Incremental Modifications of segmented image defined by discrete maps.

Luc Brun and Jean-Philippe Domenger

LaBRI, Université Bordeaux I. 351, cours de la Libération. 33405, Talence Cedex — France.[1]

Abstract

A model of representation of region of segmented image has been defined in [3]. This model allows an efficient implementation of region segmentation processings using *discrete maps*. In this paper, we present the two basic primitives modifying discrete maps: insertion of a new segment and suppression of an existing segment. These both primitives may be used to design more powerfull split and merge algorithms.

The region segmentation processings subdivide an image into connected set of pixels, which are called the *regions* of an image. The basic idea which is a idea common of region segmentation methods is to modify the partition until all image regions satisfy a specific homogeneity criterion. Among region segmentation methods, three main different approaches may be distinguished:

(i) The *top-down* approach [21,19,8] starts with large regions and divide the image into smaller and smaller regions.
(ii) The *bottom-up* approach [5,14] starts with small regions and merge them until the homogeneity criterion is satisfied.
(iii) The *mixed* approach [6,22,9] introduced by Pavlidis and Horowitz [15,16], modifies the partition by splitting or merging regions until they satisfy some homogeneity criterion. Horowitz and Pavlidis have introduced a distinction between the merging step and the grouping step. The merging step beeing a merge step restricted by the quadtree structure while the grouping step is an unrestricted merge step.

The most common structures used by top-down approaches are *pyramids* [6,25] and *quadtree* [24,12,23]. Although the quadtree is simple and efficient for top-down processing, its main drawback to implement merge processings is the time over-head needed to compute complex boundaries or to compute the set of neighbor regions of a given region. This drawback may be reduced using *roped quadtrees* [17], allow an efficient computation of the adjacent blocks of a given block. Nevertheless the problem of boundary computation remains.

The bottom-up methods begins with atomic regions of one or two pixels which grow untill a partition of the image is performed (see for example [27]). This initial partition may be

[1] Laboratoire Bordelais de Recherche en Informatique, Unité de Recherche 1304 Associée au Centre National de la Recherche Scientifique. This work is granted by the Région Aquitaine and the C.N.R.S.

represented by an array of labels [13,20]. Starting from this partition some adjacent regions with close features can be merged. The usual data structure used to effectively perform the merge operations is a *RAG* (Region Adjacency Graph). Each node of this graph represents one image region and there exists one edge between two regions if they are adjacent. The parameters used to compute the homogeneity criterion (mean colors, histogram, ...) are stored on nodes and updated during merge operations. The RAG is ill-adapted to split operations because the split region must be removed from the RAG and all the new sub-regions must be traversed in order to insert their corresponding node and their adjacency edges.

Considering a mixed approach, the data structure frequently used is a quadtree, in a first stage the merge operations are restricted to regions sharing a same parent in order to preserve the tree structure. In a second stage the quadtree structure is broken and reorganized in a RAG and unrestricted merge operations are performed. The main drawback of this processing is due to the break of the quadtree structure involved when two regions with different parents are merged. The break of the quadtree forbids further refinements of regions previously merged.

A mixed model [3], called *discrete maps*, combines a discrete geometry with the Euclidean topology. The regions considered in this work are 4-connected sets of pixels. By embedding the image regions in a discrete plane provided with the digital topology defined by Khalimsky [18], the set of region boundaries is the drawing of a topological graph [26,10,11]. The edges of this graph are the common boundary parts of two different image regions. Unlike quadtree structure which partitions each region into set of squares, an important advantage of this model is that allows segmentation algorithms to handle partitions of the image with free geometry regions.

This model is well-adapted to region segmentation methods using split and merge [7] algorithm, all usual operations may be defined in terms of graph modifications. As a matter of fact, split and merge algorithms work on the same data structure, providing to unrestricted merging operation including what it is sometime called grouping [15,16]. The region splitting can be summed up to the insertion of new segment and the region merging to the suppression of existing segments. Using these both basic primitives: segment *insertion* or *suppression*), the split and merge of regions can be performed by a segmentation algorithm or decide by user. As a matter of fact, too restricted data structure, like quadtree forbid some merge which would break the tree structure. Such restrictions can be handle by a segmentation algorithm but are unbearable for an interactive segmentation. Inserting editing steps between segmentations steps, allows us to reduce the number of generated contours. As a matter of fact, if an user desire fine details of a specific object in the image, he will have to perform an over segmentation of the whole image in order to get all the details of the wanted object. This over segmentation has to be followed by a burden editing step in order to remove all contours which do not perform a partition of the desired object. Discrete maps allows users to interactively select the regions to refine. Thus the algorithm will refine only the regions which form a partition of the desired objects. We thus avoid further segmentation of regions which do not interest the user.

In a previous work, Braquelaire and Guitton [4] have presented the updating of an image by insertion of segments. The images considered in this work have only simply connected regions and the region boundary was defined in the image plane. In this paper, we generalize the insertion to any image (eg. an region may be holed), and we present also the suppression of segments. This paper is organized as follows. Section 2 recalls some basic definitions and concepts of discrete maps. Section 3 defines the allowed data structure used to represent an image. In this section, we present also the extension and the splitting a segment. Section 4 describes the modifications of discrete maps resulting from the addition of a new segment. Section 5 is devoted to the suppression of an existing segment of discrete maps.

1 Discrete maps

In this section we briefly recall the basic definitions of the discrete maps [3]. The model of discrete maps is made of two levels:

(i) The *Geometrical level*: encodes explicitly the geometry of the region-boundary set.
(ii) The *Topological level*: encodes the topology of the subdivision by *topological maps*.

In order to obtain a rigourous topological framework, the boundaries of regions are defined in the *boundary plane* generated by translating $\mathbb{Z}^2$ by the vector $(-\frac{1}{2}, -\frac{1}{2})$. A **half-neighboring** relation is defined between the image points and the points of the boundary plane. Two points are half-neighbors if their Euclidean distance is equal to $\frac{\sqrt{2}}{2}$. Two points of the boundary plane are linked if theirs common half-neighbors belongs to different regions. The **boundary** of a region r is the set of boundary-plane points having one of theirs half-neighbors in r. It can be verified that the boundary of a region is made of closed Jordan curves. Moreover, two adjacent regions have a common boundary part. This both properties allows the structuration of the set of region boundaries by discrete maps. **Discrete maps** are the geometrical embedding of topological maps in the boundary plane. The structuration in discrete maps decomposes the boundary sets into **nodes** which are boundary points and **segment** which are the paths of linked boundary points which connect two nodes. An **augmented segment** $\overline{\boldsymbol{s}}$ is a segment $\boldsymbol{s}$ augmented with its both extremity nodes. The boundary of a region is an alternate sequence of nodes and segments. A region is implicitly defined by its boundary traversed with respect to an orientation convention. The region boundary is oriented such that the region points are to the right during an oriented traverse of the boundary. Therefore, the boundary of a simply connected regions r is a clockwise oriented curve. The boundary of region with n holes is made of $n+1$ curves, one curve surrounds the other curves. The surrounding curve is clockwise oriented, the other ones are counterclockwise oriented. The orientation of a curve can be characterized simply by counting the left and the right turns along the contour [2]. We call this amount the **orientation index** of the contour. The geometry of the region boundaries are explicitly represented by a array $\boldsymbol{BI}$, such that $\boldsymbol{BI}[p]$ encodes the boundary points linked with p. In this way, a segment may be traversed starting from one of its extremity node, knowing its first point (see [1]).

The underlying topological map of a discrete map is obviously defined by the correspondences between nodes and vertices, segments and edges, regions and faces. A face defined by a clockwise oriented curve is called a *finite face*, the face defined by a counterclockwise oriented curve is called an infinite face. The infinite face of a given topological map m is the complement of the union of the finite faces of m.

The topological maps are classically encoded by combinatorial maps. A combinatorial maps is a tuple of permutations (σ, α) defined on a set of labels called **darts**. The darts of combinatorial maps are associated with half-edges of the topological maps. The permutation α is a free-fixed point involution, and its orbits encode the edges of the topological maps. The orbits of the permutation σ encode the vertices of the topological maps. The faces are encoded by the orbits of the permutation $\varphi = \sigma \circ \alpha$ (also denoted $\sigma\alpha$). The orientation convention chosen for region boundaries induces to order counterclockwise around node the darts of σ orbits. The set of darts is denoted $\boldsymbol{\mathcal{D}}$. Usually the darts are presented by positive and negative integers. Hence, the permutation α my be implicitly encoded by $\alpha(d) = -d$. Let π a permutation and d be a dart, we denote by $\pi^*(d)$ the orbit of the permutation π generated from the dart d.

In order to identify efficiently the darts of a face, a labeling function λ is defined on $\boldsymbol{\mathcal{D}}$, such that:

$$\forall d, d' \in \boldsymbol{\mathcal{D}}, \lambda(d) = \lambda(d') \Longleftrightarrow \varphi^*(d) = \varphi^*(d')$$

In order to retrieve the orbit of a face, each face label is associated with a canonical dart by a function denoted β such that $\lambda(\beta(f)) = f$.

The geometrical inclusion between a region and its holes is expressed on combinatorial maps by an **inclusion relation** between one finite face and its included infinite faces. The finite face is called the **parent face** and its included infinite faces are its **child faces**. The inclusion relation is encoded by two functions $\boldsymbol{parent}$ and $\boldsymbol{child}$ encoding respectively the parent face of an infinite face and the list of child faces of a finite face. The computation of the parent face of an infinite face l_∞, can be performed starting from the rightmost point p_{right} of the infinite face. It is sufficient to scan the image from p_{right} to the right until a boundary point b is reached. Using $\boldsymbol{BI}$, we identify the dart d associated with the oriented augmented segment s containing b such that the point p_{right} is located to the right of s according to its orientation. The parent face of l_∞ is identified by $\lambda(d)$ if it is a finite face and by $\boldsymbol{parent}(\lambda(d))$ if not.

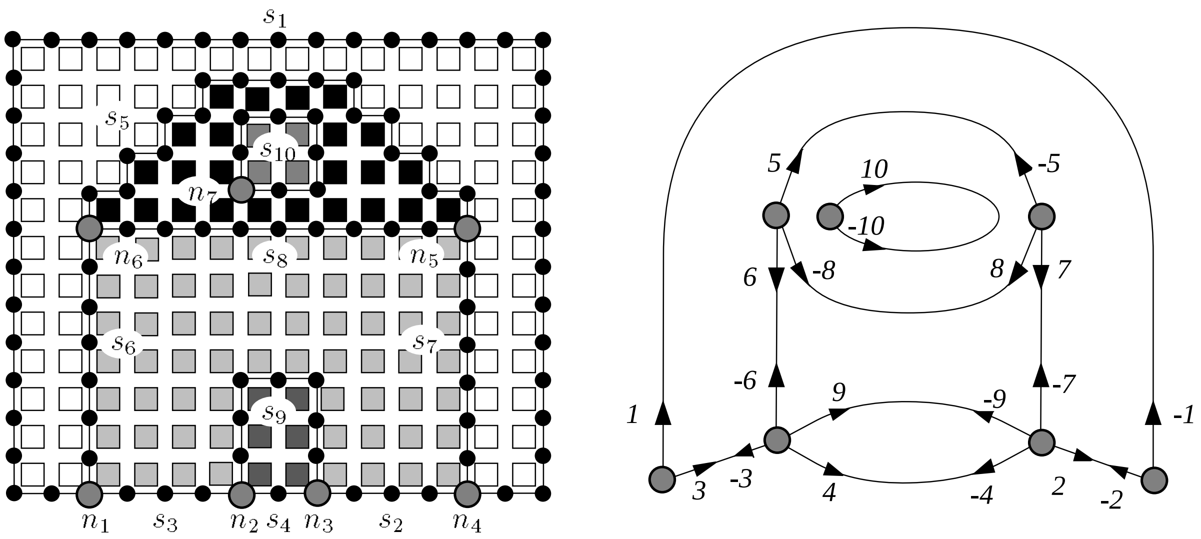
Since the darts are half-edges, they are associated with an augmented oriented segment. We denote by $\overline{\boldsymbol{s}}_d$ the oriented segment associated with the dart d and $\overline{\boldsymbol{s}}_{\alpha^*(d)}$ the segment associated with the edge $\alpha^*(d)$. Let r be the region associated with the finite face labeled by l, the boundary of r is the set of curves C_i such that:

$$\begin{array}{rclcrcl} C_0 & = & \overline{\boldsymbol{s}}_d \oplus \ldots \overline{\boldsymbol{s}}_{\varphi^n(d)} & \text{with} & d & = & \beta(l) \\ C_k & = & \overline{\boldsymbol{s}}_{d_k} \oplus \ldots \overline{\boldsymbol{s}}_{\varphi^{n_k}(d_k)} & \text{with} & d_k & = & \beta(l_k), \forall l_k \in \boldsymbol{child}(l) \end{array}$$

where the symbol $\oplus$ represent the concatenation of segments. Using region boundary, the reconstruction of the region can be performed by scan-line algorithms or seed-fill algorithm (see [1]).

An illustration of the model is presented in figure **??**.

The two levels of this model provide a data structure allowing an efficient implementation of region segmentation algorithms. For instance, if we consider the dual combinatorial map defined by (φ, α), the vertices of these maps are the orbits of φ which are associated with the regions of the partition. Hence, the RAG can be generated from these maps by suppressing all the multiple edges between two nodes and creating edges induced by the inclusion relation.



a) a segmented image and its boundary points b) the two combinatorial maps

Fig. 1. The part (a) of the figure shows an segmented image with five regions. The boundary points are represented by disks where the grey disks are the nodes. The links are short straight lines joining boundary points. The part (b) of the figure shows the two combinatorial maps associated with the boundary set. Each edge is defined by a tuple $(d, \alpha(d) = -d)$. For instance, the segment s_6 is associated with $\alpha^*(6) = (6\ -6)$ and the node n_5 is associated with $\sigma^*(-5) = (-5\ 8\ 7)$. They are two infinite faces respectively defined by $\varphi^*(-1)$ and $\varphi^*(-10)$. The region representing the roof is holed. Thus, the finite face $\varphi(5)$ is the parent face of the infinite face $\varphi^*(-10)$.

Therefore, this model represents a more powerful data structure than a RAG, because it encodes the adjacency of regions and identifies the segments inducing the adjacency.

In the next sections we will focus on the incremental modifications of a partition. Note that if the initial partition is resumed to a rectangle enclosing the whole image the modifications will rather defined a partition than modified it. The two basic primitives used to modify a partition are the insertion and the suppression of a segment. The modifications of the Geometrical level induced by the insertion or the suppression of a segment being quite simple we will mainly focus on the updates of the Topological level.

2 Insertion and suppression of nodes

We saw in previous section that our model defines implicitly each region of the segmented image by its boundary. The boudary of a region being obtained by the concatenation of the segments which form the boudary of the region. If we add a node n on a segment s we cut the segment s into two sub segments s' and s''. This operation do not modified the boundary of the region associated to s. Thus, the cut of the segment s has augmented the data structure and has not modified the partition of the segmented image. In this case we say that the set of nodes of discrete maps is not *minimal*. If we define the **rank** of a boundary point as the number of boundary point linked with it (see.[3, definition 3]). The set of nodes of discrete maps is said minimal if for each node n of rank two there is no other notes connected to n (see.[3, definition 6]). In this case, segments of discrete maps are the maximal 4-connected paths included in the intersection of the boundaries of two different regions. Nevertheless, the minimality of the data structure is too restrictive, because some algorithms may temporary require non minimal data structure. For instance, the insertion of new segment $\boldsymbol{s}$ which extremities are located to the points p and p' begins by the node creation at points p and p'. Before the effective insertion of $\boldsymbol{s}$, the data structure is not minimal because the nodes p and p' have not necessary rank greater than two. Instead of working on minimal data structure, the operations described in the following only preserve the consistency of the data structure(see definition 1).

Definition 1 *The data structure is* ***consistent*** *if the permutation σ is a free-fixed point permutation (e.g $\forall d, \sigma(d) \neq d$).*

The consistency of the data structure forbids edges $(d, \alpha(d))$ such that d or $\alpha(d)$ are fixed points of σ. These edges are called *dangling edge*. Dangling edges typically appear with segmentation methods based on edge detection. In region segmentation algorithms, such edges does not define regions and their treatments will complicate unnecessary algorithms. Another kind of edges allowed by the data structure does not define region. These edges are called *connecting edges*, they connect either two infinite faces that have the same parent face or a parent face with one of its child faces. Let $e = \alpha^*(d)$ be an edge, then e is a connecting edge if $\lambda(d) = \lambda(\alpha(d))$. The connecting edge are necessary to split an holed region. In figure 2, the two cases of connecting edge are displayed. The connecting edges e_c are first inserted in the data structure. After the insertion of the edges e_f, a new region is created and e_c becomes unconnecting edge. In order to split an holed region, it is necessary to insert temporary connecting edges in data structure.

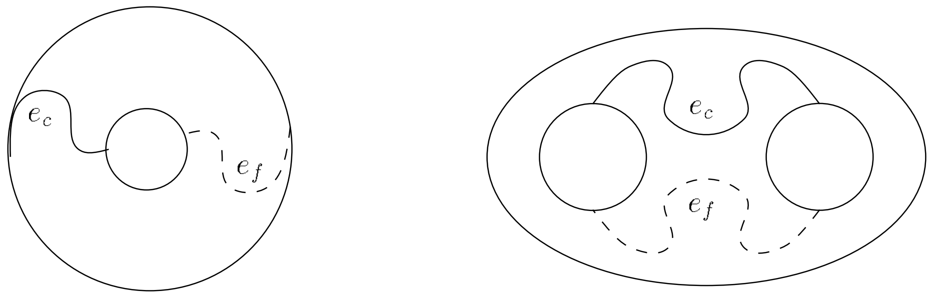


Fig. 2. The insertion of a connecting edge e_c, followed by the insertion of an edge e_f allows the splitting of a holed region.

In order to avoid dangling edge, the insertion of a new segment in the data structure requires that its both extremities are already nodes of the data structure. In order to validate this condition, a preliminary stage consisting in the creation of node has to be performed.

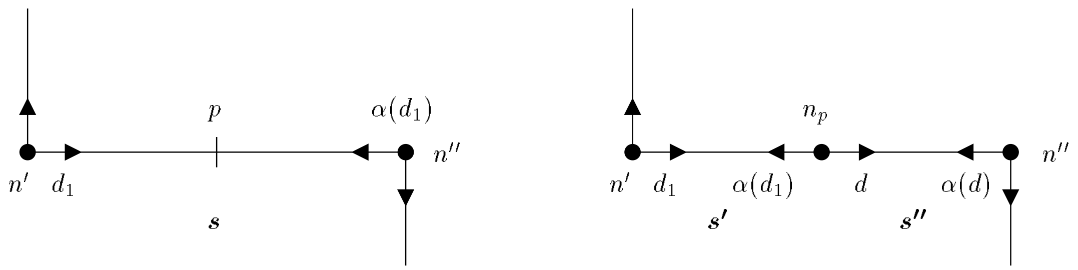


Fig. 3. This Figure shows the creation of node n_p at a point p. The creation split the segment $\boldsymbol{s}$ into $\boldsymbol{s'}$ and $\boldsymbol{s''}$. The associated edge of $\boldsymbol{s}$ is used by $\boldsymbol{s'}$ and the new edge $\alpha^*(d)$ is associated with $\boldsymbol{s''}$.

The creation of a new node at a point p belonging to a segment $\boldsymbol{s}$ split the segment $\boldsymbol{s}$ into two segments $\boldsymbol{s'}$ and $\boldsymbol{s''}$. If the segment $\boldsymbol{s}$ is defined by the edge $\alpha^*(d_1)$, in order to decrease the modifications of the data structures we affect to $\boldsymbol{s'}$ the edge $\alpha^*(d_1)$. In this way, only a new edge $\alpha^*(d)$ corresponding to $\boldsymbol{s''}$ has to be inserted in the data structure. This splitting can be interpreted as

$$\overline{\boldsymbol{s}}_{\alpha^*(d_1)} = \overline{\boldsymbol{s'}}_{\alpha^*(d_1)} \oplus \overline{\boldsymbol{s'}}_{\alpha^*(d)}$$

The node creation is decomposed as followed. We first mark p as a node in $\boldsymbol{BI}$ and we add node n_p into the data structure. Then we transform the permutation σ and the labeling function λ defined on $\boldsymbol{\mathcal{D}}$ into σ' and λ' defined on $\boldsymbol{\mathcal{D}} \cup \{d, \alpha(d)\}$ by:

$$\begin{array}{llll} \sigma'(\alpha(d_1)) & = d & \lambda'(\alpha(d)) & = \lambda(\alpha(d_1)) \\ \sigma'(d) & = \alpha(d_1) & \lambda'(d) & = \lambda(d_1) \\ \sigma'(\alpha(d)) & = \sigma(\alpha(d_1)) & \sigma'(\sigma^{-1}(\alpha(d_1))) & = \alpha(d) \end{array}$$

Note that the splitting of the segment $\boldsymbol{s}$ induces no modification of the subdivision, only topological maps are modified. The node n_p allowing the splitting of s is called an *artificial node* (see definition 2). Artificial nodes naturally appear after the suppression of a segment or before the insertion of a new segment. The artificial nodes created before the insertion of a segment becomes unartificial after the effective insertion of the segment. On the contrary, the artificial nodes created by the suppression of a segment must be explicitly removed.

Definition 2 *A node n of the data structure is* ***artificial*** *if its rank is lower or equal to two and if its σ cycle is different from $(d, \alpha(d))$.*

We have seen that connecting edges or artificial nodes have no interest for the representation of the subdivision. Thus, several data structures may represent a same partition but all of its structures may be reduced on the same minimal data structure by suppression of connecting edges and artificial nodes. The suppression of a connecting edge will be treated in the section 4. Now, we present two operations used to reduce the data structure: *node suppression* and *segment extension*. Let n be an artificial node defined by $\sigma^*(d)$, this node can be suppressed of the data structure without modification of the partition. As opposite operation to node insertion, the suppression of n involves the concatenation of the segments $\overline{\boldsymbol{s}}_{\alpha^*(\sigma(d))}$ and $\overline{\boldsymbol{s}}_{\alpha^*(d)}$, the segment resulting of this operation is denoted by $\overline{\boldsymbol{s'}}_{\alpha^*(d')} = \overline{\boldsymbol{s}}_{\alpha^*(\sigma(d))} \oplus \overline{\boldsymbol{s}}_{\alpha^*(d)}$. The modifications of the data structure are less, if the new segment $\overline{\boldsymbol{s'}}_{\alpha^*(d')}$ is identified by an already existing edge (e.g either $\alpha^*(d)$ or $\alpha^*(\sigma(d))$), the other edge being simply released of the data structure. Suppose that the suppression of the node n keeps the edge $(d, \alpha(d))$, we denote $d_s = \alpha\sigma(d)$ then the data structure is updated as follows:

$$\begin{array}{llll} \sigma'(\sigma^{-1}(d_s)) & = d & \beta'(\lambda(d_s)) & = d \\ \sigma'(d) & = \sigma(d_s) & \beta'(\alpha(d_s)) & = \alpha(d) \end{array}$$

The **extension** of a segment consist to extend its geometry by suppressing its extremity nodes which are artificial. The node suppression is iterated until its both extremities become unartificial. The extension of an augmented segment $\overline{\boldsymbol{s}}_{\alpha^*(d)}$ transforms this segment in the segment $\overline{\boldsymbol{s'}}_d$ such that

$$\overline{\boldsymbol{s}'}_{\alpha^*(d)} = \overline{\boldsymbol{s}}_{\alpha^*(d_{-n})} \oplus \ldots \oplus \overline{\boldsymbol{s}}_{\alpha^*(d)} \oplus \ldots \oplus \overline{\boldsymbol{s}}_{\alpha^*(d_m)} \;\; with \begin{cases} d_i & = \varphi^i(d), \qquad\quad 0 < i \leq m \\ d_{-i} & = \varphi^i(\alpha(d)), \; -n \leq -i < 0 \\ \mid \sigma^*(d_i) \mid = 2, & \qquad\qquad -m \leq i \leq n \end{cases}$$

The extension of a segment can be performed more efficiently than iterating node suppression, by updating directly the nodes $\sigma^*(\alpha(d_n))$ and $\sigma^*(\alpha(d_{-m}))$ as follows:

$$\begin{array}{llll} \sigma'(\sigma^{-1}(\alpha(d_{-n}))) & = \alpha(d) & \sigma'(\sigma^{-1}(\alpha(d_{-m})) & = d \\ \sigma'(\alpha(d)) & = \sigma(\alpha(d_{-n})) & \sigma'(d) & = \sigma(\alpha(d_m)) \end{array}$$

The artificial nodes are simply removed and their darts released. The extension of a segment preserves the consistency of the data structure.

3 Insertion of segments

In this section, we decribe the insertion of a new segment in the data structure. The splitting of a region into sub-regions can be expressed as the insertion of a set of new segments in the data structure. We suppose that inserted segments intersect existing boundaries in at most two points. Thus inserted segments can intersect existing boundaries in exactly two points or be closed. In this last case they intersect existing boundaries in zero or one point.

Let's consider first, the insertion of a closed segment $\boldsymbol{s}$ in the data structure. For instance, the insertion of a closed segment may be used to restrict a refinement step of the segmentation to the domain defined by $\boldsymbol{s}$. During the drawing of s in $\boldsymbol{BI}$, its rightmost point $right_s$ and the orientation index i_s are computed. A new edge $\alpha^*(d)$ is allocated. This edge corresponds to the segment $\boldsymbol{s}$. Two cases may be distinguished in the insertion of $\boldsymbol{s}$: $\boldsymbol{s}$ intersects no existing segments or $\boldsymbol{s}$ intersects existing segments in only one boundary point.

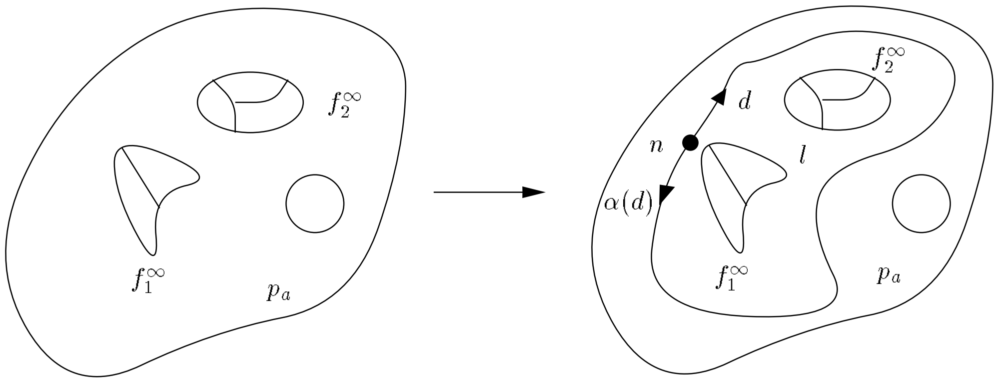


Fig. 4. *The new closed segment* $\boldsymbol{s}$ *intersects no boundary point. The insertion of* $\boldsymbol{s}$ *creates a new map in the data structure*

The figure 4 presents the insertion of a segment $\boldsymbol{s}$ which intersects no existing boundary points. In this case, the insertion of $\boldsymbol{s}$ add a new component in the boundary set. Thus, the contour $\boldsymbol{s}$ traversed clockwise defines a new region labeled by l and traversed counterclockwise a new infinite face labeled by l_∞. The data structure is modified such that:

$$
\begin{array}{llll}
\sigma'(d) & = \alpha(d)\ ; & \sigma'(\alpha(d)) & = d\ ; \\
\lambda'(d) & = l\ ; & \lambda'(\alpha(d)) & = l_\infty\ ; \\
\beta'(l) & = d\ ; & \beta'(l_\infty) & = \alpha(d)\ ;
\end{array}
$$

Arbitrary a point n belonging to $\boldsymbol{s}$ is selected, a node $\sigma'^*(d) = (d, \sigma(d) = \alpha(d))$ located at n is inserted in the data structure. In order to update the inclusion relation, we search the parent face p_a of l_∞ using their rightmost point as described above (see. section 1). In the same way, we select in $\boldsymbol{child}(p)$ the infinite faces which are now included in the new finite face labeled l. We note $\{f_1^\infty, \ldots, f_n^\infty\}$ the set of these infinite faces. The functions $\boldsymbol{child}$ and $\boldsymbol{parent}$ are then updated in the following way:

$$
\begin{array}{lll}
\boldsymbol{parent}(f_i^\infty) & = & \lambda(d), \forall i \in \{1, \ldots, n\} \\
\boldsymbol{parent}(l_\infty) & = & p_a \\
\boldsymbol{child}(p_a) & = & \boldsymbol{child}(p_a) \cup \{l_\infty\} - \{f_1^\infty, \ldots, f_n^\infty\}
\end{array}
$$

Note that, since a closed segment of length lower than 12 can not surround faces we do not have to check $\boldsymbol{child}(\boldsymbol{parent}(p_a))$ if the length of s is lower than 12. This remark may improve significantly updates of the data structure if we use it in conjunction with segmentation algorithms applied on noisy images. In this case, insertion of many small faces corresponding to noise is greatly speedup.

Considering now the case where $\boldsymbol{s}$ intersects only one existing boundary point b. If necessary, a new node n_b located in b is inserted in the data structure. The insertion of $\boldsymbol{s}$ involves the insertion of a new edge $\alpha^*(d)$ in the data structure. Let (d_1, d_2) the orbit of σ which represents n_b before the insertion of s. Using the geometry of darts, we are able to order the darts $d_1, d_2, d, \alpha(d)$ around n_b according to the orientation convention of σ around node. Suppose that the order is $(d_1, d_2, \alpha(d), d)$, we have to transform σ defined on $\mathcal{D}$ in σ' defined on $\mathcal{D} \cup \{d, \alpha(d)\}$

$$
\sigma'(d) = d_1; \qquad \sigma'(\alpha(d)) = d; \qquad \sigma'(d_2) = \alpha(d);
$$

Under these hypothesis, the node n_b is now defined by $\sigma'^*(d_1) = (d_1, d_2, \alpha(d), d)$. Since $\varphi'(\alpha(d)) = \sigma'(\alpha^2(d) = \sigma'(d) = d_1$, the darts $\alpha(d)$ and d_1 belong to the same face. We denote by l, the new label of the finite face associated $\varphi'^*(d)$. Hence, the labeling function λ defined on $\mathcal{D}$ is transformed in the function λ' defined on $\mathcal{D} \cup \{d, \alpha(d)\}$:

$$
\lambda'(d) = l; \qquad \lambda'(\alpha(d)) = \lambda(d_1);
$$

Updates of function $\boldsymbol{child}$ and $\boldsymbol{parent}$ leads us to decompose the insertion of segment $\boldsymbol{s}$ in three cases. The first case occurs when the infinite face $\lambda(d_1)$ has its geometry modified by the insertion of a new finite face $\lambda(d)$ (see Figure 5).

In this case, we have to scan the set $\boldsymbol{child}(\boldsymbol{parent}(\lambda(d_1)))$ in order to find the set $\{f_1^\infty, \ldots, f_n^\infty\}$ of faces included in $\lambda(d)$. Functions $\boldsymbol{parent}$ and $\boldsymbol{child}$ are then updated as followed :

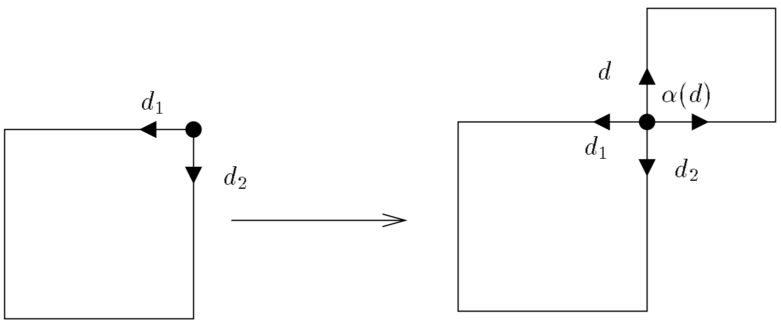


Fig. 5. *The insertion of* $\boldsymbol{s}$ *extends the boundary of the infinite face* $\lambda(d_1)$. *A new finite face is created and the domain of a parent face* $\boldsymbol{parent}(\lambda(d_1))$ *is restricted.*

$$\begin{aligned}
\boldsymbol{parent}(f_i^\infty) &= \lambda(d), \forall i \in \{1, \ldots, n\} \\
\boldsymbol{child}(\lambda(d)) &= \{f_1^\infty, \ldots, f_n^\infty\} \\
\boldsymbol{child}(\lambda(d_1)) &= \boldsymbol{child}(\lambda(d_1)) - \{f_1^\infty, \ldots, f_n^\infty\}
\end{aligned}$$

Note that, $\boldsymbol{child}(\boldsymbol{parent}(\lambda(d_1)))$ has not to be checked if s length is lower than 12. The second case appears when the infinite face $\lambda(d_1)$ becomes a finite face because the inserted segment s surrounds its boundary (see Figure 6).

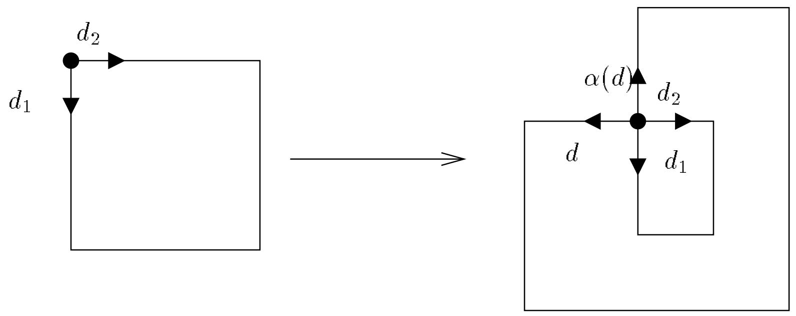


Fig. 6. *The insertion of* $\boldsymbol{s}$ *removes the infinite face* $\lambda(d_1)$. *The domain of* $\boldsymbol{parent}(\lambda(d_1))$ *is restricted by the creation of a new infinite face* $\lambda(d)$

As in the previous case we have to scan the set $\boldsymbol{child}(\boldsymbol{parent}(\lambda(d_1)))$ in order to find the set $\{f_1^\infty, \ldots, f_n^\infty\}$ of faces included in $\lambda(\alpha(d))$. Functions $\boldsymbol{parent}$ and $\boldsymbol{child}$ has to be updated as followed:

$$\begin{aligned}
\boldsymbol{parent}(\lambda(d)) &= \boldsymbol{parent}(\lambda(d_1)) \\
\boldsymbol{parent}(f_i^\infty) &= \lambda(\alpha(d)), \forall i \in \{1, \ldots, n\} \\
\boldsymbol{child}(\boldsymbol{parent}(\lambda(d_1))) &= \boldsymbol{child}(\boldsymbol{parent}(\lambda(d_1))) \cup \{\lambda(d)\} - \{f_1^\infty, \ldots, f_n^\infty, \lambda(d_1)\}
\end{aligned}$$

The set $\boldsymbol{child}(\boldsymbol{parent}(\lambda(d_1)))$ has not to be checked if s has a length lower than 16.

The last case consists to create of a new finite face $\lambda(d)$ and to insert the dart $\alpha(d)$ in the cycle $\varphi^*(d_1)$ (see. Figure 7).

We find the set of faces $\{f_1^\infty, \ldots, f_n^\infty\}$ included in $\lambda(d)$ by checking faces contained in $\boldsymbol{child}(\lambda(d_1))$. Function $\boldsymbol{parent}$ and $\boldsymbol{child}$ are then modified as followed:

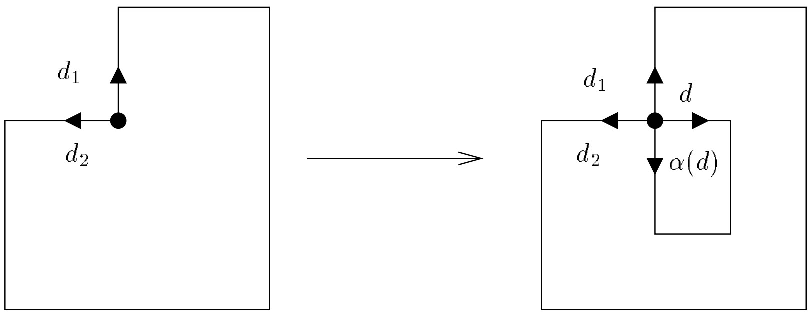


Fig. 7. *The insertion of $\boldsymbol{s}$ restricts the domain of the finite face $\lambda(d_1)$*

$$\begin{aligned}
\boldsymbol{parent}(f_i^\infty) &= \lambda(d), \forall i \in \{1,\dots,n\} \\
\boldsymbol{child}(\lambda(d)) &= \{f_1^\infty,\dots,f_n^\infty\} \\
\boldsymbol{child}(\lambda(d_1)) &= \boldsymbol{child}(\lambda(d_1)) - \{f_1^\infty,\dots,f_n^\infty\}
\end{aligned}$$

If segment s has a length lower then 12 we can ensure that the set $\{f_1^\infty,\dots,f_n^\infty\}$ is empty.

The insertion of closed segment allows segmentation algorithm to insert new faces in the data structure but they do not permit to split existing faces. This splitting is performed by the insertion of non-closed segments in the data structure. We suppose that such segments intersect the set of existing segments in exactly two points corresponding to theirs extremities. If this condition is not verified, segments can be split in several sub-segments which verify this condition. Thus, let us consider a new segment $\boldsymbol{s}$ validating the condition. In order to perform the insertion of $\boldsymbol{s}$, we have to split the segments which intersect $\boldsymbol{s}$ to intersections points b_1 and b_2. We thus create two nodes n_{b_1} and n_{b_2}. The insertion of the new segment $\boldsymbol{s}$ induces the insertion of a new edge denoted $\alpha^*(d)$. Using the geometry, the darts around the nodes n_{b_1} and n_{b_2} must be again ordered according to the orientation convention of σ. We note d_1 and d_2 the darts that follow respectively of d and $\alpha(d)$ turning in a counterclockwise sens around nodes. Thus, the permutation σ is transformed in the permutation σ' defined on $\boldsymbol{\mathcal{D}} \cup \{d, \alpha(d)\}$ with :

$$\begin{aligned}
\sigma'(d) &= d_1; \quad & \sigma'(\alpha(d)) &= d_2; \\
\sigma'(\sigma^{-1}(d_1)) &= d; \quad & \sigma'(\sigma^{-1}(d_2)) &= \alpha(d)\ ;
\end{aligned}$$

After the modifications of σ, the darts d_1 and $\alpha(d)$ (resp. d_2 and d) belong to the same face. Now, we have to update the labeling function λ and the functions $\boldsymbol{parent}$ and $\boldsymbol{child}$. Two main cases should be distinguished depending if the edge $\alpha^*(d)$ is a connecting edge.

If $\alpha^*(d)$ is not a connecting edge, the insertion of $\boldsymbol{s}$ modifies the partition in two ways a new region is created and the domain of region is restricted. The face whose boundary its modified is labelled by l such that $\lambda(d_1) = \lambda(d_2) = l$. If l is a finite face, the restricted region is identified by l, if not the restricted region is identified by $\boldsymbol{parent}(l)$. The insertion of the new edge split the cycle $\varphi^*(d_1)$ into the cycles $\varphi'^*(\alpha(d))$ and $\varphi'^*(d)$.

If the new edge split an finite face then the cycles $\varphi'^*(\alpha(d))$ and $\varphi'^*(d)$ represents finite faces (see Figure 8). We keep the label l and we create a new label l'.

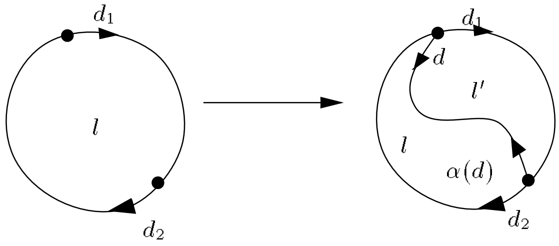


Fig. 8. *The insertion of* $\boldsymbol{s}$ *restricts the domain of the finite face* l *by the creation of the finite face* l'.

Arbitrary, the cycle $\varphi'^*(\alpha(d))$ is now labeled with l'. The array β is transformed in β' by $\beta'(l) = d$ and $\beta'(l') = \alpha(d)$. We scan the set $\boldsymbol{child}(l)$ in order to find the set $\{f_1^\infty, \dots, f_n^\infty\}$ of faces included in l'. Inclusion relation are then updated as :

$$\begin{array}{lcl} \boldsymbol{parent}(f_i^\infty) & = & l', \forall i \in \{1, \dots, n\} \\ \boldsymbol{child}(l') & = & \{f_1^\infty, \dots, f_n^\infty\} \\ \boldsymbol{child}(l) & = & \boldsymbol{child}(l) - \{f_1^\infty, \dots, f_n^\infty\} \end{array}$$

If l is an infinite face, then either the cycle $\varphi'^*(\alpha(d))$ corresponds to a finite face and $\varphi'^*(d)$ to an infinite or the opposite.

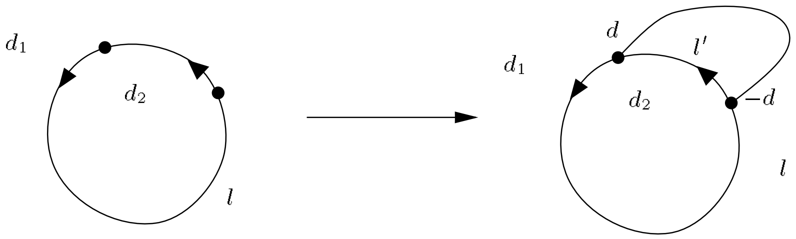


Fig. 9. *The segment* s *extends the boundary of the infinite face* $\lambda(d_1) = \lambda(d_2)$. *The domain of* $\boldsymbol{parent}(\lambda(d_1))$ *is restricted by the creation of a new finite face.*

To determine the finite face, we use the orientation index. Suppose that the orientation index of $\varphi'^*(d)$ is positive, in this case the cycle $\varphi'^*(\alpha(d))$ corresponds to an infinite face and the cycle $\varphi'^*(d)$ to a finite face. In this case, the labeling of $\varphi'^*(\alpha(d))$ is not updated, just the geometry of l is modified. A new finite face labeled by l' is created corresponding to the cycle $\varphi'^*(d)$ and the darts of $\varphi'^*(d)$ are labeled by l' (see. Figure 9). The array β is transformed in β' by $\beta'(l') = d$ and $\beta'(l) = \alpha(d)$. We select in the set of sister faces of l the set $\{f_1^\infty, \dots, f_n^\infty\}$ of faces included in l'. Then functions $\boldsymbol{parent}$ and $\boldsymbol{child}$ are updated as followed :

$$\begin{array}{lcl} \boldsymbol{parent}(f_i^\infty) & = & l', \forall i \in \{1, \dots, n\} \\ \boldsymbol{child}(l') & = & \{f_1^\infty, \dots, f_n^\infty\} \\ \boldsymbol{child}(\boldsymbol{parent}(l)) & = & \boldsymbol{child}(\boldsymbol{parent}(l)) - \{f_1^\infty, \dots, f_n^\infty\} \end{array}$$

If the edge $\alpha^*(d))$ is a connecting edge, its insertion in the data structure does not create region but induces the suppression of an infinite face. Hence, the cycles $\varphi^*(d_1)$ and $\varphi^*(d_2)$ are merged in a new cycle $\varphi'^*(d)$.

In the case, where this edge connects two infinite face with the same parent face (see. Fig-

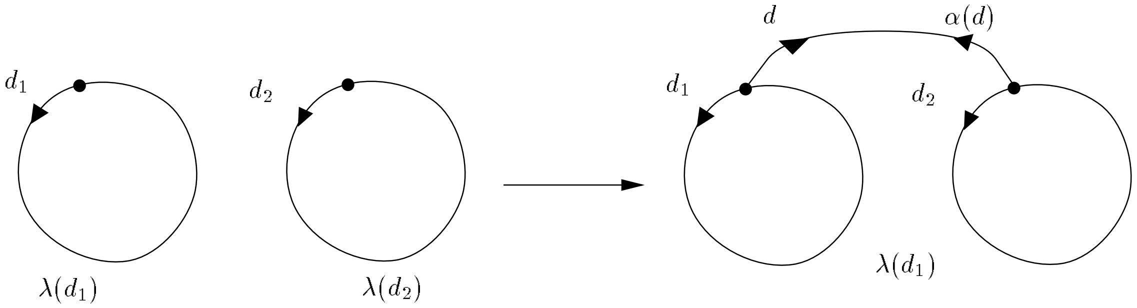


Fig. 10. *The insertion of the edge $\alpha^*(d)$ connects two infinite face having the same parent face.*

ure 10), we choose arbitrary to preserve the face $l = \lambda(d_1)$. Then we label the cycle $\varphi^*(d_2)$ with the label l. The infinite face represented by $\lambda(d_2)$ is removed of the data structure and we update the inclusion relation with $\boldsymbol{child}'(\boldsymbol{parent}(l)) = \boldsymbol{child}(\boldsymbol{parent}(l)) - \{\lambda(d_2)\}$.

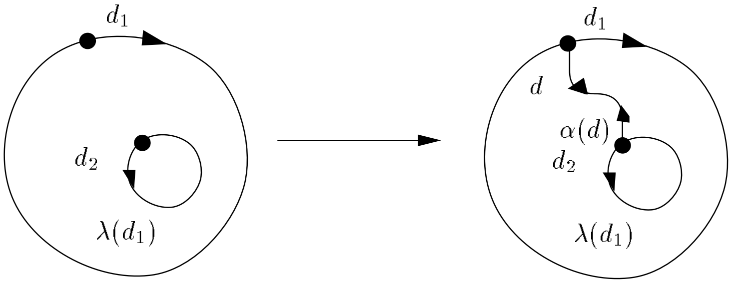


Fig. 11. *The insertion of the edge $\alpha^*(d)$ connects a finite face with one of its child faces.*

In the case, where the new segment connects the infinite face $\lambda(d_2)$ with its parent labeled by $\lambda(d_1)$ (see Figure 11), we label the darts belonging to the cycle $\varphi^*(d_2)$ with $\lambda(d_1)$. The inclusion relation is updated by $\boldsymbol{child}'(\lambda(d_1)) = \boldsymbol{child}(\lambda(d_1)) - \{\lambda(d_2)\}$.

4 Suppression of a segment

Another way to modify the partition of an image, consists to remove a segment of the data structure. The suppression of a segment can be used to merge a set of regions in order to produce a new region or to remove an existing region. The suppression of a segment must preserve the consistency (see definition 1) of the data structure without creating dangling edge.

Let $\boldsymbol{s}_{\boldsymbol{\alpha^*(d)}}$ be the segment whose extemities nodes n and n' are respectivly identified by $\sigma^*(d)$ and $\sigma^*(\alpha(d))$. The suppression of $\boldsymbol{s}_{\boldsymbol{\alpha^*(d)}}$ decreases by one the rank of n and n' if they are different and by two if they are equal. The first case where the suppression implies the inconsistency of the data structure occurs when n and n' are different and one of them is artificial. Suppose that n is artificial, in that case the suppression of the edge $\alpha^*(d)$ creates the dangling edge $\alpha^*(\sigma(d))$. In order to preserve the consistency of the data structure, only the segments with minimal nodes to theirs extremities can be suppressed. Thus, the validation of this condition may require a preliminary step consisting in the extension of the segment $\overline{\boldsymbol{s}}_{\alpha^*(d)}$ into a segment $\overline{\boldsymbol{s}'}_{\alpha^*(d)}$ (see.section 2).

The other case where the data structure becomes inconsistent after the suppression of $\boldsymbol{s}_{\boldsymbol{\alpha^*(d)}}$ occurs when the nodes n and n' denote a same node of rank three. In that case, either

$\sigma^2(d) = \alpha(d)$ or $\sigma^2\alpha(d) = d$. For instance, let us suppose that the node n is defined by the $\sigma^*(d) = (d, \sigma(d), \alpha(d))$ then the suppression of $\boldsymbol{s}_{\alpha^*(d)}$ creates a dangling edge $\alpha^*(\sigma(d))$. To avoid dangling edge, the edge $\alpha^*(\sigma(d))$ must be suppressed of the data structure before the edge $\alpha^*(d)$). These treatments are summed up in the Algorithm SUPPRESS-EDGE defined in figure 12.

```
SUPPRESS_EDGE(α*(d))
{
    Beginning of the preliminary treatment
        if(| σ*(d) |= 2 or | σ*(α(d)) |= 2) then
            extend s̄_d
        if(σ²(d) = α(d) and | σ*(d) |= 3 ) then
            SUPPRESS-EDGE(α*(σ(d)))
        if(σ²α(d) = d and | σ*(d) |= 3 ) then
            SUPPRESS-EDGE(α*(σα(d)))
    End of the preliminary treatment
    effective removing of α*(d)
}
```

Fig. 12. The algorithm SUPPRESS-EDGE remove the edge $\alpha^*(d)$. To avoid dangling edge, preliminary treatment may be performed to preserve the consistency of the data structure.

Proposition 3 *Let $e = (d, \alpha(d))$ be an edge, the algorithm SUPPRESS-EDGE(e) has a recursivity at most one, if e is not a dangling edge.*

Suppose, that $\sigma^2(d) = \alpha(d)$ and $\mid \sigma^*(d) \mid= 3$, the algorithm SUPPRESS-EDGE has a recursivity greater than one, iff $\sigma^2\alpha\sigma(d) = \sigma(d)$ or $\sigma^3(d) = \alpha\sigma(d)$.

$\sigma^2\alpha\sigma(d)$	$=$ $\sigma(d)$	$(\mid \sigma^*(\sigma(d)) \mid= 3)$		$\sigma^2\sigma(d)$	$=$ $\alpha\sigma(d)$	$(\mid \sigma^*(d) \mid= 3)$
$\alpha\sigma(d)$	$=$ $\sigma^2(d)$	$(\sigma^2(d) = \alpha(d))$		d	$=$ $\alpha\sigma(d)$	(α is an involution)
$\alpha\sigma(d)$	$=$ $\alpha(d)$	(α is an involution)		$\alpha(d)$	$=$ $\sigma(d)$	$(\alpha(d) = \sigma^2(d))$
$\sigma(d)$	$=$ d			$\sigma(d)$	$=$ d	

Since e is not a dangling edge and the extension of a segment does not produce dangling edge, we have $\sigma(d) \neq d$. Then, the recursivity of the algorithm SUPPRESS-EDGE is at most one in this case.

To the other case (e.g $\sigma^2(\alpha(d)) = d$), the arguments are the sames and we obtain $\sigma^3\alpha(d) = \alpha\sigma\alpha(d)$ or $\sigma^2\alpha\sigma\alpha(d)$ equivalent to $\sigma(d) = d$. Therefore the algorithm SUPPRESS-EDGE has a recursivity at most one.

The algorithm SUPPRESS-EDGE preserves the consistency of the data structure, because it has a recursivity at most one and the edges effectivly removed have no artificial nodes as extremities . Now, we present the modifications of the data structure induced by the effective suppression of the edge $e = \alpha^*(d)$ when extremity nodes are not artificial.

If we have $n = n'$ then e is a **looping edge**. Two cases may occur. If the rank of n is two, its rank become null and n must be removed. In that case, we have $\sigma(d) = \alpha(d)$ and $\sigma\alpha(d) = d$. Thus it is enough to remove the cycle $(d, \alpha(d))$ by freeing the related entries of the array σ. The node n is also simply released of the data structure. Suppose that $\lambda(d)$ is the finite face and $\lambda(\alpha(d))$ is the infinite one. The inclusion relation is updated as follows:

$$\begin{aligned} \boldsymbol{child}'(\boldsymbol{parent}(\lambda(\alpha(d)))) &= \boldsymbol{child}(\boldsymbol{parent}\lambda(\alpha(d))) \cup \boldsymbol{child}(\lambda(d)) - \{\lambda(\alpha(d))\} \\ \boldsymbol{parent}'(l) &= \boldsymbol{parent}(\lambda(\alpha(d))),\ \forall l \in \boldsymbol{child}(\lambda(d)) \end{aligned}$$

Since the data structure is consistent, the other case appear when the rank of n is four. Let d_1 and d_2 be the two other darts belonging to this node. The edge e being a loop we have either $\varphi(d) = d$ or $\varphi(\alpha(d)) = \alpha(d)$. Suppose that we have $\varphi(d) = d$. Under these hypothesis we have $\sigma(\alpha(d)) = d$ and the cycle of σ describing n is $(d_1, d_2, \alpha(d), d)$. When the edge e is deleted the pair of darts $(d, \alpha(d))$ is removed and the permutation σ is updated as follows:

$$\sigma'(d_1) \quad = d_2;\ \sigma'(d_2) \quad = d_1;$$

Moreover, $\varphi(\alpha(d)) = \sigma(d) = d_1$ then $\lambda(d_1) = \lambda(\alpha(d))$. If $\beta(\lambda(d_1))$ is $\alpha(d)$, we update β in β' such that $\beta'(\lambda(d_1)) = d_1$. Since each map contains only one infinite face, three cases must be considered to update the inclusion relation:

(i) $\lambda(d)$ is an finite face and $\lambda(\alpha(d))$ an infinite face. The label $\lambda(d)$ is released of the data structure.

$$\begin{aligned} \boldsymbol{child}'(\boldsymbol{parent}(\lambda(\alpha(d)))) &= \boldsymbol{child}(\boldsymbol{parent}(\lambda(\alpha(d)))) \cup \boldsymbol{child}(\lambda(d)) \\ \boldsymbol{parent}'(l) &= \boldsymbol{parent}(\lambda(\alpha(d))),\ \forall l \in \boldsymbol{child}(\lambda(d)) \end{aligned}$$

(ii) $\lambda(d)$ is an infinite face and $\lambda(\alpha(d))$ a finite face. The label $\lambda(d)$ is not released of the data structure but the cycle $\varphi'^{*}(d_2)$ is relabeled with $\lambda(d)$.

$$\begin{aligned} \boldsymbol{child}'(\boldsymbol{parent}(\lambda(d))) &= \boldsymbol{child}(\boldsymbol{parent}(\lambda(d))) \cup \boldsymbol{child}(\lambda(\alpha(d))) \cup \{\lambda(\alpha(d))\} \\ \boldsymbol{parent}'(l) &= \boldsymbol{parent}(\lambda(d)),\ \forall l \in \boldsymbol{child}((\boldsymbol{parent}(\lambda(d))) \cup \{\lambda(\alpha(d))\}) \end{aligned}$$

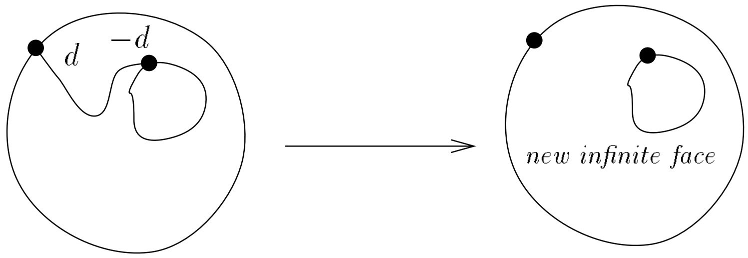
(iii) $\lambda(\alpha(d))$ and $\lambda(d)$ are finite faces. The label $\lambda(d)$ is released from the data structure.

$$\begin{aligned} \boldsymbol{child}'(\lambda(\alpha(d))) &= \boldsymbol{child}(\lambda(\alpha(d))) \cup \boldsymbol{child}(\lambda(d)) \\ \boldsymbol{parent}'(l) &= \lambda(\alpha(d)),\ \forall l \in \boldsymbol{child}(\lambda(d)) \end{aligned}$$

Now, we have to present the modification induced by the suppression of e when it is not a looping edge. First, we remove e from the topological map by updating σ in the following way:

$$\sigma'(\sigma^{-1}(d)) \quad = \sigma(d); \quad \sigma'(\sigma^{-1}(\alpha(d))) \quad = \sigma(\alpha(d))$$

The updating of the inclusion relation can be decomposed into two cases, according to the edge e is or not a connecting edge. If e is a connecting edge, we have $\lambda(d) = \lambda(\alpha(d))$. The deletion of e splits the cycle $\varphi^{*}(d)$ into the two cycles $\varphi'^{*}(\sigma(d))$ and $\varphi'^{*}(\sigma(\alpha(d)))$. Two sub-cases are considered depending on whether the edge e link a infinite face to a finite one or two infinite faces.



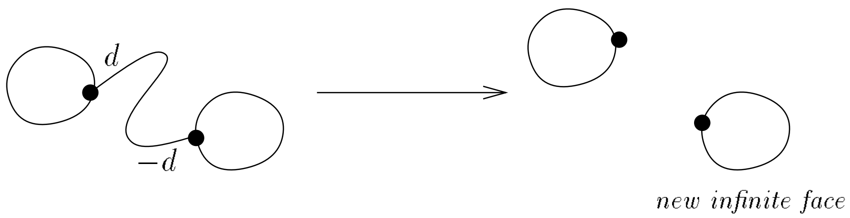
(i) Case 1.a

Fig. 13. *The edge $(d, \alpha(d))$ links a face with its Parent*

If $\lambda(d)$ is a finite face, then e connect a parent face with one its child faces (see. Figure **??**). After the suppression of e, either ${\varphi'}^*(\sigma(d))$ or ${\varphi'}^*(\sigma(\alpha(d)))$ identifie the parent face. The determination of the finite face is performed by the computation of their orientation index (see. **??**). Suppose that the face having a positive orientation index is ${\varphi'}^*(\sigma(d))$. In this case the darts belonging to ${\varphi'}^*(\sigma(\alpha(d)))$ are labeled with a new label l_∞ corresponding to a new infinite face. The data structure is updated as follows:

$$\begin{aligned}
\boldsymbol{child'}(\lambda(d)) &= \boldsymbol{child}(\lambda(d)) \cup \{l\} \\
\boldsymbol{parent'}(l) &= \lambda(d) \\
\beta'(\lambda(d)) &= \sigma(d) \\
\beta'(l_\infty) &= \sigma(\alpha(d))
\end{aligned}$$

(ii) If $\lambda(d)$ is a infinite face, then e links two infinite faces having the same parent (see Figure 14). Both cycles ${\varphi'}^*(\sigma(d))$ and ${\varphi'}^*(\sigma(\alpha(d)))$ define infinite faces. The darts belonging



Case 1.b

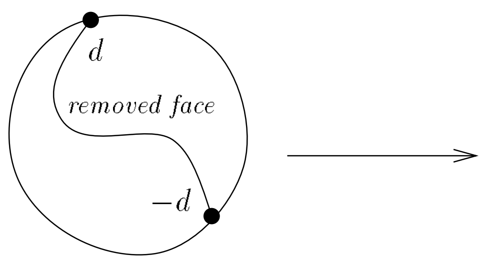
Fig. 14. *In this case the edge $(d, \alpha(d))$ links to sister faces*

to ${\varphi'}^*(\sigma(d))$ stay arbitrary labeled by $\lambda(d)$ and the darts belonging to ${\varphi'}^*(\sigma(\alpha(d)))$ are labeled by a new label l_∞. The data structure is updated as follows:

$$\begin{aligned}
\boldsymbol{child'}(\boldsymbol{parent}(\lambda(d))) &= \boldsymbol{child}(\boldsymbol{parent}(\lambda(d))) \cup \{l\} \\
\boldsymbol{parent'}(l) &= \boldsymbol{parent}(\lambda(d)) \\
{\lambda'}^{-1}(\lambda(d)) &= \sigma(d) \\
{\lambda'}^{-1}(l) &= \sigma(\alpha(d))
\end{aligned}$$

Let'us now consider the case where e is not a connecting edge, the suppression of e induce the suppression of a finite face. Thus, its deletion involves the merging of the cycles $\varphi^*(d)$ and $\varphi^*(\alpha(d))$. Two sub-cases are considered depending on whether the edge e separates two finite faces or a finite and an infinite faces.

(i) If $\lambda(d)$ and $\lambda(\alpha(d))$ both finite faces. The suppression of e extends the geometry of one finite by the addition of the domain of the other one. Suppose for instance that the



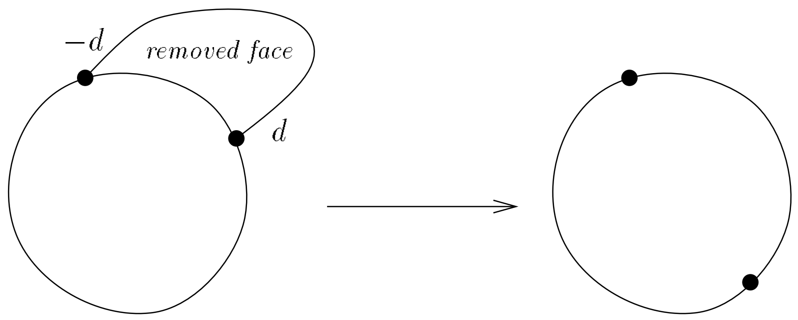
Case 2.a

Fig. 15. *In this case the edge $(d, \alpha(d))$ is shared by two finite faces*

region labeled by $\lambda(d)$ is extended. The other one (in this case $\lambda(\alpha(d))$) is released from the data structure. Therefore, the darts belonging to $\varphi^*(\alpha(d))$ are labeled by $\lambda(d)$ and the face $\lambda(d)$ become the parent face of the child faces of $\lambda(\alpha(d))$. The data structure is updated as follows:

$$\begin{aligned}
\boldsymbol{child}'(\lambda(d)) &= \boldsymbol{child}(\lambda(d)) \cup \boldsymbol{child}(\lambda(\alpha(d))) \\
\boldsymbol{parent}'(l) &= \lambda(d),\ \forall l \in \boldsymbol{child}(\lambda(\alpha(d))) \\
\lambda'^{-1}(\lambda(d)) &= \sigma(d)
\end{aligned}$$

(ii) One of the faces $\lambda(\alpha(d))$ and $\lambda(d)$ is finite and the other is infinite. The function ***parent***



Case 2.b

Fig. 16. *In this case the edge $(d, \alpha(d))$ separes a finite face and an infinite face*

is used to determine the finite face. Suppose that $\lambda(\alpha(d))$ is the finite face. It is removed from the data structure and the darts belonging to the cycle $\varphi^*(\alpha(d))$ are labeled by $\lambda(d)$. We note p the parent face of $\lambda(d)$. The data structure is updated as follows:

$$\begin{aligned}
\boldsymbol{child}'(p) &= \boldsymbol{child}(p) \cup \boldsymbol{child}(\lambda(\alpha(d))) \\
\boldsymbol{parent}'(l) &= p,\ \forall l \in \boldsymbol{child}(\lambda(\alpha(d))) \\
\beta'(\lambda(d)) &= \sigma(d)
\end{aligned}$$

Now we present, an application of the algorithm `SUPRESS-EDGE` to merge a set of regions. The merging of a set of regions $\{rm_1, \ldots, rm_n\}$ is the construction of a new region which domain is the union of regions rm_i. We make the assumption that for all regions rm_i, there exists another region rm_j such that rm_i and rm_j share at least a segment. This hypothesis makes sure that the union of domains is 4-connected. Let $\{r_1, \ldots, r_k, rm_1, \ldots, rm_n\}$ be the set of regions of an image I, the merging of $\{rm_1, \ldots, rm_n\}$ transforms the image I in the image I' whose regions are $\{r_1, \ldots, r_k, r_{k+1}\}$, with $r_{k+1} = \bigcup rm_i$. Let b be a boundary point of I

having all its half-neighbors in r_{k+1}. If b is a point of a segment $\boldsymbol{s}$ then all the other points of $\boldsymbol{s}$ have also all their half-neighbors in r_{k+1}. Such a segment is called a **shared segment** of the merging. On the other hand if b is a node then it is linked to only shared segments. Such a node is called a **shared node**.

The boundary of the region r_{k+1} is the union of the boundaries of regions rm_i without the points having all of their half-neighbors in r_{k+1}. Thus the boundary of r_{k+1} is the union of boundaries of regions rm_i from which have been removed shared nodes and share segments. The merging is initialized by building the list of shared segment. Then share segments are removed one by one from the data structure using the algorithm `SUPPRESS-EDGE` (see. Figure 12).

First let us focus on the building of the list of share segments. Remark that it can be done efficiently by using the topological data structure and searching for *shared edge* which are the edges associated with a shared segment. Since an edge is wholly defined by one of its darts, the list of share edges is encoded by a list of darts. Let l_i be the label of a region rm_i. A share edge $(d, \alpha(d))$ verifies the following conditions:

(i) $\lambda(d) \in \{l_1, \ldots, l_n\}$
(ii) $\lambda(\alpha(d)) \in \{l_1, \ldots, l_n\}$

Therefore, the set of share edges can be computed by traversing the cycle of each of the faces labelled by l_i. We use a clearing mark, to store the darts that has been traversed. For each dart d belonging to the boundary of region rm_i, we consider the dart $\alpha(d)$. In the case where $\alpha(d)$ is marked, the edge $(d, \alpha(d))$ is an share edge then the mark of $\alpha(d)$ is cleared and d is added to the list. In the other case, d is marked. Note that it is also necessary to traverse every child faces of l_i because they belong to the boundary of the region rm_i. The algorithm `SHARED-EDGES` computing the list of share edges is given in Figure 17.

```
input:  a list L of labels of adjacent regions
output: the associated list D of darts of shared edges
{
    let D be the empty list
    for (l ∈ L) do
    {
        for (f ∈ φ*(β(l)) ∪ {∪_{l'∈child(l)} φ*(β(l'))}) do
        {
            if (α(d) is marked)
            {
                insert d in D
                unmark α(d)
            }
            else
                mark d
        }
    }
}
```

Fig. 17. The algorithm `SHARED-EDGES` computes the list of shared edges of a set of adjacent regions.

Remark 4 *The darts which remain marked at the end of the algorithm `SHARED-EDGES` are* *exactly the darts of the boundary of the region r_{k+1}. Thus the remaining marks can easily be removed by traversing the faces of the region resulting from the merging.*

Once the list of shared edges, say E, is computed the updating of the data structure is performed incrementally by removing each edge of E from the data structure using the algorithm `SUPPRESS-EDGE`. In order to preserve the consistency of the data structure, the supression of an edge $\alpha^*(d)$ requires implicit edge suppressions (see. algorithm 12). We can obviously show that edges implicitly suppressed are also shared edges. Indeed, the effective extension of $\alpha^*(d)$ may imply the suppression of the edges defined either $\alpha^*(\varphi^m(d))$ or $\alpha^*(\varphi^n\alpha(d))$ (see section 2). These edges are shared edges because one of their darts is labeled by $\lambda(d)$ and the other by $\lambda(\alpha(d))$, and then they can be suppressed. The other implicit suppressed edges are:

(i) The edge $\alpha^*(\sigma(d))$ with $\sigma^2(d) = \alpha(d)$. This edge is a shared edge because $\lambda(\sigma(d)) = \lambda(\alpha(\sigma(d))) = \lambda(\alpha(d))$.
(ii) The edge $\alpha^*(\sigma(\alpha(d)))$ with $\sigma^2(\alpha(d)) = d$. This edge is a shared edge because $\lambda(\sigma(\alpha(d))) = \lambda(\alpha(\sigma(\alpha(d)))) = \lambda(d)$.

Therefore, the algorithm `SUPPRESS-EDGE`, can be used because the implicit suppressed edges are also shared edges. The last part of the merging consists in the labeling of the region r_{k+1}. When all shared edges are suppressed, there exists only one region labeled by l with $l \in \{l_1, \dots, l_n\}$, this property is induce bythe updating of functions λ and β. The selection of l is performed by the research in the data structure of the label which is always existing.

5 Conclusion

Usual data structure used by segmentation algorithms do not allow combinations of split algorithms and unrestricted merge algorithms. The data structure presented in this paper allows segmentation algorithms to alternate efficiently split steps and unrestricted merges. This last point provides an additional degree of freedom beside data structures like quadtrees. This additional degree of freedom may be used to insert editing steps between split and merge steps. These editing steps may allows users to interactively select the regions which has to be merged or further refined. Moreover our data structure allows us to partition the images into regions with a free geometry. This last point solves the problems linked to the computation of histograms on squares induced by hierarchical data structures like quadtrees.

We have presented in this paper a set of methods which allows the insertion and the suppression of segments. These methods allow us to realize a partition of the original image and to merge adjacent regions. These methods use segments and permutations σ and α defined on planar maps. The update of the data structure after the insertion or the suppression of a segment is thus performed efficiently. The segmentation algorithms are realized by the conjunction of these methods and criterion which allows to decide where segments have to be inserted or which segments need to be removed. An example of split and merge procedure is displayed in Figure 18.

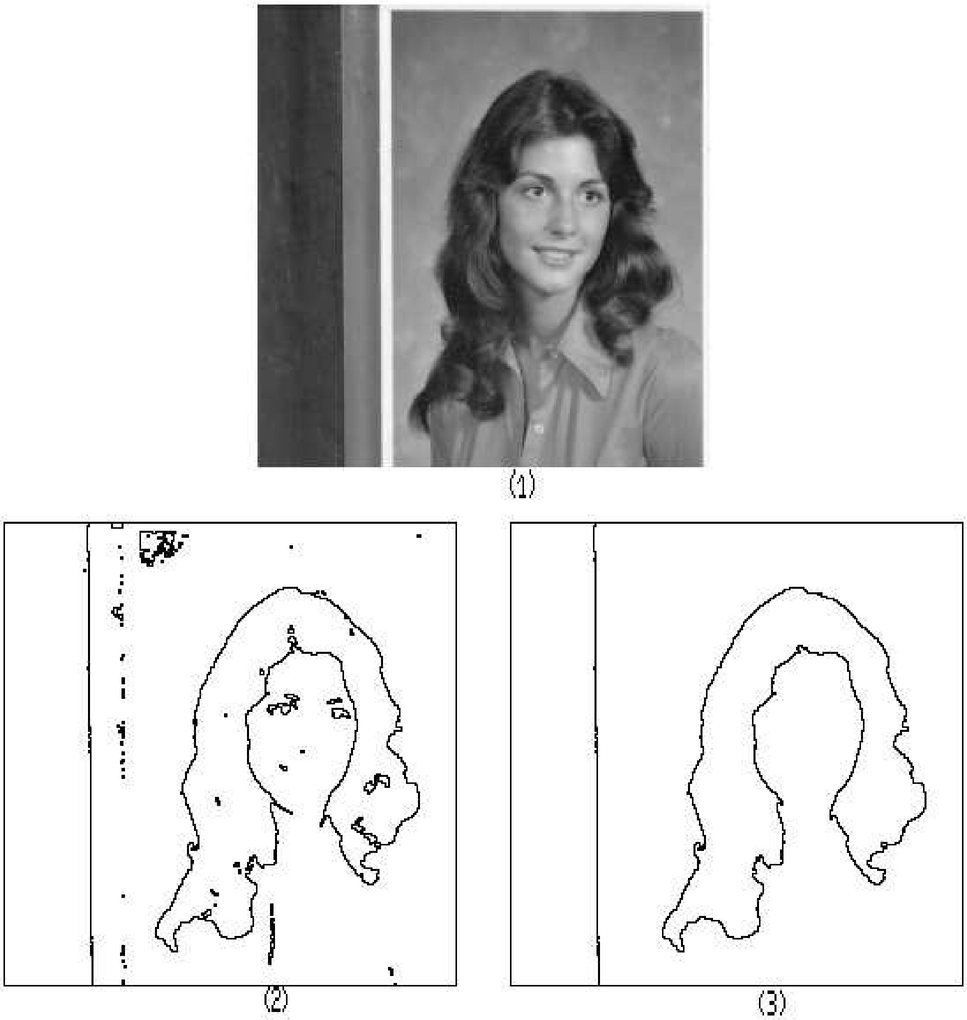


Fig. 18. *Image (1) is the original image. Image (2) is obtained by a split step based on a thresholding of Image(1). Image (3) is deduced from Image (2) by the addition of a merge step.*

References

[1] J.P. Braquelaire and L. Brun. Image segmentation with topological maps and inter-pixel representation. Technical Report 1119-96, Laboratoire Bordelais de Recherche en Informatique (LaBRI), available on http://www/LaBRI/Publications/, may 1996.

[2] J.P Braquelaire and J.P Domenger. Intersection of discrete contours. In *Proc. of Compugraphics'91*, pages 14–23, 1991.

[3] J.P. Braquelaire and J.P. Domenger. Representation of region segmented images with discrete maps. Technical Report 1127-96, Laboratoire Bordelais de Recherche en Informatique (LaBRI), available on http://www/LaBRI/Publications/, June 1996.

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

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

[6] J.D. Browning. Segmentation of pictures into regions with tile-by-tile method. *Pattern Recognition*, 15(1):1–10, 1982.

[7] L. Brun and J. P. Domenger. A new split and merge algorithm based on discrete maps. Technical Report 1128-96, Laboratoire Bordelais de Recherche en Informatique (LaBRI), available on http://www/LaBRI/Publications/, June 1996.

[8] M. Celenk. A color technique for image segmentation. *Computer Vision, Graphics, and Image Processing*, 52:145–170, 1990.

[9] F. Cheevasuvut, H. Maitre, and D. Vidal-Madjar. A robust method for picture segmentation based on a split and merge procedure. *CVGIP*, 34:268–281, 1986.

[10] R. Cori. *Un code pour les graphes planaires et ses applications*. PhD thesis, Université Paris VII, 1975.

[11] J.P. Domenger. *Conception et implémentation du noyau graphique d'un environnement $2D\frac{1}{2}$ d'édition d'images discrètes*. PhD thesis, Labri Université Bordeaux I, 351 cours de la libération 33405 Talence, avril 1992.

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

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

[14] C. Fiorio and J. Gustedt. Two linear time union-find strategies for image processing. Technical Report 375/1994, Technische Universitat Berlin, 1994.

[15] S.L. Horowitz and T. Pavlidis. Picture segmentation by a directed split-and-merge procedure. Technical report, Departement of Electrical Engineering, Princeton University , N. J. 08540, 1975.

[16] S.L. Horowitz and T. Pavlidis. Picture segmentation by a tree traversal algorithm. *Journal of The Association for Computing Machinery*, 23(2):368–388, April 1976.

[17] 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.

[18] 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.

[19] C.-H. Lee. Recursive region splitting at hierarchical scope views. *Computer Vision, Graphics, and Image Processing*, 33:237–258, 1986.

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

[21] R. Ohlander, K. Price, and D.R. Reddy. Picture segmentation using a recursive region splitting method. *Computer Graphics, and Image Processing*, 8:313–333, 1978.

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

[23] A. Rosenfeld and A. Kak. *Digital Picture Processing*, volume 2, chapter 11, pages 191–277. Harcourt and Brace and Jovanovich, 1982.

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

[25] S. Tanimoto and A. Klinger, editors. *Structured computer vision*, volume 1. Academic Press, 1980.

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

[27] L. Vincent and P. Soille. Watersheds in digital spaces:an efficient algorithm based on immersion simulations. *IEEE Trans. on pattern analysis and machine intelligence*, 13:583–597, 1991.