained by merging voxels of R. This shows a contradiction with the level 1 map definition where each inner face is removed.

Now, let us suppose that a volume v describes two different intervoxel boundaries b_1 and b_2 of a region R. In this case, some faces of v describe b_1 and other faces describe b_2. Since b_1 and b_2 are not connected (by definition of intervoxel boundary) and since v is connected (otherwise there is more than one volume), there is at least one face of v which links the two boundaries b_1 and b_2, and this face is an inner face. This shows also a contradiction with the level 1 map definition where each inner face is removed.

This proves the Prop. 21 for the level 1 map. Thus, this property is also true for level 2 and level 3 maps. Indeed, by using Prop. 20, we know that there is a bijection between volumes of the level 1 map and the ones of the level 2 and level 3 maps, and this proves the property for the other levels. This proves the proposition for the topological map. □

Proposition 22 *Let* I *be a labeled image and* M *the corresponding topological map. If two regions R_1 and R_2 are adjacent in* I*, there is a face in* M *between one volume of R_1 and one volume of R_2.*

PROOF. This property is obviously true in the level 0 map corresponding to I since this map contains all the faces corresponding to a surfel. If regions

R_1 and R_2 are adjacent, there is at least one surfel between these two regions and this surfel is present in the level 0 map.

This property is always true in level 1 maps since removed faces between the two levels are inner faces, and faces which describe adjacency between R_1 and R_2 are not inner faces.

This property is preserved for level 2 maps since when we remove an edge e during the level 2 construction, e is either a dangling edge, or a degree two edge (the two cases in which the local degree is two and the removal does not involve a face disconnection or completely remove a face). If e is a dangling edge, this cannot involves the disappearance of the incident face by definition of level 2 where this case is avoided. If e is a degree two edge, this cannot involves losing a face which describes an adjacency relation. Indeed, in such a case, the two faces incident to e are between the same two regions R_1 and R_2 (otherwise the degree of e would be greater than 2). These two faces are merged into one face by the removal of e, and this new face is between one volume of R_1 and one volume of R_2 which proves the property.

Lastly, the property is always true for level 3 maps since there is a bijection between faces of level 2 and faces of level 3 (see Prop. 19). This proves the proposition for the topological map. □

Proposition 23 *In the topological map* M *corresponding to a labeled image* I*, each volume of* M *is subdivided into faces, edges and vertices. Let* $\#f$ *(resp.* $\#e$ *and* $\#v$*) the number of faces (resp. edges and vertices),* $\#v - \#e + \#f = \chi$ *gives the Euler characteristic of the corresponding surface.*

PROOF. Firstly, let us consider each volume v of the level 1 map corresponding to I. Such a volume is subdivided in faces, edges and vertices (with $\#f$, $\#e$ and $\#v$ the number of faces edges and vertices), where each face is a square corresponding to a surfel, and the volume corresponds to an intervoxel boundary of I. It is well known that the Euler characteristic of a polyhedron can be computed by using the Euler formula $\#v - \#e + \#f = \chi$ (see for example [31]), and the surface of v in the level 1 map is a polyhedron. To prove the proposition, we show that the different simplifications made during topological map construction do not modify the Euler characteristic of the corresponding surface.

During the construction of level 2 maps, we remove each local degree two edge that does not involve a face disconnection and that does not completely remove a face. When we remove a degree two edge, we merge two distinct faces into a single one, so the number of faces decreases $\#f' = \#f - 1$, one edge is removed so the number of edges decreases $\#e' = \#e - 1$ and the number of

vertices remains unchanged. Therefore, the new Euler characteristic is equal to the old one since we have $\chi' = \#v' - \#e' + \#f' = \#v - (\#e - 1) + (\#f - 1) = \#v - \#e + \#f = \chi$. When we remove a degree one edge (with no disconnection), the number of faces does not change $\#f' = \#f$, the number of edges decreases $\#e' = \#e - 1$ and the number of vertices decreases since there is exactly one vertex of $\#e$ which is incident only to the removed edge (see possible cases in Section 4.6). This vertex is thus removed by the operation and $\#v' = \#v - 1$. That is why the new Euler characteristic is again equal to the old one.

During the construction of level 3 maps, we use two operations. Firstly fictive edge shifting, and secondly vertex removal. Fictive edge shifting does not modify anything for the Euler characteristic, since the number of cells (vertices, edges and faces) remains unchanged. The second operation is the removal of degree two vertices. For this operation, the number of faces is not modified $\#f' = \#f$, the number of edges decreases since the vertex removal involves the merging of the two incident edges into a single one $\#e' = \#e - 1$ and the number of vertices decreases since one vertex is removed $\#v' = \#v - 1$. So the new Euler characteristic is again equal to the old one. □

Proposition 24 *The topological map corresponding to a labeled image* I *is minimal in number of cells.*

PROOF. The topological map is minimal in number of cells. Firstly it is minimal in number of volumes, since each volume corresponds to an intervoxel boundary of I (see Prop. 21).

Secondly each volume is minimal in number of faces. Let us suppose it is not the case, i.e. the number of faces is not minimal. We have thus two adjacent faces that can be merged into a single face (otherwise the number of faces is minimal). In that case, the edge between these two faces is a local degree two edge and this is in contradiction with the definition of level 2 maps where all local degree two edges are removed.

A similar argument can be used to prove that the number of edges is minimal. Indeed, during the construction of level 3 maps, we only keep vertices with real degree equal to one or greater than two. For the same reason, the number of vertices is also minimal since no vertex of the topological map can be removed. □

Proposition 25 *In the topological map corresponding to a labeled image, each face is homeomorphic to the unit disk*[12] .

[12] The unit disk B^2, called also 2-balls, is the set of all points $x \in \mathbb{R}^2$ for which $||x|| == 1$, see [41].

PROOF. The level 0 map describes all the intervoxel elements of an image and not the image partition. By removing each inner face, we obtain level 1 where each face describes a surfel, and is thus obviously homeomorphic to the unit disk.

During the construction of level 2 maps, we remove each local degree two edge that does not involve a face disconnection and that does not completely remove the face. Let us consider the two types of removed edges during the construction of level 2:

- degree two edge: in this case, there are two distinct faces incident to the removed edge. These two faces, which are homeomorphic to the unit disk, are merged into a single face which is obviously homeomorphic to the unit disk;
- degree one dangling edge: in this case, the unique incident face, which is homeomorphic to the unit disk, is still homeomorphic to the unit disk after the removal of the dangling edge.

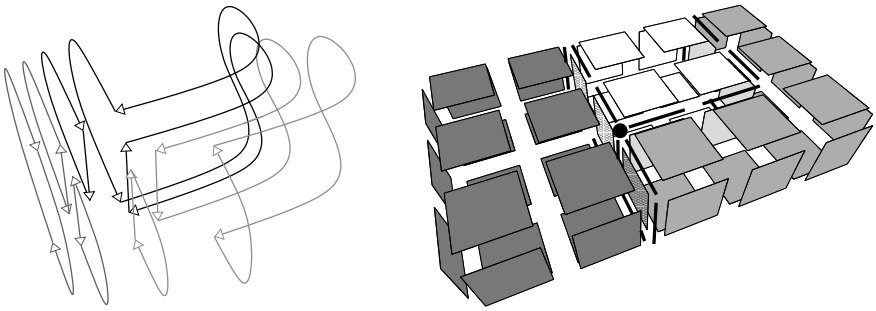
Thus, each face in level 2 maps is homeomorphic to the unit disk. And this is also the case in level 3s map since vertex removal and fictive edge shifting cannot involve a disconnection and thus cannot modify the topology of faces (as explained in the previous section). □

Proposition 26 *The topological map corresponding to a labeled image* I *does not depend on the shape of regions of* I.

PROOF. The proof is direct by definition of the topological map. Indeed, given an image I, the topological map is defined only by removing particular cells in I, and these cells are only characterized by their degree or their real degree, or by taking into account disconnections. None of these properties depend on the shape of regions of the subdivision which proves the proposition. □

5 Geometrical Part of the Topological Map

Topological map only represents the topological part of a labeled image. But in most applications, it is also necessary to represent geometry (note that it is not always the case. For instance, we need topology only to implement a segmentation algorithm by region merging, if the unique problem is to retrieve the neighbors of a given region). There are several ways to associate a geometrical model to a combinatorial map. In [2] we have given a solution based on a hierarchical representation where each face in the topological map



A B

Fig. 20. (A) Level 3 map. (B) The corresponding geometry in the intervoxel matrix, composed of 2 pointels, 18 linels and 53 surfels.

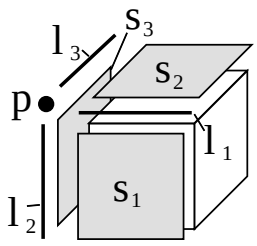
is represented by a 2D combinatorial map, and where each edge is represented by a 1D combinatorial map. The main advantage of this solution is to require only small memory space. Indeed, each face is represented by just a few basic elements. However, its main problem is the difficulty to retrieve, given a voxel, which region it belongs to.

We present here another solution which uses a matrix representing all the intervoxel elements of the corresponding image. We show the way to link the topological map and the elements of the matrix, and the modifications that have to be achieved during different removal operations in order to keep both models coherent. This solution is close to the one proposed in [9,20] where they also used a similar matrix to represent the geometry. The major difference between their work and our solution concerns the description of the topology. Indeed, in [9,20], they use an implicit representation of the combinatorial map based only on the intervoxel matrix (see [6] for more details on the comparison).

5.1 Links Between Topological Maps and Intervoxel Matrixes

We want to encode all the intervoxel elements that belong to intervoxel boundaries of the corresponding image. We can see in Fig. 20 the corresponding geometrical model associated with the topological map used during this paper.

The best way to implement all the intervoxel elements of an image is to use a matrix of bytes where each value encodes the intervoxel elements shown in Fig. 21. With this encoding, we need a matrix of (n_1+1, n_2+1, n_3+1) bytes to represent all the intervoxel elements for an image of size (n_1, n_2, n_3). Given an element at the position (i, j, k) of the matrix, each bit value is equal to zero if the corresponding intervoxel element does not belong to an intervoxel boundary, and otherwise it is equal to one.



Bit	0	1	2	3	4	5	6	7
Cell	p	l_1	l_2	l_3	s_1	s_2	s_3	–

Fig. 21. Implementation used to represent each byte of the intervoxel matrix.

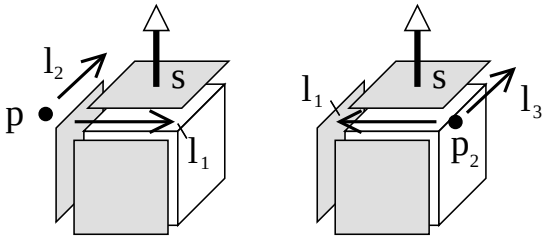
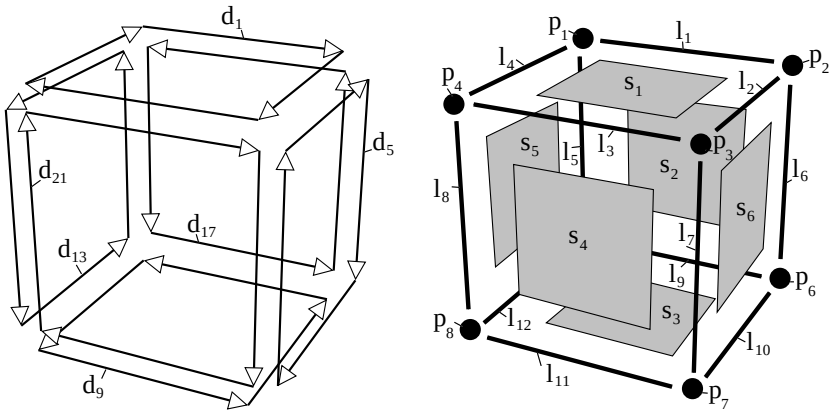


Fig. 22. Oriented surfel convention. (A) The triple (p, l_1, s) and the corresponding orientation of the surfel s obtained by the cross product $\vec{l_1} \times \vec{l_2}$. The triple (p, l_2, s) corresponds to the same surfel but with reverse orientation. (B) Different triples could represent the same oriented surfel. Here (p, l_1, s) and (p_2, l_3, s) represent the same oriented surfel.

Each dart of the topological map is linked with a triple (p, l, s) that allows to retrieve, given a dart d, the corresponding elements in the intervoxel matrix. With p we can retrieve the pointel associated with d, and so the coordinates of the corresponding vertex. With (p, l) we can retrieve the first linel associated with d. This linel gives the initial direction of the edge associated with d. Finally, with (p, l, s) we can retrieve the surfel associated with d that belongs to the intervoxel boundary associated with the face incident to d. This is an oriented surfel. The orientation is given by the cross product between $\vec{l}$ and the second linel incident to p and s (see example in Fig. 22). This orientation allows to distinguish the interior and the exterior of a volume incident to a given dart. Indeed, by convention, we chose that the normal of the oriented surfel associated to a given dart always corresponds to the normal of the surface of the associated region directed in the exterior of the region.

The links between darts and triples are made during the construction of the level 0 map which represents all the voxels of the image. Indeed, when we create the cube corresponding to a given voxel, we know exactly the correspondence between each dart and each element of the intervoxel matrix (partially given in Fig. 23). Thus, it is easy to link each dart of level 0 with its corresponding geometrical element in the intervoxel matrix as shown in Fig. 23.



d_1	(p_1, l_1, s_1)	d_5	(p_2, l_6, s_6)	d_9	(p_8, l_{11}, s_3)
d_{13}	(p_8, l_{12}, s_5)	d_{17}	(p_5, l_9, s_2)	d_{21}	(p_8, l_8, s_4)

Fig. 23. Links between darts and triples (given only for some darts of the cube). Note that the pointel p_5 is the only one hidden in this figure. Links for other darts are made in a similar way.

5.2 Intervoxel Matrix Modifications During Removal Operations

Each map level is computed starting from the previous level by applying some particular removal operations. Since each dart of the level 0 map is linked with its corresponding triple in the intervoxel matrix, this is enough to study what modifications have to be achieved in the intervoxel matrix during each particular removal operation in order to obtain the geometrical model of any map level.

A face is removed during the construction of level 1. Since the starting map is level 0, we know that each face of this map corresponds exactly to one surfel in the intervoxel matrix. Thus, each face contains exactly 8 darts, all associated to the same surfel (4 darts for a face of a given voxel, but there are two voxels around each face).

When a face is removed, the corresponding surfel in the matrix does not belong to an intervoxel boundary anymore. So we need to erase it from the matrix. This is simply achieved by setting the corresponding value to 0 in the matrix. Moreover, the 8 darts of the map that belong to the removed face are removed from the map and destroyed, and thus there is no other modification to apply for other darts.

An edge is removed during the construction of level 2. Since the starting map is level 1, we know that each edge of this map corresponds exactly to one linel in the intervoxel matrix. When an edge is removed, we just need to erase the corresponding linel from the matrix (by setting the corresponding value to 0). As for face removal, there is no other modification to apply for other darts.

A vertex is removed during the construction of level 3. As for other removal operations, the single modification is to erase the corresponding pointel from the intervoxel matrix.

5.3 What About Fictive Edges ?

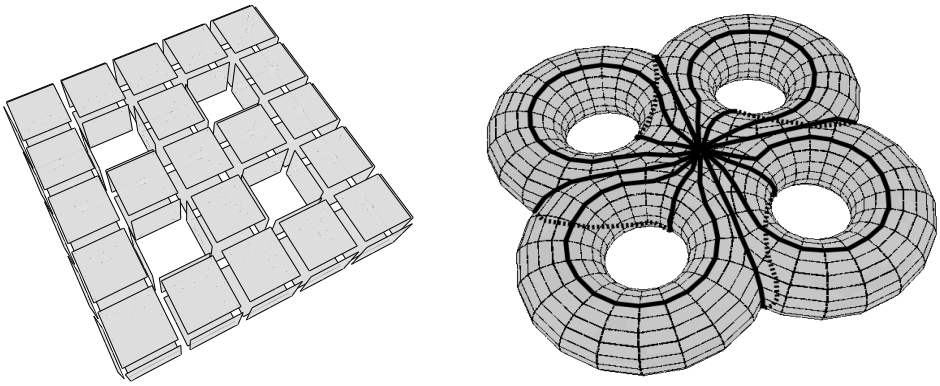
A fictive edge is an edge with degree one, i.e. incident twice to the same face. For that reason, there is no linel associated with such an edge in the intervoxel matrix. Indeed, we only keep in the matrix elements that belong to a boundary of faces (i.e. linels and pointels with degrees strictly greater than 2).

To do so, when we find a fictive edge during the construction of the level 2 map (when removing the edge induces a face disconnection or completely removing the face, see Def. 14), we remove the corresponding linel in the intervoxel matrix (like the modification made for edge removal). For the same reason, when during the construction of level 3, we find a vertex such as if we remove it, this induces removing completely a face (i.e. a real degree 0 vertex incident only to fictive edges that are loops, all the other cases cannot induce removing completely a face), the corresponding pointel does not belong to a boundary of the face and needs to be removed from the intervoxel matrix.

With these two modifications, if a region in the image has no adjacent region, the boundary of this region is composed of exactly one face, one vertex, and some edges (the number of edges depends on the Euler characteristic of the corresponding surface). The corresponding geometry of such a region in the intervoxel matrix is composed of surfels only. Indeed, there is no linel since all the edges are fictive, and there is no pointel since the vertex is a real degree 0 vertex.

Note that sometimes it could be interesting to keep a triple in the matrix in order to have a particular entry point which allows to retrieve a dart that represents the corresponding volume in the topological map, starting from the intervoxel matrix (for example to define a localization algorithm that retrieves the region that contains a given voxel). This could be possible without particular difficulty, by adding "fictive triples" in the intervoxel matrix. Note that with this solution, we do not link a geometrical element with a fictive edge but only add a fictive triple on each face that allows to retrieve a corresponding dart in the map.

It is also necessary to note that this choice of not linking fictive edges with geometrical elements in the intervoxel matrix is necessary in order to be able to obtain the minimal map. Indeed, another solution is to define a geometric analogous for each cell of the map (like the solution proposed in [9,20]). But with that choice, it is not possible to obtain the minimal map since we depend



A B

Fig. 24. (A) A 3D region whose surface is a torus with 4 holes. (B) The minimal corresponding subdivision (1 face, 8 edges and 1 vertex) which is impossible to represent in the intervoxel matrix.

on the shape of described regions, as we can see in the example in Fig. 24. In that example, the surface of the region is a torus with 4 holes, without any adjacent volume (except the region which contains the torus). The minimal corresponding subdivision (shown in Fig. 24B) has thus one face, eight edges (2 loops for each hole), and one vertex. But such a subdivision cannot be represented in the intervoxel matrix. Indeed, to represent such a subdivision, we would need 16 different linels around the pointel associated with the vertex (2 different linels for each edge, one for each edge extremity, we cannot use the same linel more than once since this induces the creation of another pointel) which is impossible in a 3D intervoxel space.

For that reason, if we want to have a geometric analogous for each topological cell, we obtain a topological model which is not minimal, and which depends on the geometry of the subdivision. Moreover, with this solution, we sometimes need to modify the topological model due to a geometrical modification on a region (like the example in Fig. 25). These problems are too important from our point of view, and that is why we choose not to link geometrical elements to fictive edges.

6 Incremental and Generic Extraction Algorithm of Any Map Level

The successive definitions of each map level seen in Section 4 immediately give a first extraction algorithm. Firstly, this algorithm builds the level 0 map of the labeled image, then we progressively simplify the map by following the level definitions. The main advantages of this algorithm are its simplicity, since it follows the definitions of the different map levels, and its complexity which is linear in number of voxels of the image. Its main problem is to build

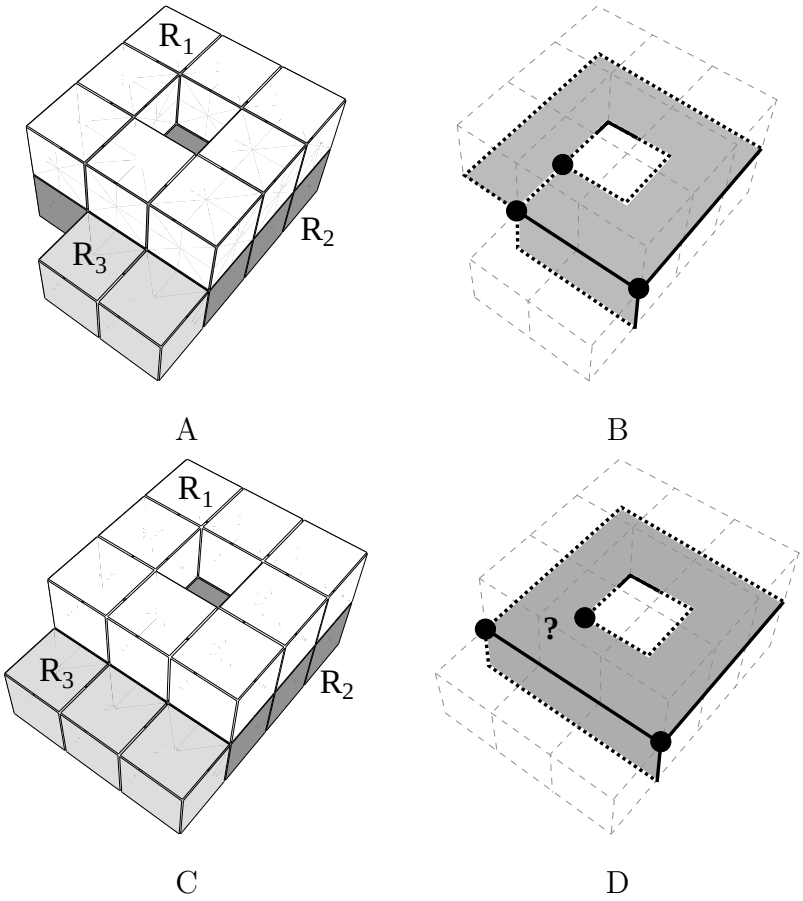


Fig. 25. Example where a geometrical modification induces a topological modification when we link fictive edges with geometrical elements. (A) Three 3D regions, R_1, R_2 and R_3, completely included in another one (not represented) without other adjacency. (B) The corresponding minimal subdivision. (C) The same regions where we have only modified the geometry of R_3. (D) With this very basic modification, it is now impossible to represent the fictive edge in the intervoxel matrix without modifying the subdivision (for example by adding a vertex).

first level 0 map, which involves the creation of many darts that are eventually destroyed during the following simplifications (the number of destroyed darts depends on the labelled image). That involves an important and useless time consumption due to memory allocation and de-allocation. Moreover, the memory cost necessary to represent a level 0 can be too important and can prevent the creation of the topological map, even if this map requires much less memory than for level 0. It is to solve these problems that we propose the incremental algorithm.

The main principle of the incremental algorithm given in Algorithm 7 consists in building the topological map incrementally by processing each voxel during a single scan of the image. Image I, which is the input of this algorithm, is a labeled image, with voxels having x-coordinates between $1 \dots n_1$, y-coordinates between $1 \dots n_2$ and z-coordinates between $1 \dots n_3$ (other voxels belong to the infinite region, see Section 2.2).

Before scanning the voxels of the image, we first create the upper border of the map (line 1 of the algorithm). Indeed, the following invariant has to be satisfied for each voxel of the image: the combinatorial map corresponding to the voxels already scanned has already been built. With this invariant, we are certain at any time that in the map, a dart belonging to the left face of the current voxel exists (resp. upper face, back face). It is to satisfy this invariant for the first voxel of the image (voxel with coordinates $(1, 1, 1)$), that we first build a combinatorial map that represents the upper border of the image.

This initial border is important, since its correct definition allows us to solve problems due to border voxels in the image, and also allows us to avoid particular processes for these voxels. For that, we identify voxels of coordinates (n_1+1, y, z) with voxels of coordinates $(0, y+1, z)$ (the last voxel of a line and the first voxel of the next line) and identify voxels of coordinates (x, n_2+1, z) with voxels of coordinates $(x, 0, z+1)$ (the last voxel of a given column and the first voxel of the same column in the next plane). In order to take these identifications into account, we use a modulo to compute the coordinates of a voxel. When the current voxel is (i, j, k), we consider it has coordinates (i modulo (n_1+1), $(j+(i \text{ div } (n_1+1)))$ modulo (n_2+1), $k+(j+(i \text{ div } (n_1+1)))$ div (n_2+1)) which gives the correct voxel in regard to the different identifications. With these identifications, when we create the last cube of a given line, this involves the creation of the left face of next line's first voxel, and when we create the last cube of a given column, this involves the creation of the back face of same column's first voxel in the next plane.

We can see in Fig. 26 this initial border for an image of size $3 \times 2 \times n_3$. This initial border is composed of the upper faces of the image that correspond to the boundary between voxels with 1 in z-coordinates and voxels of the infinite region. These faces go from 1 to n_1+1 in x-coordinates, and from 1 to n_2+1 in y-coordinates (indeed, the image scan is increased by one voxel in any of the 3 axis of the space in order to take into account the boundary surfaces with the infinite region). The initial border also contains the faces behind the first line of the image, and the face on the left of the first voxel of this first line. With this initial border, the first voxel can be sewn with the three faces around it (on the left, on top and at the back). The other voxels of the first line have a

Algorithm 7: Incremental extraction of a topological map

Input: I A labeled image of $n_1 \times n_2 \times n_3$ voxels.
Output: The topological map corresponding to I.

```
1  Step 1: last ← Build the upper border of the image;
   Step 2:
   for k = 1 to n_3 + 1 do
       for j = 1 to n_2 + 1 do
           for i = 1 to n_1 + 1 do
2              tmp ← new cube corresponding to voxel (i, j, k);
               up ← ComputeUpFromLast(last);
               behind ← ComputeBehindFromLast(last);
3              3-sew this cube along the three faces last, up and behind;
               Let v_1, v_2 and v_3 the volumes incident to last, up and behind;
               Let f_1, f_2 and f_3 the faces incident to last, up and behind;
               Let e_1, e_2 and e_3 the edges incident to last, β_0(up) and β_0(behind);
               Let v the vertex incident to last;
4              if v_1 (resp. v_2, v_3) belongs to the same region as the current
               voxel then
                   Remove f_1 (resp. f_2, f_3);
5              if the local degree of e_1 (resp. e_2, e_3) is two then
                   if the removal of e_1 (resp. e_2, e_3) does not involve a face
                   disconnection and does not remove completely the face
                   then
                       Remove e_1 (resp. e_2, e_3);
                   else
                       Mark e_1 (resp. e_2, e_3) as fictive edge;
6              if the real degree of v is two and the two real edges are
               non-loop then
                   Shift all fictive edges incident to v;
                   Remove v;
               last ← tmp;
7  Step 3: Remove each real degree zero vertex incident to at least one
   non-loop edge e, after having shifted all fictive edges incident to this
   vertex, except e;
8  Step 4: Compute the region inclusion tree;
```

face at the back and on top, and each new voxel involves the creation of the next voxel's left face (see Prop. 27 for proof).

We can see in Fig. 26 how the extreme faces are sewn in order to identify the voxels as explained beforehand. It is important to notice that this initial border does not satisfy the combinatorial map definition since it is not closed. Indeed, each dart is 3-free (a dart d is 3-free if there is no dart d' in the map

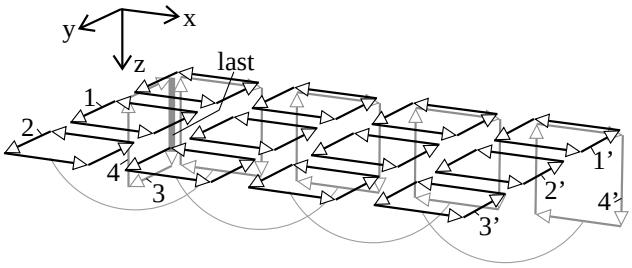


Fig. 26. The upper border created before starting the image scan, here for an image of size $3 \times 2 \times n_3$ (for an image of size $n_1 \times n_2 \times n_3$, the upper border size is $(n_1 + 1, n_2 + 1)$ due to the voxels of the infinite region). β_2 relations are given for some darts, either by drawing grey curves or by naming darts with same number i and i'.

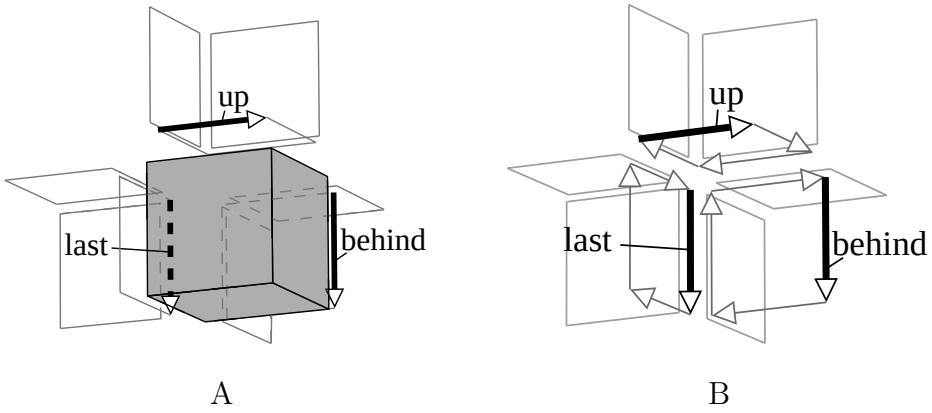


Fig. 27. Darts *last*, *up* and *behind* in respect to the current voxel. This is a partial drawing, where only a few dart orientations are given in order not to make the figure unreadable. (A) Darts are drawn with the current voxel. (B) Only the darts are represented.

such that $\beta_3(d') = d$). But this is not a problem because we iteratively close each dart by successively processing each voxel. Moreover, we only consider the map locally around closed cells (as explained in the following) and we never pass through a 3-free dart during an orbit scan. Moreover, we can notice that the initial border is closed for other dimensions (0, 1 and 2).

After the upper border is created, the *last* dart is initialized in order to point out a precise dart on the face on the left of the first image voxel. We also need a particular dart on the faces behind and on top of the current voxel. These particular darts are called respectively *last*, *up* and *behind*. We can see in Fig. 27 the positions of these three particular darts. Note that the choice of a dart for each face is purely arbitrary and that we can choose other darts. The important point is to be coherent between the dart positions in each algorithm.

During the image scan, we only keep the *last* dart as the other two darts can be retrieved given *last* by using Algorithm 8 and Algorithm 9. These two

Algorithm 8: Computation of *up* dart

Input: A map M;
last the dart on the left of the current voxel.
Output: *up* the dart on top of the current voxel.

$up \leftarrow \beta_{02}(last)$;
while up *is not 3-free* **do**
  $up \leftarrow \beta_{32}(up)$)
$up \leftarrow \beta_1(up)$;
return up;

Algorithm 9: Computation of *behind* dart

Input: A map M;
last the dart on the left of the current voxel.
Output: *behind* the dart behind the current voxel.

$behind \leftarrow \beta_2(last)$;
while behind *is not 3-free* **do**
  $behind \leftarrow \beta_{32}(up)$)
$behind \leftarrow \beta_{11}(behind)$;
return behind;

algorithms are based on the same principle. In Algorithm 8, the *up* dart is first initialized with $\beta_{02}(last)$. This dart belongs to the edge incident to the two faces which contain *last* and *up*. Then, that is enough to turn around this edge, by using β_{32}, until obtaining a dart which is 3-free. Indeed, faces between the one which contains *last* and the one which contains *up* all have their darts which are not 3-free. So if the current dart is 3-free, that means that it belongs to the face that contains *up*. That is then enough to apply β_1 to find dart *up*. Algorithm 9 uses exactly the same principle but for a different initial dart, and with a different movement at the end.

The complexity of Algorithm 8 is linear in the number of faces incident to the edge between the face which contains *last* and the one which contains *up* (or *last* and *behind* for Algorithm 9). This number of faces is always equal to 0, 1 or 2, since each face corresponds to a surfel (see Fig. 28 to see all the possible configurations). So the complexity of these two algorithms is in constant time in our particular case of discrete subdivision.

The main part of the incremental algorithm is the image scan. For each voxel, a corresponding cube is created in the map (line 1 of Algorithm 7) and 3-sewn with its neighbors using the three particular darts *last*, *up* and *behind* (line 3). Then, we test the different new cells in order to know which ones need to be simplified. But here, these tests are only done on the *closed* cells incident to the new cube.

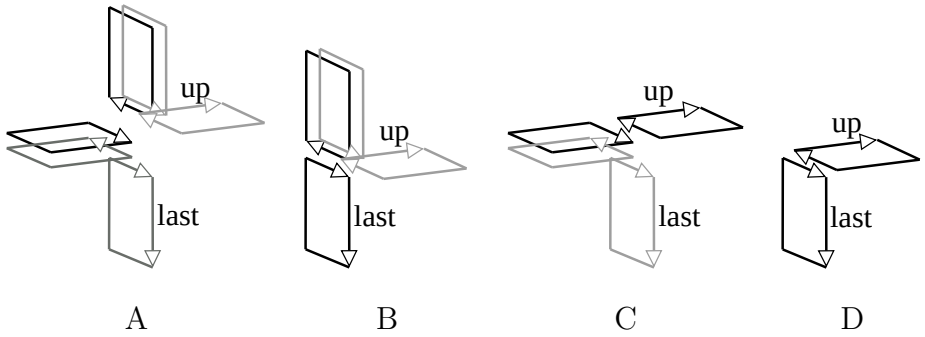


Fig. 28. The four possible configurations of faces between the one which contains *last* and the one which contains *up*.

The term "closed" cell is used here to speak about a cell in the combinatorial map where no dart is *3*-free. Indeed, we know that the topological map is closed for all dimension, but during its construction, the lower border of the map is 3-free, and the corresponding cells cannot be simplified since we do not know them completely.

At the end of the incremental algorithm, after the image scan of all the voxels of the image, we retrieve the initial border below the topological map and disconnected from it. This border is now useless and can thus be destroyed. The last work consists in inclusion tree computation in order to be able to place relatively the different connected components of the map. This is achieved with Algorithm 10 given in Section 6.2 and the obtained structure is thus the topological map corresponding to the initial image.

Note that in order to simplify the presentation of the algorithm, we have chosen to make the removal of real degree zero vertices in a separate step (line 7 of Algorithm 7), after the image scan. This separation allows us to simplify the algorithm and explanations. Moreover, we will see in Section 7 that the additional time due to this step is not really important in comparison to the time taken by the image scan. Finally, it is of course possible to optimize this algorithm by applying the removal of real degree zero vertices in the same step as the image scan. But this implies some additional tests in order to guarantee not to miss a vertex.

Note also that this incremental algorithm is generic. Indeed, it is enough to add a parameter to extract a precise level, and make removal operations only if they correspond to the level wanted. We do not present this possibility here in order to make Algorithm 7 shorter but this can be achieved immediately.

To prove that Algorithm 7 computes the topological map, we begin to prove that the following invariant is true during the extraction:

Proposition 27 *During Algorithm 7, before processing voxel* (i, j, k)*:*

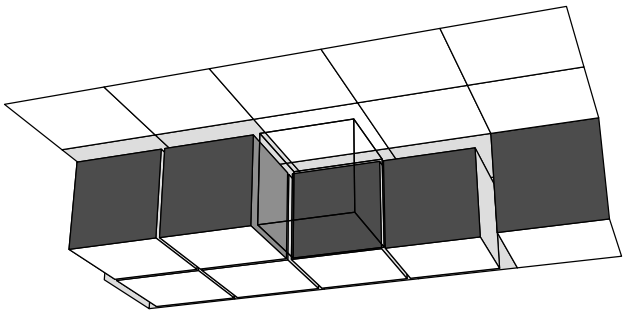


Fig. 29. Proposition 27 ensures that different faces exist around the next voxel to process v (drawn in wire frame). A face to the left of v (in light grey), faces behind the last line of voxels (in dark grey), and faces above the voxels of the last plane (in white).

(1) there is a face to the left of voxel (i, j, k)*;*

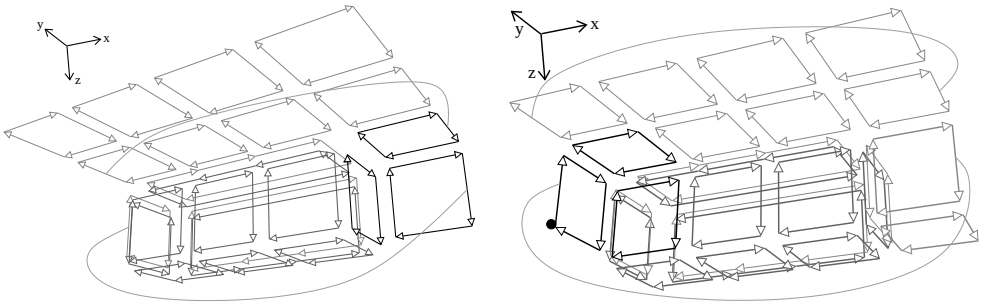
(2)
- $\forall x, 1 \leq x < i$*, there is a face behind voxel* $(x, j+1, k)$*;*
- $\forall x, i \leq x \leq n_1 + 1$*, there is a face behind voxel* (x, j, k)*;*

(3)
- $\forall x, 1 \leq x < i$*, there is a face above voxel* $(x, j, k+1)$*;*
- $\forall x, 1 \leq x \leq n_1+1, \forall y, 1 \leq y < j$*, there is a face above voxel* $(x, y, k+1)$*;*
- $\forall x, i \leq x \leq n_1 + 1$*, there is a face above voxel* (x, j, k)*;*
- $\forall x, 1 \leq x \leq n_1 + 1$, $\forall y, j < y \leq n_2 + 1$*, there is a face above voxel* (x, y, k)*;*

This invariant ensures that whatever the current voxel (i, j, k), the border of the map exists and allow to attach the future cubes. Items 2 and 3 of this proposition are cut into different parts, because we need to consider differently the previous voxels of the same line and the previous voxels of the same plane and future voxels (see Fig. 29). Indeed, for example in item 2, the previous voxels of the same line ($\forall x, 1 \leq x < i$, voxels (x, j, k)) have been already built and not future voxels of the same line ($\forall x, i \leq x \leq n_1$, voxels (x, j, k)). For the first type of voxels, we have to consider voxels on the next line (voxels $(x, j+1, k)$) while for the second type we consider voxels of the same line (voxels (x, j, k)). This is the same for item 3 but with more cases since previous voxels can be on the same line (first part of item 3) or on previous lines (second part of item 3), and the same for future voxels.

PROOF. Proposition 27 is satisfied for the first voxel of the image (voxel with coordinates $(1, 1, 1)$) due to the creation of the initial border (see Fig. 26). Indeed, there is a face to the left of voxel $(1, 1, 1)$. $\forall x, 1 \leq x \leq n_1 + 1$, there is a face behind voxel $(x, 1, 1)$. Lastly, $\forall x, 1 \leq x \leq n_1+1$ and $\forall y, 1 \leq y \leq n_2+1$, there is a face above voxel $(x, y, 1)$.

Let us suppose that the invariant is true for the voxel $v = (i, j, k)$ (with $1 \leq i \leq n_1 + 1$, $1 \leq j \leq n_2 + 1$ and $1 \leq k \leq n_3 + 1$) and prove that it is already true for the next voxel v' processed in the algorithm. There are four



A B

Fig. 30. Configuration of the map during its construction. (A) Before processing the last voxel of the first line. (B) And after. We can observe that the creation of the last cube of a line involves the creation of the left face of the first voxel of the following line.

cases to consider depending on the position of the current voxel in regard to the three borders of the image:

(1) $i \leq n_1$; then $v' = (i+1, j, k)$.
(2) $i = n_1 + 1$ and $j \leq n_2$; then $v' = (1, j+1, k)$.
(3) $i \leq n_1$ and $j = n_2 + 1$; then $v' = (i+1, j, k)$.
(4) $i = n_1 + 1$ and $j = n_2 + 1$; then $v' = (1, 1, k+1)$.

(1) In the first case, $v = (i, j, k)$ and $v' = (i+1, j, k)$. When we create the cube corresponding to voxel (i, j, k), this creates a face to the left of v', and this proves item 1 of Prop. 27 for the next voxel v'. This also creates a face in front of v which is behind voxel $(i, j+1, k)$. This proves the first part of item 2 of Prop. 27 for v'. The second part of item 2 is already true since the voxels concerned are not modified. Lastly, the cube creation creates a face under v which is the face above voxel $(i, j, k+1)$. This proves the first part of item 3 of Prop. 27 for v'. Other parts are already true since other voxels are not modified by the creation of the cube corresponding to v.
(2) In the second case, $v = (n_1 + 1, j, k)$ and $v' = (1, j+1, k)$. The creation of the cube creates the left face of the first voxel of the following line (see example in Fig. 30. This proves item 1 of Prop. 27 for voxel v'.

    The new cube involves the creation of a face in front of v which is behind voxel $(n_1 + 1, j + 1, k)$. Since the invariant is true for voxel v, and with this additional face, we can conclude that $\forall x, 1 \leq x \leq n_1 + 1$, there is a face behind voxel $(x, j+1, k)$. This proves item 2 of Prop. 27 for voxel $v' = (1, j+1, k)$. The same type of proof can be made to prove the third item of Prop. 27.
(3) In the third case, $v = (i, n_2+1, k)$ and $v' = (i+1, n_2+1, k)$. The creation of the cube creates the face behind the first voxel of the same column of the following plane (see Fig. 31B). The invariant for v' can be proven by

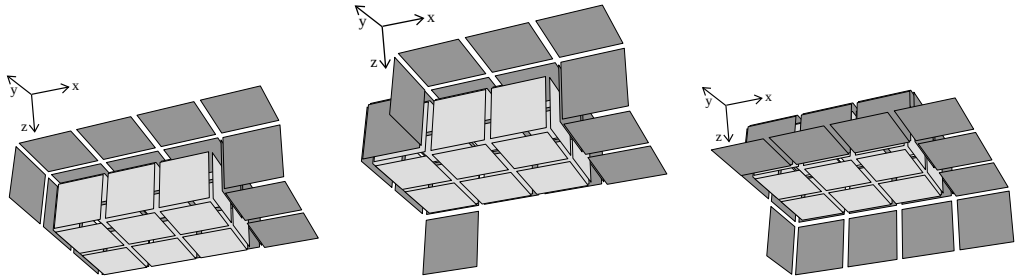


Fig. 31. Configuration of the map during its construction. The arrows on darts are not drawn to make figures more understandable. The faces corresponding to the infinite regions are drawn in dark grey. (A) Before processing the last voxel of the first column. (B) And after. We can observe that the creation of the last cube of a column involves the creation of the face behind the first voxel of the same column of the following plane. (C) Configuration obtained after having processed all the voxels of the first plane.

using exactly the same arguments as for the first case.

(4) In the fourth case, $v = (n_1 + 1, n_2 + 1, k)$ and $v' = (1, 1, k + 1)$. After creating of the cube corresponding to v, we obtain a configuration similar to the one shown in Fig. 31C. Indeed, the creation of the cube involves the creation of the two faces to the left of the first voxel of the next plane, and behind the last voxel of the first line of the next plane. We can observe in Fig. 31C that the configuration we obtain is the same for voxels of the next plane as the configuration of the initial border for the voxels of the first plane. For this reason, we can continue the image scan for the next plane and we can prove that Prop. 27 is true for the v' which is the first voxel of the next plane. □

Thanks to Prop. 27, we are sure that during Algorithm 7, before processing voxel $v = (i, j, k)$, there is a face to the left, above and behind v. This point is crucial to prove that Algorithm 7 computes the topological map.

Proposition 28 *Given a 3D labeled image* I*, Algorithm 7 computes the topological map associated to* I*.*

PROOF. To prove Prop. 28, we prove that each cell of the level 0 map associated with I is created during the image scan, and that each cell is removed if necessary depending on the topological map definition. This proves that the map obtained is the topological map by definition of the different levels.

Algorithm 7 runs through each voxel $v = (i, j, k)$, with $1 \leq i \leq n_1 + 1$, $1 \leq j \leq n_2 + 1$ and $1 \leq k \leq n_3 + 1$. Thanks to Prop. 27, we know that before processing voxel v, there is a face to the left, above and behind v. This ensures

that when we create the cube corresponding to voxel v, it can be 3-sewn to the three faces containing *last*, *up* and *behind*.

This closes some cells that were previously open and these cells can now be tested and eventually simplified. The cells in question are the three faces incident to *last*, *up* and *behind*, the three edges incident to *last*, $\beta_0(up)$ and $\beta_0(behind)$, and the unique vertex incident to *last*.

Since this is done for each voxel of the image, this proves that each cell of the level 0 map is created by Algorithm 7. However, there are some additional cells which do not belong to the level 0 map and which are created by the incremental algorithms: these cells are the faces between voxels that belong to the infinite region (i.e. voxels (n_1+1, j, k), voxels (i, n_2+1, k) and voxels (i, j, n_3+1). But these cells are necessarily removed by Algorithm 7 since they are between two voxels that belong to the same region (i.e. the infinite region).

Thus, we are sure that Algorithm 7 creates exactly the same cells as the ones of the level 0 map. Then, by processing, for each voxel, the three faces incident to *last*, *up* and *behind*, the three edges incident to *last*, $\beta_0(up)$ and $\beta_0(behind)$, and the unique vertex incident to *last* we have considered all the cells corresponding to the level 0 map. Since we use exactly the same tests during Algorithm 7 as for the definition of the different map levels in order to remove (or not) each cell, and since each cell is checked once, this proves that the final map obtained is the level 3 map corresponding to the initial image.

The last part of the proof consists in showing that the inclusion tree computation is correct. This is achieved by proving that Algorithm 10 computes the inclusion tree, and the proof is given in Section 6.2.

This proves that the structure obtained is therefore the topological map corresponding to the initial image. □

Proposition 29 *The average complexity of Algorithm 7 is linear in the number of voxels of the input image.*

PROOF. The average complexity of the incremental extraction algorithm is linear in number of voxels n of the input image (of size $n_1 \times n_2 \times n_3$). First, the upper border creation has $O(n_1 \times n_2)$ complexity. Second, we scan each voxel exactly once during the image scan. For each voxel, the cube creation is in constant time, and each different removal operation is also in constant time (since each face has exactly 8 darts, each edge at the most 8 darts and each vertex at the most 24 darts). Moreover, testing if an edge removal involves a disconnection can be bounded by a constant, thanks to the union-find trees. Finally, fictive edge shifting and real vertex degree computation are linear in number of edges incident to the considered vertex. This number can

be bounded by 6 in the average case (see [28]). This is the reason why the complexity of Algorithm 7 is not linear in the worst case but only in average. Note that the worst case does not occur in practical cases as we can see in our experiments (cf. Section 7). With all these complexities, we obtain that the final average complexity is linear in number of voxels of the image. □

6.2 Inclusion Tree

The computation of the inclusion tree is the last step of our extraction algorithm, after the extraction of a level l map. This tree is necessary to keep a relation between all the different connected components of the map. Indeed, a region R in the input map can be made up of several disconnected surfaces if there are some regions completely included in R. The inclusion relation used here is the one presented in Section 2.2, but we only represent direct inclusions (relations we cannot 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.

To compute the inclusion tree, we run through all the darts of the map connected component by connected component. Indeed, a connected component is made up of regions which all have the same father. In order to retrieve the different inclusions we need some specific 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 on the upper left face of this region. This property ensures us we can directly 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 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 scan from top to bottom, from back to front and from left to right. This order allows to guarantee, when the current region is R, that its father was already created and placed in the inclusion tree.

These properties are given by our extraction algorithm. Indeed, the list of regions 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. Assigning the region of each dart is carried out at the same time as the cube is created (with a look-up table that gives for each label the corresponding region, note that this is possible

only because each region is 6-connected, and this is the unique reason why this work is limited to labeled images). When we meet a new region, we label the new darts within this new region. Otherwise, we copy the label of the same region's existing darts without using the look-up table.

Algorithm 10: Inclusion tree computation

Input: M the map of a labeled image I;
L The sorted list of regions of the image I.
Output: An inclusion tree T.

Unmark each region of L;
Create 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 $region(\beta_3(d_{init}))$;
2 **foreach** *dart* d *of the connected component incident to* d_{init} **do**
    if region(d) *not marked* **then**
      Create a node corresponding to *region(d)*;
      Set this node as a son of *father* in T;
      Mark *region(d)*;
return T

The computation of the inclusion tree is presented in Algorithm 10. This algorithm's input is a map of any level and a list of regions, sorted as explained above, and its output is the corresponding inclusion tree. This algorithm is made of two loops. The first loop runs through regions of list L and the second loop processes each region that belongs to a same connected component, and thus which are all included in the same region. This algorithm uses two classical basic operations which allow to handle trees: a first operation which creates a new node associated with a given region, and a second one which sets a node n as son of a given node *father*. This operation only updates the tree locally so that n becomes the son of *father*.

Proposition 30 *Given a map* M *corresponding to a labeled image* I*, and* L *the sorted list of regions contained in* I *(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 scan from top to bottom, from back to front and from left to right.), Algorithm 10 computes the inclusion tree associated with* I.

PROOF. Proposition 30 can be proved by induction depending on the current region R of the first loop. Let us suppose that each region before R in the list L is correctly placed in the inclusion tree under construction. This is obviously true for the first region of the list since there is no region before the first region.

If R is marked, that is because it was already considered and placed in the inclusion tree so we can process the next region on the list.

If R is not marked, we know that R is included in $region(\beta_3(representative(R)))$. Indeed, since list L is sorted, 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, on its left and on back. So $\beta_3(d_{init})$ is obviously a dart of the region containing the current region (called *father*). Moreover, each region belonging to the same connected component as R is also included in the region *father*.

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

This proves that after it was considered, region R, plus all the regions that belong to the same connected component of R, are correctly added to the inclusion tree. Since we can prove this for each region on the list, this proves that at the end of our algorithm, we have computed the inclusion tree corresponding to image I. □

Proposition 31 *Algorithm 10 has $O(n)$ complexity, with n the number of darts of map* M.

PROOF. During Algorithm 10, we process each dart exactly once, because we run through the connected components one by one, and two different connected components are inevitably disjointed. We also process each region only once because we run through the list of regions and we mark each treated region. Moreover, each basic operation used in this algorithm is done in constant time. Based on these facts, we can deduce that Algorithm 10 has $O(n + n_R)$ complexity, with n_R the number of regions of the image. However, since the number of regions is lower than the number of darts, the complexity is thus linear in number of darts of the map. □

We can note that Algorithm 10 works for every map level and not only the last level. Moreover, its execution time decreases when the level increases, even if the global complexity does not change. Indeed, the number of darts to run through decreases more and more for higher levels.

7 Experiments and Analysis

We have implemented the incremental extraction algorithm of the topological map and made some experiments in order to study time and memory required by our approach. All our experiments were made on a classical personal computer with an Athlon 2000MHz CPU with 512Mb of memory and a Linux Debian System. Our computer software was developed in C++. We have chosen to represent each dart by a structure where each β_i relation is a pointer to the dart i-sewn. This implementation is efficient in time since we can access to each β_i of a given dart in direct access, and we can also directly modify each β_i relation. But of course this is to the detriment of memory space occupation. The intervoxel matrix is represented with a matrix, and we keep the original image in a second matrix. The goal is, in our future works, to use our program in segmentation algorithm, and for that it is necessary to keep the initial image during the different segmentation steps.

We have made our experiments on artificial images in order to analyze the memory space and the computation time independently of the image content. We have worked on two types of images:

(1) cubic images where each region is a cube of size s;
(2) random images where each region is randomly created.

For the first type of image, we have generated images of size between 32^3 and 255^3, and for each size we have generated each image starting with s (the size of each cubic region) equal to 4 and we multiply it by two until s is equal to the size of the image. We have chosen to start with region of size 4, since smaller regions are, most of the time, not interesting to consider in real images. Moreover, even if in a real image there could be small regions, it is reasonable to think that there are not many, and moreover that they are balanced by big regions.

For the second type of image (random images), we have also generated images of size between 32^3 and 255^3, and for each size we have generated 10 random images in order to compute an average of the results obtained. For each image, the number of generated regions is a random number between 1 and the size of the image. To obtain a 3D labeled image given a random image, we use a basic connected component labeling algorithm.

In the following, we present average values for the time of our incremental extraction algorithm, and for the memory space used to represent the topological map. Of course, each result depends on the image contents but our experiments allow to verify the linearity of our extraction algorithm and the linearity of the topological map in memory space. Table 1 and Table 2 show information concerning regions, respectively for cubical images and for ran-

Table 1
Information about regions for cubic images. Each value is the average on all the images of a same size. *Nb Regs* is the number of regions. *Size* is the region size. *Std Size* is the standard deviation of the region size.

Size	32^3	64^3	128^3	160^3	192^3	224^3	255^3
Nb Regs	146,2	936,2	6241,5	12193,3	21065,8	33454,5	49932
Size	224,1	280	336	335,9	335,9	335,9	332,1
Std Size	1431,9	4086,8	11580,5	9208,2	9199,3	9865,3	11446

Table 2
Information about regions for random images. The key is the same as for Table 1, plus *Depth Tree* is the depth of the inclusion tree, *Nb Tunnels* is the number of tunnels and *Std Tunnels* is the standard deviation of the number of tunnels.

Size	32^3	64^3	128^3	160^3	192^3	224^3	255^3
Nb Regs	77,9	730	7457,4	15393,7	27144	47243	68951,2
Depth Tree	2	2,7	3	3	3	3	3
Size	420,6	359,1	281,2	266,1	260,7	237,9	240,5
Std Size	3164	8211,9	20615	28066	36668,8	43746,8	53737,3
Nb Tunnels	1,01	0,71	0,34	0,27	0,23	0,2	0,18
Std Tunnels	5,9	9,8	11,5	12,3	12,7	13,8	14,4

dom images: the number of regions, the depth of the inclusion tree (not given for cubic images since it is always equal to one as there is no included region in such types of images), the size of regions, the number of tunnels (not given for cubic images since it is always equal to zero due to the type of generated images). Since these last both values are average values, we give also the associated standard deviation.

We can observe in Table 1 and Table 2 that the standard deviation increases when the size of image increases (for region sizes and for number of tunnels for random images). This is due to the fact that for bigger images, differences between regions are more important. For example for tunnels, we can have regions with very small number of tunnels and with very big number of tunnels (for image of size 255^3, the number of tunnels is between 0 and more than 4000).

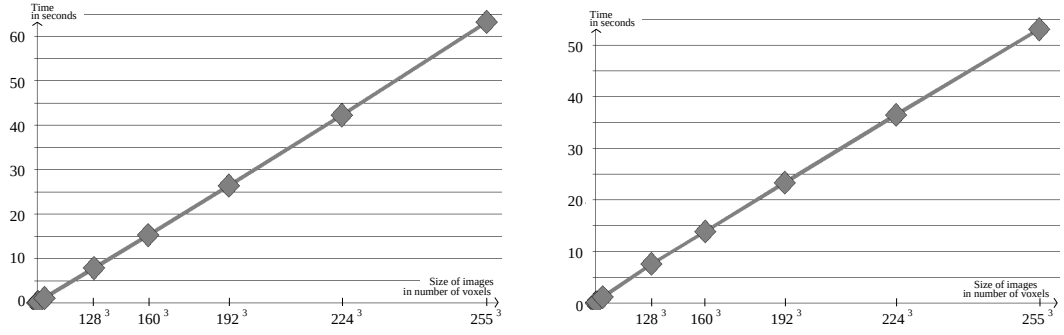
We can see in Table 3 the results of topological map extraction for cubic images. In this table, we give the time required to extract the topological map by using the incremental algorithm, and we have detailed each step of the algorithm (construction of the upper border, scanning of the image, removal of real degree zero vertices and inclusion tree computation). Similar results are given for random images in Table 4. Finally, we have summarized the total

Table 3
Time (in seconds) required to extract the topological map by using the incremental algorithm for cubic images. Times are detailed for the different steps of the algorithm. Step 1: Border creation. Step 2: Image scan. Step 3: Removal of real degree zero vertices. Step 4: Inclusion tree computation. The last line gives the total time used for the extraction, including all the different steps.

Size	32^3	64^3	128^3	160^3	192^3	224^3	255^3
Step 1	0	0	0,01	0,01	0,02	0,03	0,04
Step 2	0,15	1,09	7,67	14,93	25,79	41,29	61,89
Step 3	0	0	0,05	0,09	0,16	0,26	0,41
Step 4	0	0,01	0,11	0,23	0,42	0,67	0,87
Total	0,15	1,11	7,83	15,26	26,39	42,25	63,21

Table 4
Time (in seconds) required to extract the topological map by using the incremental algorithm for random images. The key is the same as for Table 3.

Size	32^3	64^3	128^3	160^3	192^3	224^3	255^3
Step 1	0	0	0,01	0,01	0,02	0,03	0,04
Step 2	0,16	1,09	7,1	12,99	21,87	34,22	49,83
Step 3	0	0,04	0,34	0,62	1,05	1,61	2,28
Step 4	0	0,01	0,13	0,24	0,4	0,64	0,91
Total	0,16	1,15	7,58	13,86	23,34	36,49	53,06



A B

Fig. 32. Total time (in seconds) required to extract the topological map by using the incremental algorithm for different image sizes. This time includes all the different steps of the extraction algorithm. (A) Time for cubic images. (B) Time for random images.

time necessary to extract the topological map in Fig. 32.

If we look at the two curves of Fig. 32, we can first verify that the extraction

algorithm is actually linear. This allows to verify the theoretical complexity which is only given in average in Section 6. We can also note that the time required is not so long, particularly if we remember that our software is only a prototype. By using some classical programming techniques (for example block memory allocator) and also by using some optimizations of our algorithm (for example by using precode notion [2]), we can obtain a faster program. Finally, if we look at the time for each step of the algorithm (given in Table 3 and Table 4), we can notice that the time required to remove the real degree zero vertices is not important in comparison with the time of the image scan and map construction. This shows that we do not waste much time by separating these two steps, and we have won in simplification of our algorithm.

If we compare the two curves, we can observe that the time necessary to compute the topological map is longer for cubic images than for random images. This is due to the fact that cubic images generated with regions of small sizes have many more regions than random images [13] . When the number of regions is smaller, there are more local configurations that are completely inside a region. These configurations are processed faster than configurations in a region boundary, since all the three faces incident to the new voxel are removed. After these removals, there are no edges and no vertex to simplify. For this reason, there are obviously no tests of disconnection and no fictive edge management.

Now we present in Table 5 the memory space occupation for the topological map for cubic images. In this table, we give the total memory used to represent the topological map, and we have detailed different parts of our model (combinatorial map, inclusion tree, image, intervoxel matrix and union-find trees). Similar results are given for random images in Table 6. Finally, we have summarized the total memory required to represent all the topological map, including all the different parts, in Fig. 33.

If we look at the two curves of Fig. 33, we can verify that the memory space used is also linear in number of voxels of the image. And if we compare the two curves, we can observe that the memory space required to represent the topological map is more important for random images than for cubic images. This is due to the fact that for cubic images, regions are regular, and so there are less darts to represent regular regions since each region (except those in the border of the image) has exactly 6 different neighbors and thus exactly 6 faces. This is not the case for random images where regions can have different types of neighborhoods and so more faces. Note that in memory space occupation, we have taken into account the space occupied by union-find trees, but this memory space is only used during the extraction algorithm, and can be freed after the extraction.

[13] For example, 255^3 cubic image with regions of size 4^3 is composed with 262144 regions and its extraction takes 123 seconds.

Table 5
Memory space used to represent the topological map for cubic images (units are given in the table since they are not always the same: kb for kilo-bytes, and mb for mega-bytes). Memory spaces are detailed for the different structures used in our model. *Map*: Combinatorial map. *Incl. tree*: Inclusion tree. *Image*: Original image. *Intervoxel*: Intervoxel Matrix. *UF-trees*: Union-find trees. The last line gives the total memory space including all the different parts.

Size	32^3	64^3	128^3	160^3	192^3	224^3	255^3
Map	185,1 kb	1228.8 kb	7,6 mb	14,9 mb	25,6 mb	40,5 mb	60,4 mb
Incl. tree	6,9 kb	43,9 kb	0,3 mb	0,6 mb	0,6 mb	0,7 mb	0,8 mb
Image	64,0 kb	512,0 kb	4,0 mb	7,8 mb	13,5 mb	21,4 mb	31,6 mb
Intervoxel	70,2 kb	536,4 kb	4,1 mb	7,9 mb	13,7 mb	21,7 mb	32,0 mb
UF-trees	114,0 kb	691,2 kb	4,3 mb	8,4 mb	14,2 mb	22,4 mb	32,7 mb
Total	440,3 kb	3012,3 kb	20,3 mb	39,6 mb	67,6 mb	106,8 mb	157,5 mb

Table 6
Memory space used to represent the topological map for random images. The key is the same as for Table 5.

Size	32^3	64^3	128^3	160^3	192^3	224^3	255^3
Map	312,5 kb	2150,4 kb	13,5 mb	23,8 mb	38,9 mb	59,3 mb	83,4 mb
Incl. tree	3,6 kb	32,7 kb	0,3 mb	0,7 mb	1,3 mb	2,1 mb	3,0 mb
Image	64,0 kb	512,0 kb	4,0 mb	7,8 mb	13,5 mb	21,4 mb	31,6 mb
Intervoxel	70,2 kb	536,4 kb	4,1 mb	7,9 mb	13,7 mb	21,7 mb	32,0 mb
UF-trees	116,9 kb	651,0 kb	3,6 mb	6,1 mb	9,8 mb	14,6 mb	20,4 mb
Total	567,2 kb	3882,5 kb	25,5 mb	46,3 mb	77,2 mb	119,2 mb	170,4 mb

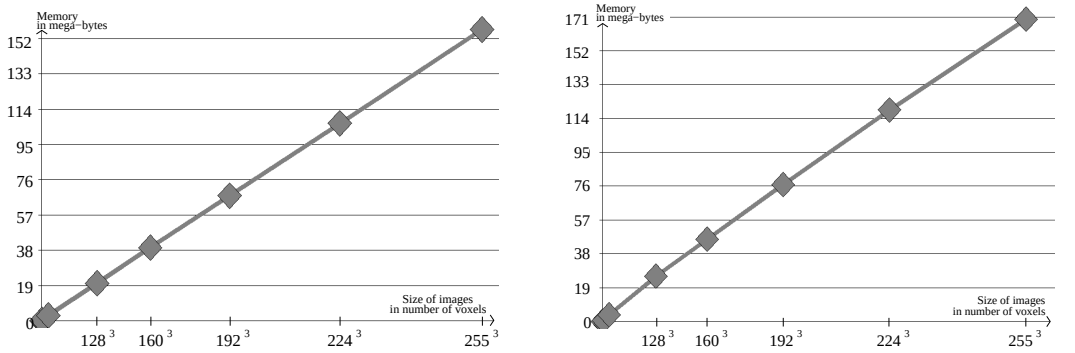


Fig. 33. Total memory (in mega-bytes) required to represent the topological map for different image sizes. This memory size includes all the different parts of the topological map. (A) Memory for cubic images. (B) Memory for random images.

8 Conclusion

In this paper we have presented a model that allows to describe 3D labeled images, the topological map, and we have given an incremental extraction algorithm which computes this model from a labeled image in linear time. The main interest of this model is to explicitly represent structural information on image regions. This information can be used in image processing when we need to compute efficiently some particular criteria (for example retrieve all the regions adjacent to a given region during a segmentation algorithm by region merging, or compute Euler characteristics of a given surface).

Topological map definition is based on removal operations, and is carried out in several successive simplification steps. This allows to simplify each step of the construction, and to analyze each problem separately since we focus, for one construction step, on a particular problem. Moreover, this also allows to simplify the study of topological map properties, by studying separately each operation used to build each level, and prove progressively the properties.

Thanks to these levels, we have proved in this work that the topological map describes the represented labeled image by showing that no information is lost during the different simplifications. We have also proved that this map is minimal in number of cells because we cannot remove anything in this map without changing the topology, and representative of the image topology since the topological map does not depend on region geometry. For these reasons, the topological map is a good model for image processing. Indeed, it allows to retrieve most of the information needed by an image processing algorithm with a low computational cost. Moreover, the topological map regroups topological information (like adjacencies, that we can find in a RAG for example) and geometrical information (for instance the shape of a surface is given by the surfels which compose it) inside a single model.

Of course, all the information contained in the topological map can be retrieved directly by using only the image. However, this information can be retrieved efficiently by using the topological map. For example, if we want to retrieve all the regions adjacent to a given region R, we need to traverse all the voxels in the boundary of R in the image, while we can get this information only by looking at the faces of R in the topological map, and there are only few faces since our model is minimal.

We also have presented in this paper an incremental extraction algorithm which allows to extract the topological map of a labeled image in a single image scan. We have completely implemented this algorithm, and made several experiments in order to show how it is efficient (complexity in time) and to show the memory used to represent topological map.

Now we are working on using 3D topological maps for image processing. Some first works in 2D [5] have shown that using topological maps in existing image processing algorithms (for example segmentation algorithms) can improve the final result. We have also proposed split and merge operations using 3D topological maps [19]. Now we are working to propose a 3D image segmentation algorithm that only uses information computed from the topological map. We want to use topological map properties during the segmentation process, for example to compute topological features which can be used as criteria for the segmentation.

Another advantage of our model is that it can be used for visualization, and for interactive manipulation, for example to allow interactive correction of an expert. Our goal is to propose a complete image processing chain which is only based on the topological map and which allows to takes into account topological information of the analyzed image during each step of the process. Moreover, we are also working to extend this work in order to represent multi-scale segmentation by using generalized pyramids [46]. This can be used for example to handle a same image at different resolution levels, depending on the needs of a particular operation.

We also are interested in defining the topological map in any dimension. Of course some topological problems still remain since we do not know the topological classification of objects (as for example the problem of interlaced rings, see [36] for a discussion about these problems and a possible solution). Nevertheless, we can define the topological map in any dimension and use this tool in order to give a first characterization of discrete objects. It can be possible to add fictive cells into the topological map in order to keep each i-cell homeomorphic to an i-ball. This can be achieved without any particular problems, thanks again to our progressive simplification levels, where we can control if a removal operation leads to a topological modification. The problem is then to study how the objects obtained can be interpreted and how modification algorithms have to process these particular elements. Another problem concerns the way that these fictive elements have to be managed in order to obtain the minimal representation.

Acknowledgments

The author is very grateful to Valerie Gral for her careful reading and correction of this paper. Thanks also to Simon Thurston for his help concerning the English. Lastly, we would like to give our special acknowledgements to Yves Bertrand and Pascal Lienhardt for their encouragements and fruitful discussions about this work.

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 in Pattern Recognition*, pages 64–73, Ischia, Italy, may 2001. IAPR-TC15.

[4] 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.

[5] 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.

[6] A. Braquelaire, G. Damiand, J-P. Domenger, and F. Vidil. Comparison and convergence of two topological models for 3d image segmentation. In *Workshop on Graph-Based Representations in Pattern Recognition*, number 2726 in Lecture Notes in Computer Science, pages 59–70, York, England, june 2003.

[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 in Pattern Recognition*, 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 in Pattern Recognition*, 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 in Pattern Recognition*, Lyon, april 1997. IAPR-TC15. published in Advances in Computing (Springer).

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

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

[15] 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.

[16] G. Damiand and D. Arrivault. A new contour filling algorithm based on 2d topological map. In *Workshop on Graph-Based Representations in Pattern Recognition*, number 4538 in Lecture Notes in Computer Science, pages 319–329, Alicante, Spain, June 2007.

[17] G. Damiand, Y. Bertrand, and C. Fiorio. Topological model for two-dimensional image representation: definition and optimal extraction algorithm. *Computer Vision and Image Understanding*, 93(2):111–154, february 2004.

[18] G. Damiand and P. Lienhardt. Removal and contraction for n-dimensional generalized maps. In *Discrete Geometry for Computer Imagery*, number 2886 in Lecture Notes in Computer Science, pages 408–419, Naples, Italy, november 2003.

[19] 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.

[20] P. Desbarats. *Structuration des images segmentées 3D discrètes.* Thèse de doctorat, Université Bordeaux 1, décembre 2001.

[21] 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.

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

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

[24] 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.

[25] S Fourey and Malgouyres R. A digital linking number for discrete curves. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(7):1053–1074, 2001.

[26] 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.

[27] J. Françon. Discrete combinatorial surfaces. *Graphical Models and Image Processing*, 57(1):20–26, January 1995.

[28] J. Françon and Y. Bertrand. Topological 3d-manifolds : a statistical study of the cells. *Theoretical Computer Science*, 234(2):233–254, 2000.

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

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

[31] R. Klette and A. Rosenfeld. *Digital Geometry.* Morgan Kaufmann, 2004.

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

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

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

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

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

[37] 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.

[38] 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.

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

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

[41] J.R. Munkres. *Elements of Algebraic Topology.* Perseus Books, 1984.

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

[43] G. Reeb. Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. In *Comptes Rendus de l'Académie des Sciences*, volume 222, pages 847–849, Paris, France, 1946.

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

[45] A. Rosenfeld, T.Y. Kong, and A.Y. Wu. Digital surfaces. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, 53(4):305–312, july 1991.

[46] C. Simon, G. Damiand, and P. Lienhardt. Pyramids of n-dimensional generalized maps. In *Proceedings of 5th IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, volume 3434 of *Lecture Notes in Computer Science*, pages 142–152, Poitiers, France, Avril 2005.

[47] R. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

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