Representing and Segmenting 2D Images by Means of Planar Maps with Discrete Embeddings.

Achille Braquelaire, Jean-Philippe Domenger

LaBRI,Laboratoire Bordelais de Recherche en Informatique
Université Bordeaux 1, 351, cours de la Libération, 33405 Talence, France

Abstract

Representing the regions of a segmented image is an important aspect of image segmentation. Several different models have been proposed to represent the regions of a segmented image but most of them are dedicated to a specific method. Among the non hierarchical models, the model of planar maps with discrete embedding is certainly the most versatile one. Maps have the great advantage to provide a continuity of representation from the abstract mathematical model to the concrete implementation. They encode and provide most of topological and geometrical features required by segmentation algorithms and can be efficiently updated. In this paper we give an overview of the use of planar maps with discrete embedding in the context of image segmentation and we show how to design, implement and use a general environment for 2D image segmentation.

Key words: Image representation, image segmentation, combinatorial maps, discrete boundaries, feature extraction.

1 Introduction

Representing the regions of a segmented image is an important aspect of image segmentation. On the one hand the representation provides the features used to build the decomposition of the image into homogeneous regions. Thus the description of the decomposition must be powerful enough to allow to extract any required feature. On the other hand segmentation process generally build

Email address: {achille.braquelaire,jean-philippe.domenger@labri.fr} (Achille Braquelaire, Jean-Philippe Domenger).

solutions by refining progressively the decomposition. Thus the model used to described regions must be efficiently updated all along the segmentation steps. The simplest way to get an efficient model is to specialize it according to the minimal set of operations required by a specific method. But such a model is shown to be too restrictive to implement more general methods. Efficiency and versatility are thus two objectives which are hard to conciliate and when many models have been proposed to represent the regions of a segmented image, most of them are dedicated to a specific method.

The most basic method to represent segmented images is the array of labels [1–3] which consists in associating each pixel with a label such that all the pixels sharing a same label belong to a same region. This structure is very simple to implement but is ill-adapted to region splitting which involves a relabeling of all the pixels of the new sub-regions. Moreover this model does not efficiently provide topological features.

Hierarchical data structures allows to process images at different level of resolution. Several hierarchical models have been proposed such as quadtrees [4–7], pyramids [8–10], irregular pyramids [11–13], linked pyramids [14,15], dual pyramids [16], and more recently combinatorial pyramids [17]. Such data structures are very efficient to implement top-down region based algorithms. Starting at a coarse level of resolution, the initial image partition can be refined from level to level until reaching the resolution level of the original image. Nevertheless the hierarchical organisation of data restricts the possibilities of merging (and thus of interactive editing) and features like boundary geometry or neighbourhood are not immediate to extract.

Among the non hierarchical models, the model of embedded planar maps, or topological maps with discrete embedding [18–21], is certainly the most versatile one. Embedded maps encode both the geometry and the topology of the regions of a segmented image and allow free region editing, splitting, and merging. They are more general than region adjacency graphs [22,23] which have no geometric embeddings, which does not encode the whole topology of the segmented image, and which are ill-adapted to splitting. Two dimensional planar maps have been used for image editing [18,24–26,20], for image segmentation [19,21,27–30] and for video processing [31–33]. Maps have been generalized in the context of topological based scene modeling and in order to represent n-dimensional spaces [34], and several recent works have addressed the problem of the representation of 3D discrete images with maps for 3D image segmentation [35–39].

In the context of 2D image analysis, embedded maps provide an efficient framework to implement most of the operations involved by segmentation algorithms [40], such as domain reconstruction (or restoration) [41] which consists in traversing each point of the region domain, region splitting and merging,

point inclusion or point membership property [42,43] which consists in determining if a point is located inside or outside a given region, region localisation which consists in finding the region containing a given point, obtaining of geometric features (such as area, perimeter, boundary shape, etc.) [44,43,45] and of topological features (neighbourhood, surroundness [44], regions inside or outside a given boundary, counting holes [46,47], etc.).

In this paper we give an overview of the use of planar maps with discrete embedding in the context of image segmentation. Maps have the great advantage to provide a continuity of representation from the abstract mathematical model to the concrete implementation. We would like to illustrate that by showing how to design, implement and use a general environment for 2D image segmentation, from the mathematical model up to a real application. In the following section we recall how to represent the topology of a continuous segmented image with a set of planar maps. In section 3 we show how to associate such a set of maps with discrete boundaries and thus how to use this model to represent and segment discrete images. In sections 4 and 5 we briefly describe data structures and algorithms to use this model in the context of image segmentation. In section 6 we specify a small API that is enough to implement most of split and merge segmentation methods. Finally in section **??** we describe a full example of constrained segmentation method designed with this API to solve a real problem in the context of medical imaging.

2 Representing regions

A segmented image is a partition of an image into a set of regions, each region being a connected subset of points of the image. In the Euclidean plane a region is *simply connected* when its boundary is a simple closed curve also called Jordan's curve. Remark that a Jordan's curve defines two regions: a bounded region without hole which is called the inside region, and its complement which is an unbounded region with a hole which is called the outside region. When a bounded region is not simply connected, it has some holes and its boundary consists of several Jordan's curves, one of them — the outer boundary — surrounding all the other ones — the inner boundary(ies).

Consider the example developed in Fig 1. The image of Fig 1-a can be segmented into seven regions each one being a homogeneously textured area. The region boundaries have been drawn in white on the image. The wide region located in the bottom of the image, say A, is a simply connected region and its boundary consists of a unique Jordan's curve. On the other hand the background is a bounded region with two holes and thus is not simply connected. Its outer boundary is the boundary of the image and its inner boundary consists of two Jordan's curves, one of which being the outer boundary of the

region A and the other one being the largest oval-shaped contour.

In the Euclidean plane the boundaries of the regions of a segmented image can be partitioned in a natural way according to their neighbouring. Consider for instance the both small half-oval shaped regions of the running example. Both these regions share a part of boundary which is an horizontal line. Thus the boundary of these regions can be split into three *segments* of boundary which are respectively the horizontal line shared by the both regions, the part of boundary which is adjacent to the upper region and not to the lower one, and the part which is adjacent to the lower one and not to the upper one. Each segment of boundary is a Jordan's arc (a set of points homeomorphic to the real interval $[0, 1]$). This decomposition induces a graph the edges of which correspond to the Jordan's arcs and the vertices to the segment junctions. The graph induced by the boundaries of the segmented image of Fig 1-a is the graph shown in Fig 1-b. This graph has height vertices labeled from a to h and eleven edges. Remark that it is necessary to add an arbitrary vertex to associate with a graph edge the outer boundary of an isolated region. For instance the vertices labelled by a and h have been arbitrarily added on respectively the outer boundary of the image and the outer boundary of the region A.

A partition of the Euclidean plane into simply connected region is called a *topological planar map*, or simply a *planar map*. More formally a planar map [48] is a decomposition of the Euclidean plane into a finite set V of points, a finite set E of disconnected open Jordan's curves, each one having its extremities in V, and a finite set of simply connected regions whose boundaries are unions of elements of V and E. The elements of V and E are respectively the vertices and the edges of the map, and each simply-connected region is called a *face*. The faces corresponding to bounded regions are called the *finite faces* and the face corresponding to the unique unbounded region is called the *infinite face*.

Each connected component of the boundary graph of a segmented image is thus a planar map. For instance on the running example the boundary graph is decomposed into four planar maps which are the subgraphs defined respectively on the set of nodes $\{a\}$, $\{b, c\}$, $\{d, e, f, g\}$, and $\{h\}$. The second one has two vertices, three edges (labeled by 8, 9, and 10) and three faces, two finite faces which are the face surrounded by the sequence of edges $(8, 9)$ and $(8, 10)$, and an infinite face the boundary of which is the sequence $(9, 10)$.

A topological map can be efficiently encoded by a pair $\langle \sigma, \alpha \rangle$ of permutations defined on a set of labels called *darts*. Each dart can be seen as an half-edge of the topological map. Given an orientation of the plane, say counterclockwise, a vertex v of the map is describe by a circular sequence $(d_1, d_2, \ldots, d_n)$ which is the sequence of darts reaching it. This sequence is a cycle of the permutation σ and the notation $(d_1, d_2, \ldots, d_n)$ is a shortcut for $\sigma(d_1) = d_2, \ldots, \sigma(d_{n-1}) = d_n$, and $\sigma(d_n) = d_1$. The permutation σ is the set of all such cycles. The

permutation α is an involution without fixed point (each cycle is of length 2). Each cycle $(d, \alpha(d))$ of α encodes an edge of the map by linking two darts. Such a representation is called a *combinatorial map* [49].

If π is a permutation and x is an element which has an image by π, we denote by $\pi^*(x)$ the cycle of π that contains x. According to this notation, $\sigma^*(d)$ (resp. $\alpha^*(d)$) is the vertex (resp. the edge) that contains the dart d.

It may be convenient to encode the darts by positive and negative integers such that $\alpha(d) = -d$ [26]. According to this convention, a representation of the four combinatorial maps of the running example is shown in Fig 1-c. The set of darts is the set $\{-11, \ldots, -1, 0, 1, \ldots, 11\}$. The related permutations are:

- $\sigma_1 = (-1, 1)$ and $\alpha_1 = (-1, 1)$;
- $\sigma_2 = (8, 9, -10)(-8, 10, -9)$ and $\alpha_2 = (-8, 8)(-9, 9)(-10, 10)$;
- $\sigma_3 = (-2, 4, -7)(3, -5, -4)(5, 6, 7)(2, -6, -3)$ and $\alpha_3 = (-2, 2)(-3, 3)(-4, 4)(-5, 5)(-6, 6)(-7, 7)$;
- $\sigma_4 = (11, -11)$ and $\alpha_4 = (-11, 11)$.

A face the map is encoded by the circular sequence of darts encountered when turning around the face, clockwise for a finite face and counterclockwise for the infinite one. For instance the map $\langle \sigma_3, \alpha_3 \rangle$ has four faces which are the cycles $(-2, -6, 7)$, $(5\ -4, -7)$, $(-5, 6, -3)$ and $(4, 3, 2)$ (see also Fig. 2-a). The infinite face (in this case the last one) represents the unbounded region of the Euclidean plane which is the complement of the union of all the finite regions.

A given combinatorial map $\langle \sigma, \alpha \rangle$ may be associated with several different topological maps. That means that there are different ways to draw a combinatorial map on the plane. In fact there as many topological maps as there are faces in the combinatorial map. To draw a combinatorial map consists thus in first deciding which face is the infinite face and then to organize vertices and edges according to this choice. Once the infinite face is set, all the possible drawings are topologically equivalent.

A remarkable property of combinatorial maps is that the faces of the map $\langle \sigma, \alpha \rangle$ are encoded by the cycles of the permutation $\varphi = \sigma \circ \alpha$. Consider for instance a dart of the map $\langle \sigma_3, \alpha_3 \rangle$, say the dart -2. The cycle of -2 in the permutation φ is the circular sequence $(\sigma \circ \alpha)^*(-2)$. We have $\sigma(\alpha(-2)) = -6$, $\sigma(\alpha(-6)) = 7$, and $\sigma(\alpha(-7)) = -2$. The cycle $\varphi^*(-2)$ is thus the finite face $(-2, -6, 7)$.

The permutations φ and σ are defined on the same set of darts and the tuple $\langle \varphi, \alpha \rangle$ is also a combinatorial map. Moreover the maps $\langle \sigma, \alpha \rangle$ and $\langle \varphi, \alpha \rangle$ are dual. The orientation apart the dual map can be drawn by associating with each face of the primal map a vertex of the dual map, and by intersecting each edge of the primal map by an edge of the dual map, both edges being defined by the same pair of darts. For instance, on the running example, the dual map

of $\langle \sigma_3, \alpha_3 \rangle$ is the map $\langle \varphi_3, \alpha_3 \rangle$ with $\varphi_3 =$ (-2, -6, -7) (5, -4, -7)(-5, 6, -3)(4, 3, 2) and $\alpha_3 =$ (-2, 2)(-3, 3)(-4, 4)(-5, 5)(-6, 6)(-7, 7) (see Fig. 2-b). Let us underline that a dual map encodes the adjacency of the regions of a segmented image in a more general way than a region adjacency graph does, since there is in the dual map an edge for each segment of boundary shared by two adjacent regions.

Combinatorial maps are a very simple and elegant formalism to describe both planar maps and operations defined on them. For instance, on the running example, removing the edge $(-5, 5)$ consists in merging the face containing the dart 5 which the one containing the dart -5, i.e. the faces (-5, 6, -3) and (-7, 5, -4). This operation may be defined either on the map $\langle \sigma, \alpha \rangle$, by setting $\sigma'(3) =$ -4 and $\sigma'(7) = 6$, or on the map $\langle \varphi, \alpha \rangle$, by replacing both cycles (-5, 6, -3) and (5, -4, -7) by the cycle (6, -3, -4, -7) (see for instance [26,20,27,50,51] for a formal definition of these operations).

Remark that neither the vertices nor the edges need to be explicitly encoded. Each vertex, edge or face may be represented by any dart of the permutation cycle that represents it. Moreover by taking $\alpha(d) =$ -d the permutation α has not to be stored. So a combinatorial map, and thus a topological map, can be implemented with only an array of integers the size of which is twice the number of edges of the map.

It is noticeable that the same construction provides both a mathematical tool for formal proof and an efficient data structure for implementation purpose. Finally we may chose to implement either the primal representation or the dual one, each of then being obtainable from the other one with a negligible computational overhead.

Since a map encode the topology of only one connected component of the boundary graph, there are as many maps associated with the boundary graph as there are connected components. It is thus necessary to encode how these maps contribute to the representation of the topology of a non simply connected region of the segmented image.

It is of course possible to define an inclusion tree which nodes are the regions of the segmented image [46,52]. Nevertheless it is enough to encode the inclusion for only the infinite faces. It may be done in a straightforward way by adding to the model an *inclusion relation* which associates the finite face corresponding to the outer boundary of each non simply connected region with the list of the infinite faces corresponding to the outer boundary of the holes this region contains. The finite face is called a *parent face* and the infinite ones the associated *children faces*. For instance the inclusion relation of the four maps of Fig 1-c may be described by the relation: (1, -11), (1, 4), (-6 -10).

Of course this encoding is not unique since any dart of a face cycle may be used

to define the relation. We shall see in the next section that it is convenient to associate a label which each face. The relation of inclusion will then be defined in a unique way according to this labeling. Note that operations on maps may also modify this relation [20,27,51].

3 Representing discrete regions

In order to use combinatorial maps with discrete images it is necessary to get a representation of a discrete segmented image that can be associated with a set of planar maps. It requires the decomposition of a segmented image into discrete correspondents of vertices, edges and faces.

Discrete boundaries can be defined either in the image domain as pixel based contours [53–55,41,43] or in a discrete space different of the image space as interpixel contours [56,41,57–61]. Pixel based contours have several drawbacks when used to represent the boundaries of a segmented image. For instance if boundary segments are part of image regions a given boundary segment belongs to only one of its neighbouring regions, and if boundary segments are not part of image regions, the set of regions does not define a partition of the image. On the other hand, the interpixel representation provides a consistent topological framework [62] and makes it possible to define in a natural way discrete analogous of the edges of the boundary graph [61,20]. On the running example of Fig 1, a discrete discrete segmented image is displayed in Fig 1-d. The pixels are represented as colored unit square and the regions are the maximal 4-connected component. The image of Fig 1-e shows an example of boundaries defined with pixel contours and the one of Fig 1-f an example of interpixel boundaries which is compatible with the boundary graph of Fig 1-b.

Intuitively interpixel boundaries are drawn *between* pixels. If the image plane is the discrete space $\mathbb{Z} \times \mathbb{Z}$, the boundary plane is the *half-integer* plane which is obtained by translating the discrete plane $\mathbb{Z} \times \mathbb{Z}$ by $(-\frac{1}{2}, -\frac{1}{2})$ [61]. A point $p = (x_p, y_p)$ of the image plane and a point $p' = (x'_p, y'_p)$ of the boundary plane are *half-neighbours* if $|x_p - x'_p| = |y_p - y'_p| = \frac{1}{2}$. Each point of the image plane has four half-neighbours in the boundary plane and each point of the boundary plane has four half-neighbours in the image plane. Finally two adjacent points of the boundary plane share exactly two half-neighbours in the image plane.

Each point of the boundary plane having two or more half-neighbouring points belonging to different regions of a segmented image is a *boundary point*. Two adjacent boundary points are *linked* if their common half-neighbours belong to different regions, and the *rank* of a boundary point is the number of boundary points linked to it. Two boundary points b and b' are *connected* if there is a boundary path (or *contour*) linking them [63].

The boundary ∂r of a region r is the set of boundary points adjacent to both a point inside r and a point outside r. It consists of four-connected closed paths of boundary points. Each path is a sequence of boundary points $b_1, b_2, \ldots, b_n$ with $n > 1$ and such that b_i is linked to $b_i + 1 \quad \forall i$ with $1 \leq i < n$, and $b_i \neq b_j$, $\forall i, j$ with $i \neq j$. It can be shown that such boundaries are discrete Jordan's curves by embedding both the boundary and the image planes into the Khalimsky's plane [60,64].

According to these definitions the boundary of a discrete segmented image can be decomposed into *segments* and *nodes* in the same way than the boundary graph of a continuous segmented image can be decomposed into edges and vertices. The nodes are either *natural nodes* or *arbitrary nodes*. Natural nodes are the boundary points of rank greater than two. Arbitrary nodes are points arbitrary selected on each boundary component consisting of a unique closed contour (one arbitrary node selected for each closed contour). A *segment* is then a maximal contour without node. On the example of Fig. 3, the boundary points are displayed with disks. The nodes are boundary points displayed with grey disks. There are two nodes, each one of rank 3, and three segments. On this example both nodes are natural nodes.

Now a segmented image associated with a boundary plane can be partitioned into a set of nodes, a set of discrete Jordan's curves joining two nodes, and a set of 4-connected regions. If all regions are simply connected, such a structure is the discrete analogous of a topological map and is called a topological map with discrete embedding, or a discrete map. When some regions are not simply connected, there are several discrete maps associated by an inclusion relation as there are several topological maps in the Euclidean case.

Like topological maps, discrete maps can be described by pairs of permutations. The analogous of a dart of a combinatorial map is a *geometrical dart*. Each geometrical dart is associated with an end of segment. Is p is a boundary point at the end of a segment and n the node linked to p the associated geometrical dart is the pair (p, δ) where δ is the elementary direction from n to p. By this way a discrete map induces a combinatorial map defined as follows:

– each geometrical dart of the discrete map is associated with a dart of the combinatorial map;

– each pair of dart associated with the two geometrical darts of a same segment forms a cycle of the permutation α;

– each sequence of darts associated with a sequence of geometrical darts sharing a same node forms a cycle of the permutation σ, according to the order of the geometrical darts around the node [63].

4 From model to data structure

The next step to get an image segmentation environment is now to define a minimal data structure both to encode the models previously described and to perform feature extraction and representation updating involved by segmentation algorithms.

4.0.0.1 Topological data structure. As stressed above the data structure used to represent a map is simply an array of integers indexed by darts. This array may encode either σ or φ. We have chosen to encode the second one. Moreover, all the maps involved in the representation of the topology of a segmented image are disjointed in the sense that each map $\langle \varphi_i, \alpha_i \rangle$ is defined on a set of darts D_i disjointed from the other ones. We define the permutation φ by $\varphi = \cup \varphi_i$ and the permutation α by $\alpha = \cup \alpha_i$. We have $\varphi_{|D_i} = \varphi_i$ and $\alpha_{|D_i} = \alpha_i$, and we may thus simply denote by $\langle \varphi, \alpha \rangle$ the set of maps $\{\langle \varphi_i, \alpha_i \rangle\}$. In the same way σ denotes the union of all the σ_i permutations. The permutation φ is encoded by an array `phi` of integers, and for any dart d, if D_i is the set of darts that contains d, we have `phi[d]` $= \varphi_i(d)$, and `phi[-d]` $= \sigma_i(d)$.

In order both to address regions and to define the inclusion relation it is convenient to define a *region labeling*. Therefore each dart is labelled with a labeling function λ such that two darts have the same label by λ if and only if they belongs to the same cycle of φ. Thus for each pair of darts (d, d') we have: $\lambda(d) = \lambda(d') \Leftrightarrow \varphi^*(d) = \varphi^*(d')$.

Conversely, a function β associates each face label f with a dart of the corresponding face cycle such that $\lambda \circ \beta(f) = f$, and for each dart d, $\beta \circ \lambda(d) \in \varphi^*(d)$. This dart, called the *canonical dart* of f, is an arbitrary entry point in the dart face cycle. The face of label f is thus the cycle $\varphi^*(\beta(f))$, and for each face label f we have: $\forall d \in \varphi^*(\beta(f)),\ \lambda(d) = f$. Both the function λ and the function β are encoded by an array of integers (respectively the array `lambda` and the array `beta`). A labeling of the maps of the running example is given in Fig 4.

For each face f associated with a holed region, the relation ***children*** gives the list of infinite faces associated with the holes. Conversely the relation ***parent*** gives for each infinite face f the finite face f' such that $f \in$ ***children***(f'). The relations ***parent*** and ***children*** are the inclusion relations. Remark that each region is associated which only one finite face and possibly several infinite faces. Each region may thus be labeled in a unique way by the label of its associated finite face. For instance the background region of the segmented image of the running example is labeled by 1 which is also the label of its associated finite face. The other faces associated with this region are the two

infinite faces labeled respectively by 8 and 10. The inclusion relations of the running example are given in Fig 5. Two regions r and r' may be neighbouring regions because their associated faces are adjacent in one of the dual maps: $\exists d \in \varphi^*(\beta(r))$, and $\exists d' \in \varphi^*(\beta(r'))$ such that $\alpha(d) = d'$. But two regions may also be neighbouring regions according to the ***parent*** (or ***children***) relation. That leads us to consider three different neighbouring modes on which we shall come back in section 6.

4.0.0.2 Geometrical data structure. We have now to define a data structure to represent the geometrical embedding of maps, i.e. the geometry of interpixel boundaries. It is possible to associate each segment with a local embedding. This embedding can be defined independently for each segment, either explicitly by a sequence of elementary steps or implicitly in a procedural way. Another solution consists in defining a global embedding of the whole boundaries by encoding the whole part of the boundary plane corresponding to the image domain. The main advantage of the first solution is to provide simplest updating, especially for segment removing. However the local encoding is ill-adapted to the splitting of a region and to geometrical editing of regions. The solution retained here is thus to encode explicitly the boundary plane by using a global data structure called boundary image.

The boundary plane and the image plane are isomorphic. Thus an image and its boundary image have about the same number of elements (in fact when an image is of size $N \times M$, its boundary image is of size $(N+1) \times (M+1)$ in order to encode the image outer boundary). The boundary image encodes both boundary points and link. A boundary point may have from two to four links. An element of the boundary image without links cannot be a boundary point and conversely an element of the boundary image with links is a boundary point. Thus it is only necessary to explicitly encode links. Moreover the linking relation is a symmetrical relation. Thus it is only necessary to encode links along two of the four possible directions. For instance if we chose to encode upward and rightward links it is possible to know if a boundary point p' is linked downward to a boundary point p by checking if its downward neighbour p is linked upward to it. Since there is no way to recognize an arbitrary node it is also necessary to mark nodes in the boundary image. Thus only three bits are needed to store each entry of the boundary image.

To sum up, if the domain of an image is the set of points $\{(i,j) \in \mathbb{Z}^2,\ 0 \leq i < H,\ 0 \leq j < W\}$ the associated boundary domain is the set $\{(i-\frac{1}{2},\ j-\frac{1}{2}),\ (i,j) \in \mathbb{Z}^2,\ 0 \leq i \leq H,\ 0 \leq j \leq W\}$. The boundary domain is encoded by an array `B` called *boundary image* where the entry `B[i][j]` encodes three boolean informations: whether the boundary point p of coordinates $(i-\frac{1}{2}, j-\frac{1}{2})$ is a node, whether p is linked upward to either another boundary point or to a node, and whether p is linked rightward to either another boundary point

or to a node. One can retrieve the geometry of a segment by following links from one of its geometrical darts to the other one.

4.0.0.3 Correspondence between topology and geometry. The correspondence between topological and geometrical data structures is encoded by a associating combinatorial and geometrical darts. A geometrical dart is encoded by a point (which is a pair of coordinates) and a direction (upward, leftward, downward or rightward). The relation between geometrical and topological darts is stored in an array of geometrical darts indexed by topological darts. A hash table with pairs of node coordinates as keys is used to avoid to traverse this array when looking for a geometrical dart and get an efficient access to the topological representation from the geometrical one.

5 Overview of algorithms

We have described in the previous section the whole data structure used to implement a topological and geometrical representation of a segmented image. Let us now give a short overview of related algorithms.

It is possible to find the region that contains a point by scanning the boundary image from this point until reaching a segment. If the boundary image is scanned horizontally (for instance rightward) it is enough to look for the first encountered vertical link. Once the segment is reached the traversal continue by following this segment until reaching a geometrical dart. The associated topological dart d identify a face which is an infinite face if the scanning has reached the outer boundary of a connected component and a finite face in the other case. The region is then given by the label $\lambda(d)$ in the case of a finite face and by the label $\boldsymbol{parent}(\lambda(d))$ in the case of an infinite face.

The boundary of a region f is obtained by traversing the cycle $\varphi^*(\beta(f))$ and the cycles $\varphi^*(\beta(f'_i))$ of the faces f'_i of $\boldsymbol{children}(f)$. The geometry of this boundary is obtained by traversing the segments corresponding to the geometrical darts associated with the topological darts of $\varphi^*(\beta(f)) \cup \bigcup_i \{\varphi^*(\beta(f'_i))\}$.

According to the orientation of the plane the finite faces are traversed clockwise and the infinite ones counterclockwise. The domain of a region can thus be reconstructed by building the list of all image points that are on the left of an upward link or on the right of a downward link and by sorting this list relatively to lines and then to columns. The resulting list is exactly the list of horizontal lines covering the region domain.

The construction of the representation of a segmented region r can be done with a complexity $K \times |r|$ where $|r|$ is the number of points of the region, and

where K is a constant equal to 7 in the worst case. The topological updates involved by split and merge can be expressed by mean of elementary operations on the permutations and on the inclusion relations. The cost of geometrical updating involved by splitting and merging is $O(\sum_{s \in S} |s|)$ where S is the set of inserted or removed segments. All these algorithms have been described in detail in previous works [26,20,27,50,63,51].

6 Designing a toger API

In order to validate the interest of the model of planar maps with discrete embedding in the contex of image segmentation, we describe in this section how to interact with such an environment, that we call *toger kernel* in the following of this paper. We first give a short description of the types of objects manipulated by such an API and then of the main functions of the interface.

6.0.0.4 Types. It is convenient to provide both a high level and a low level of interaction. At the high level (or region level) the objects returned by the functions of the API are regions or lists of regions, *paths* or lists of paths which encode the geometry of region boundaries, and *domains* which encode the geometry of regions. A region is a label encoded by an integer. A path is a sequence of adjacent points that can be encoded by an array or in a more suitable way by a generator, i.e. a set of functions { `first`, `last`, `next`, `previous`, `length` }, that is a usual interface to traverse the elements of a list. The domain of a region r is encoded by a sequence of pairs of points. A point is simply a pair of integer coordinates. Each pair defines a *span* of the region r which is an horizontal maximal line (P_{2i}, P_{2i+1}) belonging to r. The point P_{2i} (resp. P_{2i+1}) is thus located on the right (resp. the left) of a vertical interpixel boundary element of r. The list of pairs of points is the list of all the spans of the domain of r. Finally we have seen that two adjacent regions can be related according to three different neighbouring modes. The type `neighbouring_mode` is used to denote these modes. The high level of interaction does not require any knowledge of the internal representation. Conversely the low level (or map level) provides interactions directly with the topological maps. The data types used at map level are the darts, the face labels and the geometrical darts. As seen above, both darts and face labels are elementary types encoded by integers. We have seen that a region label is the label of its finite face. Thus region labels are a subset of face labels. The geometrical darts are pairs consisting of a point and a direction. Finally, the type `toger` is used to refer to the whole representation. All these types are summarized in Table 1.

dart	set of integers
face	set of integers
region	set of integers
neighbouring_mode	{ *DIRECT*, *INNER*, *OUTER*, *ANY* }
point	pair of coordinates
direction	{ *UPRIGHT*, *LEFTWARD*, *DOWNWARD*, *RIGHTWARD* }
geometrical_dart	point and direction
path	generator of points
domain	list of pairs of points
toger	topological and geometrical representation

Table 1
Types of the toger API.

6.0.0.5 Side effect functions. Among the side effect functions (i.e. functions that modify the representation they receive in parameter) we need a function that builds the representation of a segmented region: *split_region* : *toger* × *region* × (*point* × *point* → *boolean*) → *list of regions*. The sub-regions of the segmented region are implicitly described by a partitioning function f : *point*×*point*→*boolean* which receives two neighbouring points in parameter and returns *true* if these points belong to a same region and *false* if not. The function *split_region* updates the topological and the geometrical representation of a region (second parameter) according to a representation (first parameter) and to a partitioning function (third parameter); the result is the list of labels of the new sub-regions.

In order to remove a boundary shared by two adjacent regions we need a function that merges these regions: *merge_regions* : *toger* × *region* × *region* → ∅. This function modifies a representation (first parameter) by merging two regions (second and third parameters). The label of the region resulting from the merge is the same as the one of the first region. Note that the removed part of boundary is not necessarily connected and that this operation may disconnect to components in the boundary graph. This function is the only low level deleting function that can be defined in such an API because other basic delete operations (like segment removing for instance) does not guarantee that the consistency of the representation is preserved. Higher level deleting functions can be considered like for instance the removing of all the region being inside a closed contour.

It may also be convenient to modify the representation by inserting a new contour. It can be done by a function like *insert_contour* : *toger* × *path* → *list of regions*. In order to preserve the consistency of the representation, an

inserted contour must be either a closed contour or a contour joining two nodes [26]. Thus the contour to insert (second parameter) must be preprocessed by the function *insert_contour* before being inserted, in order to satisfy one of these conditions. All these functions modify both the geometry and the topology of the representation. It is also possible to consider functions that modify only the geometry, which can be for instance useful to locally smooth a boundary.

6.0.0.6 Point inclusion and region localisation. We also need function to search for a region. We have shortly described how to retrieve the region containing a given point. This functionality can be provided by a function like *find_region* : *toger* $\times$ *point* $\rightarrow$ *region* which returns the label of the region containing a point (second parameter) according to a representation parameter). It is also possible to define a function *belongs_to* : *toger* $\times$ *point* $\times$ *region* $\rightarrow$ *boolean* that checks if a point parameter) belongs to a region (third parameter) according to the representation (first parameter),

6.0.0.7 Geometrical features. The main geometrical features are the domain of a region which can be obtained by a function like `region_domain` : *toger* $\times$ *region* $\rightarrow$ *domain*, and the geometry of its boundary which can be obtained by a function like *region_boundary* : *toger* $\times$ *region* $\rightarrow$ *list of closed_path*. Since a boundary may consist of several closed path, it its convenient to get the outer boundary as the first element of the returned list. It may also be convenient to get only the outer boundary: *region_outer_boundary* : *toger* $\times$ *region* $\rightarrow$ *closed_path* and the part of boundary shared by two adjacent regions by using the function *regions_common_boundary* : *toger* $\times$ *region* $\times$ *region* $\rightarrow$ *list of paths*. Geometrical features of segment may also be useful. That leads us to consider functions like *segment* : *toger* $\times$ *dart* $\rightarrow$ *path*, that returns a path which describes the geometry of the segment associated with a dart, and *segment_length* : *toger* $\times$ *dart* $\rightarrow$ *integer*, that returns the segment length. Several different length estimators can be used, such as the number of elementary steps or the length of the associated Euclidean path [65].

6.0.0.8 Low level topological features. Defining the interaction with the low topological level is straightforward. Two functions *lambda* : *toger* $\times$ *dart* $\times$ *face* and *beta* : *toger* $\times$ *face* $\rightarrow$ *dart* implement the functions λ and β associating darts with labels. The functions *alpha* : *toger* $\times$ *dart* $\rightarrow$ *dart*, *sigma* : *toger* $\times$ *dart* $\rightarrow$ *dart*, and *phi* : *toger* $\times$ *dart* $\rightarrow$ *dart* implements the permutations α, σ, and φ used to traverse the combinatorial representation of maps. Finally the connected components can be traversed with function *parent* : *toger* $\times$ *face* $\rightarrow$ *face* that returns for a face f the face $\boldsymbol{parent}(f)$ if f is infinite or a void value if f is finite, and with the both functions *first_child* :

toger × *face* → *face* and *next_child* : *toger* × *face* → *face* that implement the relation ***children***.

6.0.0.9 Topological marking. It is sometimes necessary to traverse the graph of regions with respect to some marking. By associating marks to dart it is possible to maintain the consistency of the marking when doing a side effect operation. When an edge (d_1, d_2) is split into two adjacent edges (d_1, d_2) and (d'_1, d'_2), the dart d'_i receives a copy of the marks of the dart d_i. By this way, when a region is split into subregions, it is possible without overhead to preserve the consistency of marks on faces and more generally on any contour. It is also possible to preserve the consistency of contour marking through region merging. When a region r' is merged with a region r, and if the common boundary of r and r' is a unique edge e, the marking can be preserved by marking the face associated with r before removing e. If there are several edges $e_1, ..., e_n$, the removing of these edges disconnects from one to $n-1$ components of the map. In this case it is also necessary to unmark some edges in these components. It is possible to mark an oriented edge, an non-oriented edge (by marking a cycle of α), a node (by marking a cycle of σ), or a face boundary (by marking a cycle of φ). The number of marks depends on the space allocated for each mark. By allocating one byte per dart we get height marks per dart which is enough for most of traversal algorithms and which the memory cost is reasonable with modern computers. The type *set of flags* is a boolean combination of flags used to manipulate the marks.

The interface of marking operations can be defined like the one of geometrical feature functions. The functions *mark_region_boundary* : *toger* × *region* × *set of flags* → ∅ and *unmark_region_boundary* : *toger* × *region* × *set of flags* → ∅ respectively marks and unmarks the boundary of a region. Similarly, the functions *mark_outer_boundary* : *toger* × *region* × *set of flags* → ∅ and *unmark_outer_*
boundary : *toger* × *region* × *set of flags* → ∅ respectively marks and unmarks only the outer boundary of a region, and *mark_common_boundary* : *toger* × *region* × *region* × *set of flags* → ∅ and *unmark_common_boundary* : *toger* × *region* × *region* × *set of flags* → ∅ the part of boundary shared by two regions. Finally we need functions like *mark_all_darts* : *toger* × *set of flags* → ∅ and *unmark_all_darts* : *toger* × *set of flags* → ∅ to set and clear marks on all the darts, and a function *dart_is_marked* : *toger* × *dart* × *set of flags* → *boolean* to check if a dart is marked. Each time, a parameter of type *set of flags* is used to specify which is the mark (or are the marks) to modify.

6.0.0.10 High level topological functions. The neighbourhood of a region may be considered according to the three neighbouring modes. Both of the neighbouring modes, the *inner* and the *outer* ones, are oriented relations.

The third one is a symmetric relation. A region r' is a direct neighbour of a region r if the intersection of their outer boundaries is not empty. A region r' is an inner neighbour of a region r if the outer boundary of r' intersects the internal boundary of r. In that case r is an outer neighbour of r'. In other terms a region r' is a direct neighbour of a region r if $\exists d \in \varphi^*(\beta(r))$, and $\exists d' \in \varphi^*(\beta(r'))$ such that $\alpha(d) = d'$. A region r' is an inner neighbour of a region r (or r is an outer neighbour of r') if $\exists d \in \varphi^*(\beta(r'))$ such that $\textbf{\textit{parent}}(\lambda(\alpha(d))) = r$. The neighbouring modes of the regions of the running example are given in Fig. 7.

The neighbourhood of a region may also be considered according to the topological marking. The neighbouring relation is thus restricted such that two regions are not considered as neighbouring regions if their common boundary is marked, according to a given set of marks. That leads us to define the interface of the function *region_neighbourhood* that gives the neighbourhood of regions as a function of signature: *toger* $\times$ *region* $\times$ *neighbouring_mode* $\times$ *set of flags* $\rightarrow$ *list of regions*. It is also useful to have a function that gives any neighbouring region of a given one, for instance when looking for a neighbouring region to merge with. So the function *any_region_neighbour* : *toger* $\times$ *region* $\times$ *neighbouring_mode* $\times$ *set of flags* $\rightarrow$ *region* returns a neighbouring region of a region according to a neighbouring mode and a set of marks. The function *are_neighbours* : *toger* $\times$ *region* $\times$ *region* $\times$ *neighbouring_mode* $\rightarrow$ *boolean* checks if two regions are neighbouring regions. Finally more general functions may be easily defined, such as a the function *inner_regions* : *toger* $\times$ *region* $\rightarrow$ *list of regions* which gives the list of regions located inside the outer boundary of a given region, or the function *all_regions* : *toger* $\rightarrow$ *list of regions* which gives the list of all the currently defined regions.

6.0.0.11 Updating the region attributes. When implementing a segmentation method it is generally necessary to attach attributes to regions. These attributes have to been initialized and/or updated when regions are split or merged. It may raise a problem of software engineering when the split or the merge is activated by a program module which is not the one which has in charge the update of attributes. In that case the activation of attribute update should be done automatically by the kernel. A classic solution to this problem consists in attaching processings to each side effect function. For instance, we can attach to the *region_split* action a function f which the signature is *region* $\times$ *region* $\times$ *parameter* $\rightarrow \varnothing$, where *parameter* is a type of generic parameter (like `Object` in Java or `void*` in C and C++). Each time the *region_split* function is executed, the function f is automatically called by the kernel with the regions to be merged as parameters. It may be also possible to attach a list of functions, or functions being called either before or after the side effect. This mechanism is implemented for each side effect function.

For instance let us suppose we want to attach descriptive statistical moments to each region. These moments are be used to compute features like mean or variance. Since these moments are additive, when two regions are merged, the moments can be updated by simply adding themselves the moments of the same order of each region and storing the result in the remaining region [27]. This can easily be done by the way of the mechanism described above.

7 Conclusion and perspectives

In this paper we have given an overview of the use of planar maps with discrete embedding in the context of image segmentation. We have described how to represent the topology of a segmented image with a set of planar maps and how to associate with planar maps the boundaries of a discrete segmented image. We have show how this model can be used to implement most of operations involved in the design of segmentation algorithms, such as reconstruction of the geometry of the boundary and of the domain of a region, point inclusion, region localisation, obtaining of geometric and topological features, and updating involved by region splitting and merging. We have described the data structures and the algorithms that permits to use this model in the context of image segmentation.

This model has been used with success to implement various segmentation applications. We have thus decided to capitalize on this experiment to develop a portable general image representation library implementing the model and the algorithms summarized in this paper. A first version of the API of this library have been presented in section 6. Finally we have illustrated this approach by describing a full example of constrained segmentation method designed with this API. The General Image Representation Library will be available under LGPL in the course of year 2005.

In parallel we are working on the extension on this model to represent and segment three-dimensional discrete images. Two different models were proposed, one of them by Braquelaire, Desbarats, Domenger and Wütrich [35,38,66,67] and the other one by Bertrand, Damiand and Fiorio [36,37,68]. We are currently collaborating to mix both models and design an implementation of a 3D general image representation library [39].

References

[1] F. Ferri and E. Vidal. Colour image segmentation and labeling through multiedit-condensing. *PRL*, 13:561–568, 1992.

[2] M. Suk. A new segmentation technique based on partition mode test. *PaRe*, 16(5):469–480, 1983.

[3] C. J. Nicol. A systolic approach for real time connected component labeling. *Computeer vision and Image understanding*, 61(1):17–31, January 1995.

[4] G.M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, 1(2):145–153, April 1979.

[5] R.C. Dyer, A Rosenfeld, and S Hanan. Region representation: Boundary codes from quadtrees. *ACM: Graphics and Image Processing*, 23(3):171–179, March 1980.

[6] H. Samet. Region representation: Quadtrees from boundary codes. *ACM : Graphics and Image Processing*, 23(3):163–170, March 1980.

[7] C. Ang, H. Samet, and C.A. Shaffer. A new region expansion for quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(7):682–686, 1990.

[8] S.L. Tanimoto. Image data structures. In S. L. Tanimoto and A. Klinger, editors, *Structured Computer Vision*, pages 31–56. Academic Press, New York, 1980.

[9] M. Bister, J. Cornelis, and A. Rosenfeld. A critical view of pyramid segmentation algorithms. *Pattern Recognit Letter.*, 11(9):605–617, 1990.

[10] D. Willersinn and W.G. Kropatsch. Dual graph contraction for irregular pyramids. In *International Conference on Pattern Recogntion D: Parallel Computing*, pages 251–256, Jerusalem, Israel, 1994. International Association for Pattern Recognition.

[11] P. Meer. Stochastic image pyramids. *Computer Vision Graphics Image Processing*, 45:269–294, 1989.

[12] A. Montanvert, P. Meer, and A. Rosenfeld. Hierarchical image analysis using irregular tesselations. *PAMI*, 13(4):307–316, 1991.

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

[14] M. Pietikainen, A. Rosenfeld, and I. Walter. Split and link algorithms for image segmentation. *Pattern Recognition*, 15(4):287–298, 1982.

[15] J.M. Jolion and A. Montanvert. The adaptative pyramid : A framework for 2D image analysis. *Computer Vision, Graphics and Image Processing : Image Understanding*, 55(3):339–348, May 1992.

[16] W. Kropatsch. Preserving contours in dual pyramids. pages 563–565, 1988.

[17] L. Brun and W. Kropatsch. Introduction to combinatorial pyramids. In G. Bertrand and A. Imiya, editors, *Digital and image geometry*, volume 2243 of *LNCS*, pages 108–127. Springer, 2001.

[18] J.P. Braquelaire and P. Guitton. A model for image structuration. In *Proc. of Computer Graphics International'88*, pages 426–435. Springer, 1988.

[19] K.C. Keeler. *Map Representations and Optimal Encoding for Image Segmentation*. Phd thesis, Harvard University, 1990.

[20] J.-P. Domenger. *Conception et implémentation du noyau graphique d'un environnement 2D$\frac{1}{2}$ d'édition d'images discrètes*. PhD thesis, Univ. Bordeaux 1, avril 1992.

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

[22] P. Jasiobedzki. Adaptive adjacency graphs. In *Proceedings, SPIE Geometric methods in Computer Vision*, San Diego, CA, 1993.

[23] L.G. Shapiro. Connected component labeling and adjacency graph construction. In Kong and Rosenfeld, editors, *Topological Algorithms for Digital Image Processing (Machine Intelligence and Pattern Recognition, Volume 19)*. Elsevier, 1996.

[24] P. Baudelaire and M. Gangnet. Planar maps: an interaction paradigm for graphic design. In *Proc. of CHI'89*, pages 313–318. Addison-Wesley, 1989.

[25] M. Gangnet, J.C. Hervé, T. Pudet, and J.M. VanThong. Incremental computation of planar maps. In *Proc. of SIGGRAPH'89*, 1989.

[26] J.P. Braquelaire and P. Guitton. $2\frac{1}{2}$ scene update by insertion of contour. *Computer and Graphics*, 15(1):41–48, 1991.

[27] L. Brun. *Segmentation d'images couleur à base topologique*. PhD thesis, Université Bordeaux 1, december 1996.

[28] C. Fiorio. A topologically consistent representaion for image analysis: the topological graph of frontiers. In S. Miguet, A. Montavert, and S. Ubéda, editors, *Lectures Notes in Computer Sciences*, volume 1176, pages 151–162, 1996.

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

[30] U. Köthe. Xpmaps and topological segmentation – a unified approach to finite topologies in plane. In A. Braquelaire, J.-O. Lachaud, and A. Vialard, editors, *Proc of DGCI'02*, volume 2310 of *LNCS*, pages 22–33. Springer, 2002.

[31] J. Benois-Pineau, A. Braquelaire, and A. Ali-Mhammad. Interactive fine object-based segmentation of generic video scenes for object-based indexing. In Ebroul Izquierdo, editor, *proc. of WIAMIS'2003*, pages 200–203, 2003.

[32] J. Benois-Pineau, G. Peretie, and A. Braquelaire. Adaptive video pre-processing for bit-rate reduction in object-based predictive coding schemes. In *proc. of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, pages 27–30, Orlando, Florida, July 2003.

[33] P. Krämer, J. Benois-Pineau, and J.P. Domenger. Scene similarity measure for video content segmentation in the framework of rough indexing paradigme. In *Proc of AMR'2004*, pages 144–155, 2004.

[34] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasimanifold. *Int. Journ. of Comp. Geom. and Appl.*, pages 275–324, 1994.

[35] J.P. Braquelaire, P. Desbarats, J.P. Domenger, and C. Wütrich. A topological structuring for aggregates of 3D discrete objects. In *Proc of GBR'99, Osterreichische Computer Gesellschaft, ISBN 3-8580-126-2*, pages 193–202, 1999.

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

[37] 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.

[38] P. Desbarats. *Strucuration d'images segmentées 3D discrtes.* Thèse de doctorat, Université Bordeaux 1, décembre 2001.

[39] A. Braquelaire, G. Damiand, J.P. Domenger, and F. Vidil. Comparaison and convergence of two topological models for 3D image segmentation. In *proc. of GBR 2003, ISBN 887146579-2*, pages 32–43, 2003.

[40] L. Brun, J.P. Domenger, and J.P. Braquelaire. Discrete maps: a framework for region segmentation algorithms. In J.M Jolion and W Kropatsch, editors, *Proc. of Graph based Representations, GbR'97*, pages 83–92. Springer-Verlag, 1998.

[41] A. Rosenfeld and A. Kak. *Digital Picture Processing*, volume 2. Academic Press, 1982.

[42] R.G. Loomis. Boundary networks. *Communications of the ACM*, 8(1):44–48, 1965.

[43] L.W. Chang and K.L. Leu. A fast algorithm for the restoration of images based on chain codes descriptions and its applications. *Computer Vision, Graphics and Image Processing : Image Understanding*, 50:296–307, 1990.

[44] A. Rosenfeld. *Picture Languages.* Academic Press, 1979.

[45] S.V. Raman, S. Sarkar, and K.L. Boyer. Hypothesizing structures in edge-focused cerebral magnetic resonance images using graph-theoretic cycle enumeration. *CVGIP: Image Understanding*, 57(1):81–98, 1993.

[46] P. Morse. Concepts of use in contour map processing. *Communications of the ACM*, 12(3):147–152, 1969.

[47] A. Rosenfeld. Digital topology. *Amer. math. monthly*, 86:621–630, 1979.

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

[49] R. Cori. Un code pour les graphes planaires et ses applications. Thèse d'état de l'université Paris VII, and Astŕisque 27, 1973 and 1975.

[50] L. Brun and J.P. Domenger. A new split and merge algorithm based on discrete map. In *Proc.of WSCG'97*, pages 21–30, 1997.

[51] L. Brun, J.P. Domenger, and M. Mokhtari. Incremental modifications on segmented image defined by discrete maps. *Journal of visual commmunication and Image representation*, 14:251–290, 2003.

[52] M. Gangnet and J.M. Van Thong. Robust boolean operations on 2D paths. In *Proc. of COMPUGRAPHICS'91*, pages 434–444, 1991.

[53] H. Freeman. On the encoding of arbitrary geometric configurations. *IRE Trans. Electr. Compu.*, 10:260–268, June 1961.

[54] R.D. Merrill. Representation of contours and regions for efficient computer search. *Communications of the ACM*, 16(2):69–82, 1973.

[55] T. Pavlidis. *Algorithms for Graphics and Image Processing.* Computer Sci., Washington, 1982.

[56] R. Brice and C.L. Fennema. Scene analysis using regions. *Artificial intelligence*, 1:205–226, 1970.

[57] T.Y. Kong and A. Rosenfeld. Digital topology: Introduction and survey. *Computer Vision, Graphics and Image Processing : Image Understanding*, 48:357–39, 1989.

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

[59] H. Bieri. Hyperimages – an alternative to the conventional digital images. In *Eurographics'90 proceedings*, pages 341–352, 1990.

[60] E. Khalimsky, R. Kopperman, and P.R. Meyer. Boundaries in digital planes. *Journal of applied Math. and Stocastic Analysis*, 3:27–55, 1990.

[61] J.P. Braquelaire and J.P. Domenger. Intersection of discrete contours. In *Proc. of Compugraphics'91*, pages 434–444, 1991.

[62] E. Ahronovitz, J.P. Aubert, and C. Fiorio. The star-topology: a topology for image analysis. In *5th DGCI Proceedings*, pages 107–116, 1995.

[63] J.P. Braquelaire and J.P.Domenger. Representation of segmented images with discrete geometric maps. *Image and vision Computing*, 17:715–735, 1999.

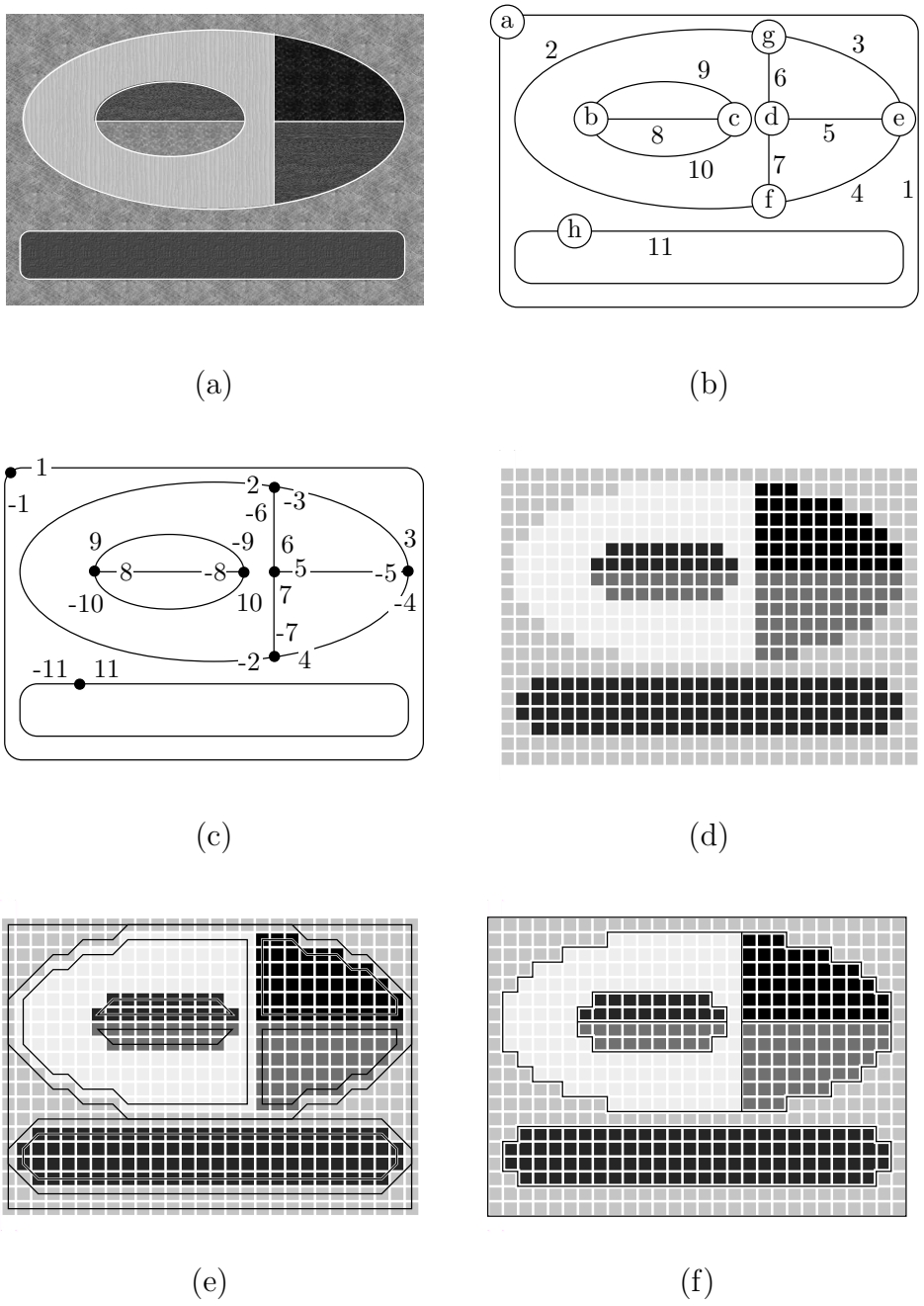
[64] E. Khalimsky, R. Kopperman, and P.R. Meyer. Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications*, 36:1–17, 1990.

[65] A. Vialard. Geometrical parameters extraction from discrete paths. *Lecture Notes in Computer Science*, 1176:24–35, 1996. Discrete Geometry for Computer Imagery'96.

[66] A. Braquelaire, P. Desbarats, and J.P. Domenger. 3D split and merge with 3-maps. In *In proc. of GBR 2001, ISBN 887146579-2*, pages 32–43, 2001.

[67] P. Desbarats and J.-P. Domenger. Retrieving and using topological characteristics from 3D discrete images. In *Preoceedings of the 7th Computer Vision Winter Workshop*, pages 130–139. PRIP-TR-72, 2002.

[68] Y. Bertrand, G. Damiand, and C. Fiorio. Topological map : minimal encoding of 3D segmented images. In *In proc. of GBR 2001, ISBN 887146579-2*, pages 63–73, 2001.



(a) (b)

(c) (d)

(e) (f)

Fig. 1. Representation of the topology and of the geometry of a segmented image: (a) a segmented continuous image with its region boundaries, (b) the associated boundary graph, (c) the representation of this graph by combinatorial maps, (d) a discrete segmented image with the same topology, a representation of the geometry of regions (e) with pixel based contours, and (f) with interpixel contours.

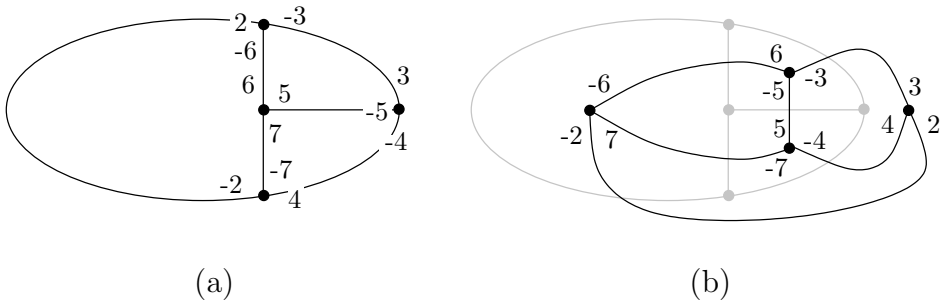


Fig. 2. The map $\langle \varphi, \alpha \rangle$ of Fig. b is obtained by defining a vertex for each face of the map $\langle \sigma, \alpha \rangle$ of Fig. a and by crossing each edge of the map $\langle \sigma, \alpha \rangle$ by an edge defined by the same pair of darts. The resulting graph is a representation of the map $\langle \varphi, \alpha \rangle$ according to an inverse orientation (or of the map $\langle \varphi^{-1}, \alpha \rangle$).

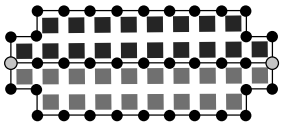
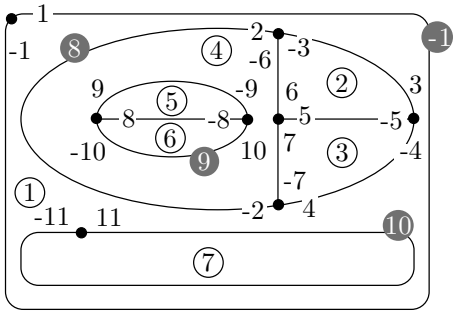


Fig. 3. Example of interpixel boundary defined in the half-integer plane.



d	$\lambda(d)$	d	$\lambda(d)$
-11	10	1	1
-10	9	2	8
-9	9	3	8
-8	5	4	8
-7	3	5	3
-6	4	6	2
-5	2	7	4
-4	3	8	6
-3	2	9	5
-2	4	10	6
-1	-1	11	7

f	$\beta(f)$
-1	-1
1	1
2	-3
3	5
4	7
5	-8
6	8
7	11
8	4
9	-9
10	-11

Fig. 4. A face labeling of the running example. The label in gray are labels of infinite faces.

f	***parent***(f)
8	1
9	4
10	1

f	***children***(f)
1	$\{8, 10\}$
4	$\{9\}$

Fig. 5. The inclusion relation of the running example.

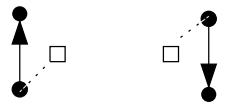
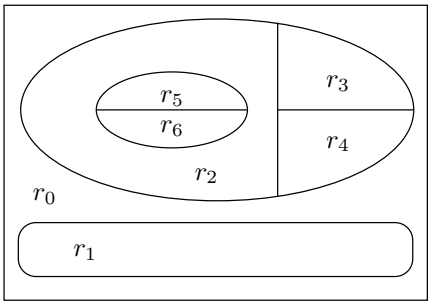


Fig. 6. Reconstruction of the domain of a region. The dark disks represent boundary points and the white squares image points.



	r_1	r_2	r_3	r_4	r_5	r_6	r_7
r_1		I	I	I			I
r_2	O		D	D			
r_3	O	D		D			
r_4	O	D	D		I	I	
r_5				O		D	
r_6				O	D		
r_7	O						

Fig. 7. Example of region neighbouring relations. The letters D, I and O denote respectively the direct, inner and outer neighbouring. For instance the second line of the array means that region r_1 has one outer neighbour (the region r_0), two direct neighbours (the regions r_2 and r_3), and two inner neighbours (the regions r_4 and r_5).