Image Segmentation with Topological Maps and Inter-pixel Representation

Jean-Pierre Braquelaire and Luc Brun

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

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

A new Data Structure for Image Segmentation

Luc Brun

LaBRI, Université Bordeaux I
351, cours de la Libération
33405, Talence Cedex — France

brun@labri.u-bordeaux.fr
Phone Number : 56.84.86.74
Fax Number : 56.84.66.69

Abstract

In this paper we present a data structure improving region segmentation of 2D images. This data structure provides an efficient access to both geometric features such as the set of pixels of a region and topological features like the frontier of a region, the neighbors of a region or the set of regions included in one region. It allows us to combine different segmentation algorithms without restriction. Interactive refinement or merge of regions can also be performed efficiently.

Keywords

Image segmentation, Region inter-pixel boundary, topological map.

List of symbols

Greek Letters

α : alpha,the permutation which associates to each dart d the dart -d.
σ : sigma, the permutation which associate to each dart the next dart by turning conter-clockwize.
ϕ : phi, the permutation which encodes the set of dart of one face.
ϕ^*, α^* : The cycles of permutations ϕ and α.
λ : lambda, a function which associates a label to each dart.
λ^{-1} : a function which associates a dart to each label.
Σ : upper case sigma which denotes the standard deviation of a face
δ : delta, denotes the frontier of a face.
μ : mu, the mean of one face.

Other Symbols

$\mathbb{Z}$: the set of relative integer.
$P_{\frac{1}{2}}$: the inter-pixel plane.
$\mathcal{D}$: The set of darts
$\mathcal{F}$: The set of labels

I. INTRODUCTION

The problem of extracting objects from a complex image has been widely studied for the last fifty years. It quickly appeared that this problem cannot be solved without an priori knowledge of the objects to be recognized. Segmentation algorithms can thus be categorized in two classes: domain-dependent algorithms which attempt to recognize specific objects in a scene – for instance tumors in chest radiographs [2] – and domain-independent algorithms which produce a partition of the original image into a set of regions according to an homogeneity criterion. In that case the recognition of objects from regions is done in latter stages. Among domain-independent methods based on image regions three different approaches can be distinguished:

- The bottom-up approach which leads to merging algorithms [31] consisting in aggregating small regions into larger ones.
- The top-down approach which leads to splitting algorithms [21], [24] consisting in recursively dividing an image into smaller and smaller regions.
- The mixed approach which leads to split and merge algorithms [25], [10], [6], [1], [9] introduced by Horowitz and Pavlidis [16], [17], and combining splitting and merging.

All these methods partition original image by recursively splitting and/or merging its regions. In the following bottom-up, top-down and split and merge methods will be regrouped into the general term of *region-based methods*.

The definition of region-based methods require the knowledge of different features about regions. These features can be grouped into five basic requirements:

1. Traverse a region in order to extract parameters: mean, variance, histogram, etc.
2. Find the set of regions being adjacent to another one.
3. Find the boundary of a region.
4. Find the boundary element shared by two regions or the two regions sharing a same boundary element.
5. Traverse a region boundary in order to extract parameters: length, mean gradient along the boundary, etc.

The calculation of all of these features involves the topology and/or the geometry of the segmented image. There is also a sixth requirement involving a hierarchical organization:

6. Find the split which has produced a given region and the other regions generated by this split.

Other high level features like the set of regions included into another one can be required by region-based algorithms.

Only a subset of these requirements is used by top-down and bottom-up methods. Bottom-up methods mainly use requirements 1 and 2 while top-down methods use requirements 1 and 6. Most of split and merge methods use about all of the six requirements. Data structure used to find efficiently information required by split algorithms are often made of squares organized in *quadtrees* or *pyramids*. These structures are then broken and reorganized into a *Region Adjacency Graph* in order to perform merge operations. These data structures have mainly two disadvantages. First, the initial partition in squares involves extra computations to break the square aspect of segmented images (see [21] for further details). Second, the incompatibility between data structures used for the split and merge operations forbids further refinements of regions previously merged.

In this paper we describe a data structure which allows us to store and process regions of any size or form. This data structure also allows efficient computations of any feature usually required by segmentation algorithms. Moreover this structure can be efficiently updated during split or merge operations. This last property allows us to alternate split and merge operations without restriction. In section II we discuss advantages and disadvantages of the main usual data-structure used by segmentation-algorithms. In section III we present the data structure and some algorithms based on it. In section IV we present some applications of our data structure to segmentation algorithms and in the last section we describe a full example.

II. Usual data structures

One of the first and may be simplest data structure used by segmentation algorithm is the *array of labels* [14], [29], [23]. It consists in associating each pixel with a label such that all the pixels sharing a same label belong to a same region. This data structure is well adapted to operators applied on the whole image such as morphological operators. However this structure is ill-adapted to top-down algorithms since the split of a region involves a relabeling of the pixels of created sub-regions. Moreover regions frontiers are implicitly defined as pixels sides shared by pixels with different labels. Thus, features linked to regions frontier, such as boundary segments or neighboring regions, require burden computation.

The different regions composing an image can also be coded by *median axis transform* (in short MAT) [22], [27], [26]. Each pixel P is associated with the set of upright squares of odd side centered at P. Each square contains uniform labels values. Let us denote by S_P the square of maximal side of this set. The block S_p is a maximal block if there is no point Q such that S_P is contained in S_Q. Each region of an image is then represented by a union of maximal blocks (see Figure 5). For a given region R defined by a set of squares $\{S_1,\dots,S_n\}$ the boundary of R is made of part of squares side $\{b_1,\dots,b_n\}$ such that each b_i belongs to one and only one S_j. Recovery of regions by maximal blocks have to be updated after each split or merge operation. Note that if we do not attempt to preserve regions recovery by maximal blocks, merge operations can be done efficiently by merging squares lists of regions. Thus the MAT data structure allows efficient computations on a fixed partition of the original image and can be used to perform merge operations. However this data structure is ill-adapted to top-down and split and merge methods which have to perform a large number of split.

Arrays of labels and MAT do not provide an explicit representation of the boundary of a region. Thus to process a data along a region boundary it is necessary to perform a reconstruction of this boundary. It is possible to explicitly encode the boundary, for instance by storing for each region the sequence of pixels adjacent to at least a pixel not belonging to the region. Such a boundary can be represented by one of its point and a sequence of moves (see Figure 7).

An important drawback of boundaries defined as sequence of pixels is that they are not topologicaly consistent. Moreover two adjacent regions do not share boundary elements. An alternative to overcome these problems is to consider *inter-pixel boundaries* (see Figure 8). This approach have been first described by Brice and Fennema [5] in 1970 when introducing grouping segmentation algorithm. Later several discrete topologies have been developed which provide formal tools to study such a representation [19], [20], [15]. The boundary of a region is made of closed paths of half-integer points[12] which are the points of the plane $P_{\frac{1}{2}}$ defined by

$$P_{\frac{1}{2}} = \{(i+\tfrac{1}{2}, j+\tfrac{1}{2}), \text{ with } (i,j)\in \mathbb{Z}^2\}$$

It can be shown that a closed single paths of $P_{\frac{1}{2}}$ satisfies Jordan's theorem and that each closed path can be also decomposed into *segments* which are maximal subpaths shared by two regions. Segment extremities are called *nodes* [4]. The geometry of boundaries is encoded by using an array $\boldsymbol{B}$ of size $(M+1)\times(N+1)$ where $M\times N$ is the size of the image. Each entry of the array encodes the neighboring of a boundary point. Thus a segment can be encoded by storing only one of its nodes and an initial move. The other points of the segment are retrieved by reading successive moves in the array $\boldsymbol{B}$. Inter-pixel boundary representation allows an efficient computation of region boundary and of parameters associated with segments. On the other hand this representation only provides and implicit definition of regions and the extraction of data involving topological features may be costly. For instance the computation of the list of regions included into a region r requires to traverse all the pixels of r.

Data structures discussed above can often be combined to produce new structures allowing us to design segmentation algorithms manipulating more sophisticated concepts. In particular, it may be interesting to process an image at different level of resolution. Data structure providing such multi-resolution descriptions are called hierarchical structures. Starting at a coarse level of resolution initial image partition can be refined from level to level until the resolution level of the original image. Using a multiprocessor-architecture different regions can be refined on different processors.

Matrix-pyramids [6], [30] (see Figure 6), or M-pyramids provide such a multi-resolution description. A M-pyramid may be defined as follows: a M-pyramid P is a sequence $\{M(L), M(L-1), \ldots, M(0)\}$ of arrays where M(L) represents the original image, M(L-1) a version of M(L) at half the resolution and so on until M(0) which contains a single pixel.

If segmentation algorithm have to access frequently to various level of resolution, it may be easier to define pyramids in term of trees rather than arrays. Tree-pyramids correspond to the tree version of M-pyramids. Their formal definition can be expressed as followed: a tree pyramid P is a set of p-nodes (k, i, j) with $0 \leq k \leq L$, $0 \leq i \leq 2^L - 1$, and $0 \leq j \leq 2^L - 1$, each p-node corresponding to a pixel of a M-pyramid : the p-node (k, i, j) being the pixel (i, j) of the image $M(k)$ in the M-pyramid $\{M(L), M(L-1), \ldots, M(0)\}$. The father of a p-node (k, i, j) is equal to $(k-1, [i/2], [j/2])$ where $[x]$ denotes the largest integer lower or equal to x.

Each leave of a tree-pyramid represents a pixel of the original image. If we label each leave with its region label all the sons of a node may share the same label. These sons can be represented by their father with no loss of information. Thus tree-pyramid can be pruned, until no node have all its sons with a same label. The Tree-pyramid is then no more balanced and the resulting structure is called a quadtree [26], [28], [13]. Rotations by 90 degrees and boolean operations such that complements, intersections and union can be performed on a quadtree in time proportional to the number of nodes of the input quadtree.

M-pyramids, Tree-pyramids or quadtrees data structures are mainly devoted to top-down region based algorithm. Thus they allow us to efficiently compute children and ancestors of a region defined at any level of recursion. Moreover parameters of a given region can be computed without traversing all pixels of this region. Nonetheless, the computation of boundary elements of a region or computation of its neighbors may involves burden computations. Hunter and Steiglitz [18] have developed an augmented quadtree structure referred to as *roped quadtrees* which allows us to efficiently compute leaves of a quadtree adjacent to a given block.

As said above hierarchical structures are devoted to top-down region based algorithms and do not provide efficient implementations of merge algorithms. Indeed computing adjacency of regions with a tree structure may involves complex processing. Moreover the merge of two regions sharing different fathers breaks the tree structure. Thus split and merge algorithm usually use a hierarchical structure to perform splits and then create a new data structure more adapted to merge operations. Therefore split and merge algorithms are mainly restricted to a split stage followed by a merge stage.

A usual data structure to perform merge operations is the *Region adjacency graph* (RAG). Each node of this graph represents a region of the segmented image and two adjacent regions share an edge in the graph (see Figure 9). The merge of two regions r_1 and r_2 implies to update corresponding nodes in the RAG by contracting the edge linking their nodes. Both nodes r_1 and r_2 are merged and the resulting node is linked to each node of a region adjacent to r_1 or r_2. Multiple edges, edges linking two same nodes are removed (see Figure 10). The Region adjacency graph may be implemented as plexes or adjacency list or half array of bits. When implemented as an half array the merging of two nodes i and j requires to invalidate line and column i and to update line and column j. The region adjacency graph may be used in conjunction with an array of region descriptions. Each entry of this array may contain parameters of one region. These parameters are updated during successive merges and may be used to label the edges of the RAG. For example, this array may contain the mean color of regions. Thus at each step of the merge algorithm, one can merge the two regions r_1 and r_2 such that :

1. The region r_1 and r_2 are adjacent in the RAG,
2. The mean color distance between r_1 and r_2 is minimal.

Parameters such that mean, variance or histogram can be efficiently updated during successive merge operations.

Thus region adjacency graphs give an efficient description of regions adjacency. Moreover adjacency relationships and most of regions parameters can be efficiently updated during successive merge operations. Nonetheless the split of a region r into a set of regions $\{r_1, r_2, \ldots, r_n\}$ involves:

1. To suppress region r in the RAG.
2. To traverse regions $\{r_1, r_2, \ldots, r_n\}$ in order to insert the corresponding nodes and adjacency relation ships in the RAG.

Thus the update of the RAG structure after a split operation involves too many computations. The RAG data structure is thus devoted to merge algorithms.

All structures discussed above are mainly devoted to one kind of operation. Arrays of label and hierarchical structures are devoted to split operations while strings data structure, medial axis transform and region adjacency graph are devoted to merge operations. Moreover each structure is convenient for computing a subset of parameters but may induces complex processing to compute other parameters. For example, computation of the frontier of a given region using a quadtree data structure required many computations. On the other hand the data structure presented in next section may be used for split and merge operations. Moreover this structure allows efficient computations of parameters required by segmentation algorithm.

III. Discrete maps

In this section we briefly recall the model of discrete map and we give some example of basic algorithms that can be design from this model. As sketched above there are two aspects in segmented image representation : the geometrical aspect which describes the shape of regions and the topological one which describes neighborhoods and inclusions of regions. When usual models are either geometrical or topological oriented, discrete maps are based on both a geometric and topologic levels of representation cooperating together.

It appears that the Euclidean topology is much more simple and efficient for encoding the topology of a segmented image than discrete ones. Moreover we have seen that the inter-pixel boundary representation provides a geometrical representation of segmented images which is consistent from a topological point of view. Thus by mixing the topological level with a discrete geometry we get a unified representation model allowing an efficient computation of both geometrical and topological required data.

The topological level is based on planar maps [32]. Using such a structure, the segments between regions are considered like *edges* of a graph. Each edge is decomposed into two opposite half edges called *darts*. The set of darts is denoted by $\boldsymbol{\mathcal{D}}$. Two opposite darts represent the same segment traversed in opposite sense. Thus an edge represented by two darts d and $-d$ is denoted by $(d, -d)$. The intersections of segments defines the vertices of the planar graph. A fundamental property of planar graphs is that such graphs can be represented using two permutations on $\boldsymbol{\mathcal{D}}$ [11] (see Figure 11):

- a permutation α which associates to each dart d it's opposite dart $-d$.
- a permutation σ which associates to each dart its next dart by turning counterclockwise around a node.

The permutations α and σ can be composed together to form the permutation $\varphi = \sigma \circ \alpha$. The cycles of permutation φ encode faces of map (σ, α). For example, in Figure 12 the cycle (-4,5) of the permutation φ encodes the roof of the house. Given a permutation φ and a dart d, we denote by $\varphi^*(d)$ the cycle of φ containing d. Thus, the roof of Figure 12 may be defined as either $\varphi^*(-4)$ or $\varphi^*(5)$.

In order to address faces, we associate a label to each dart of map (φ, α). We thus create a function λ constant on each cycle of φ. Such a function satisfies:

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

The set of labels is denoted by $\boldsymbol{\mathcal{F}}$. In order to access efficiently to the cycles of φ, we also define a function λ^{-1} such that :

$$\forall f \in \boldsymbol{\mathcal{F}}\ d = \lambda^{-1}(f) \Rightarrow \forall d' \in \varphi^*(d)\ \lambda(d') = f$$

The function λ^{-1} associates to each label one dart in the corresponding cycle, the choice of this dart may vary from an implementation to another one. Note that the function λ^{-1} is not the inverse of function λ which is not bijective. More precisely we have the following equations:

$$\forall\ f\ \in \boldsymbol{\mathcal{F}}\ \lambda \circ \lambda^{-1}(f) \quad = \quad f$$

and

$$\forall\ d\ \in \boldsymbol{\mathcal{D}}\ \lambda^{-1} \circ \lambda(d) \quad \in \quad \varphi^*(d)$$

The functions φ, α, λ and λ^{-1} defined above allows us to define a first neighborhood relation.

Definition 1: Given two faces f_1, f_2 we say that f_1 is an external neighbor of f_2 if and only if :

$$\exists d \in \varphi^*(\lambda^{-1}(f_2))\ /\ -d \in \varphi^*(\lambda^{-1}(f_1))$$

An example of external neighborhood is displayed on Figure 15. The set of external neighbors of a face can be stored in a list using Algorithm 1.

```
find_external_neighbors(f,L)
{
    dart d,d_first

    d = d_first = λ^{-1}(f)
    do
    {
        if(λ(-d) is not in L)
            store λ(-d) in L
        d = φ(d)
    }
    while(d != d_first)

}
```

Fig. 1.

If we use a counterclockwise orientation for permutation σ all the cycles of φ excepted one are clockwise oriented. A clockwise cycle of φ corresponds to a *finite face* and thus to finite regions. The cycle of φ which is counterclockwise oriented is unique for each planar map. This cycle corresponds to a face named the *infinite face* of the map (φ, α) and is denoted by f^∞_φ. The corresponding region of f^∞_φ is equals to the complement of the union of all finite regions associated with the faces of the map (φ, α)(see Figure 13). Given a map (φ, α), we use the infinite face f^∞_φ, which represents nothing for segmentation purpose to store included information. One drawback of planar maps model is that such a model does not provide a representation of included faces or of holes. Given a map (φ, α), if we insert a closed edge (d,-d) which does not intersect any edge of the map (φ, α) we create a new map (φ', α') (see Figure 12). The location of the new edge (d,-d) in the map (φ, α) is not encoded by planar maps model. Included information is added by the two functions *parent* and *children* denoted by $\boldsymbol{Parent}$ and $\boldsymbol{Child}$. This two functions are defined as followed:

- The parent of an infinite face is the finite face which includes it.
- The children of a finite face f are infinite faces included in f.

By using functions $\boldsymbol{Parent}$ and $\boldsymbol{Child}$ we can define a new neighborhood relation which takes into account inclusion relations. This new neighborhood relation can be defined has followed.

Definition 2: Given two faces f_1 and f_2 of two maps (φ_1, α_1) and (φ_2, α_2), we say that f_1 is a neighbor of f_2 if and only if:

- $(\varphi_1, \alpha_1) = (\varphi_2, \alpha 2)$ and f_1 is an external neighbor of f_2 or
- f_1 is an external neighbor of $f^{\infty}_{\varphi_1}$ and $\boldsymbol{Parent}(f^{\infty}_{\varphi_1}) = f_2$ or
- f_2 is an external neighbour of $f^{\infty}_{\varphi_2}$ and $\boldsymbol{Parent}(f^{\infty}_{\varphi_2}) = f_1$.

An example of neighborhood is given on Figure 16. Neighborhood of one face can be computed using Algorithm 2.

```
find_neighbors(f,L)
{

    find_external_neighbors(f,L)
    if an infinite face belong to L replace it by its mother.
    for each f' in daughter(f)
        find_external_neighbors(f',L)

}
```

Fig. 2.

The permutations α and φ and the functions λ, $\boldsymbol{Parent}$ and $\boldsymbol{Child}$ are efficiently updated across split or merge operations. More details can be found in other papers about the fundations of the model [4] and about update algorithms [8].

The model described above, which allows us to describe relationships between faces has to be augmented in order to encode their geometry, (see Figure 14). Thus it is associated with an inter-pixel boundary representation (see section II) by putting in relation respectively boundary segments and map edges and boundary nodes and map vertices. Thanks to this correspondence between edges and segments, we can formalize definition of regions. A face f defined in a map (φ, α) corresponds to cycle $\varphi^*(\lambda^{-1}(f)) = (d_1, \dots, d_n)$. Each dart d_i corresponds to a segment s_i defined in the $P_{\frac{1}{2}}$ plane. Concatenation of segments $(s_1, \dots, s_n)$ creates a closed curve C_f. Region R_f corresponding to face f is then defined as the set of pixels inside curve C. Note that since, C_f is defined in $P_{\frac{1}{2}}$ plane it defines a Jordan curve. This property is used to compute parameters of regions and to update functions parent and children. Inter-pixel boundaries associated with topological maps can be seen as a geometrical embedding of topological maps. For this reason we call them discrete maps.

IV. Application

Discrete maps allows us to compute neighborhood relations between faces and to get the frontier of a given region R_f associated with a face f. This frontier which defines a Jordan curve is used to traverse the region R_f. This operation is performed thanks to a scan-line algorithm which traverse the frontier of R_f and mark some of its pixels as *Begin points* and other ones as *End point* (see Figure 18). Then the filling algorithm traverse all lines of the original image from Begin points to End points. The traverse of region is used to compute and attach attributes to faces. We have chosen to attach to each face the moments M_0, M_1 and M_2 defined by :

$$\begin{array}{lcl} M_0(f) & : & \text{number of pixel contained in region} R_f \\ M_1(f) & = & \sum_{(v_1,\dots,v_n)\in R_f}(v_1, \dots, v_n) \\ M_2(f) & = & \sum_{(v_1,\dots,v_n)\in R_f}(v_1^2, \dots, v_n^2) \end{array} \quad (1)$$

where n denotes the original image dimension and $(v_1, \ldots, v_n)$ denotes the value of a pixel. For instance $n = 1$ for a grey level image and $n = 3$ for a usual color image.

The functions M_0, M_1 and M_2 allow one to define more useful functions on labels. For example, the mean value $\mu(f)$ and the variance $var_i(f)$ of a face f may be defined by :

$$\begin{array}{l} \mu(f) = M_1(f)/M_0(f) \\ var_i(f) = \frac{1}{M_0(f)}(M_2(f)_i - \frac{M_1(f)_i}{M_0(f)}) \end{array} \tag{2}$$

where i represent the i^{th} axis in the feature space of dim n.

In our implementation we have chosen to compute by need the attributes of a face f and to mark attributes as invalid when a face f is split. These attributes can be efficiently updated during merge operations thanks to the following property :

$$\forall \;\; i \in \{0, 1, 2\} \;\; M_i(f_1 \cup f_2) = M_i(f_1) + M_i(f_2) \tag{3}$$

where $f_1 \cup f_2$ denotes the face resulting from the merge of faces f_1 and f_2. Thus the updating of moments for merge operations do not require to traverse merged faces.

Using a scan line algorithm and attributes of faces we can compute histograms, variance or squared error in order to decide if a face has to be split further or not. Using our model, we can easily and efficiently store labels of faces created by a split algorithm. This last property allows us to easily design recursive split algorithms. An example of recursive split algorithm based on squared error is given on Figure 3. Our data structure allows segmentation algorithms to create and handle any partition of the image. Thus this structure is quite general and may be used by all segmentation algorithms. Note that, since our structure remains efficient for most segmentation algorithms it allows one to design *meta segmentation algorithms*. By these terms we mean algorithms analyzing a region in order to apply the best suited algorithm to segment it. For example global segmentation algorithms such as clustering may be used in conjunction with edge detection algorithms. Applied on homogeneous regions edge detection algorithms may use global information about the region to be segmented. Nevertheless, the planar map model is based on regions, thus it do not allow insertion of dangling edges. This may be an inconvenient for edge detection algorithms which perform a detection of strong edges followed by a closure step. This inconvenient can be partially removed by closing dangling edges around pixels (see Figure 17). Edges added to close dangling edges can be marked and removed after the closure step.

```
recursive_split(label f, squared_error ref)
{

    sqe = M_0(f) \sum_{i=i}^{n} var_i(f)
    if(sqe/ref < ε)
        return

    split_face(f)
    L = give_inserted_faces()
    for all f' in L
        recursive_split(f',ref)
}
```

Fig. 3. *Recursive split algorithm*

Functions defined by equations 1 and 2 can also be used to design a merge score used for merge operations. Beveridge [3] has designed a merge score defined by the product of three functions measuring region feature similarity, region size and connectivity. Most of merging algorithms use some of these region parameters. Using our formalism Beveridge merge score can be expressed as :

$$S(f_1, f_2) = S_{sim}(f_1, f_2) S_{size}(f_1, f_2) S_{conn}(f_1, f_2)$$

$$\text{with :} \begin{cases} S_{sim}(f_1, f_2) & = & \frac{\mu(f_1) - \mu(f2)}{max(1, \Sigma(f_1) + \Sigma(f_2))} \\ S_{size}(f_1, f_2) & = & min(M_0(f_1), M_0(f_2)) \\ S_{conn}(f_1, f_2) & = & \frac{min(|\partial f_1|, |\partial f_2|)}{|\partial f_1 \cap \partial f_2|} \end{cases}$$

where Σ denotes the standard deviation and ∂f denotes the frontier of region R_f.

Beveridge adds some parameters in order to normalize the functions S_{sim} S_{size} and S_{conn}. Using such functions a low value of similarity function $S(f_1, f_2)$ indicates that the faces should be merged whereas a high value of the $S(f_1, f_2)$ indicates they should not. Note that S_{sim} and S_{size} use moments and functions derivate from moments. Thus, such functions can be efficiently computed thanks to moments linearity as defined in equation 3. The function S_{conn} uses segment size. This parameter can also be efficiently updated. Moreover using our implementation label of merged face $f_1 \cup f_2$ is equal to $\lambda(f_1)$ or $\lambda(f_2)$. This last property allows us to efficiently update the set of faces to be proceeded during merge operations. Thus, using neighborhood relations and merge scores based on attributes of faces, efficient merge algorithms can easily be designed. An example of merge algorithm based on Beveridge merge score is provided on Figure 4. This merge algorithm receives a list L of faces and merge the faces contained in L until the minimum value of $S(.,.)$ on L is greater than a given threshold ϵ.

```
merge(list of faces L)
{

    do
    {
        for all f_1 in L
        {
            store the neighborhood of f_1 in list V (see definition 2)
            min_{f_1} = Min_{f_2 ∈ V} S(f_1, f_2)
            if(min_{f_1} < min)
            {
                min = min_{f_1}
                f = f_1
                f' = f_2
            }
        }
        if(min > ε)
            return
        f ∪ f' = merge(f, f')
        remove f and f' from L
        insert f ∪ f' in L
    }
    while(ever)
}
```

Fig. 4. *An example of merge algorithm*

Since the same data structure is used to perform split and merge operations, split operations can be performed after the merging step. This last point allows us to partition further a region which has been missed during the first splitting step or incorrectly merged during the merging step. This ability to alternate split and merge operations allows us to design interactive segmentation algorithms which allow users to select the face to be merged or to be split further. Moreover split operations followed by a merging step allows us to remove noises and to perform merges in order to give to users meaningful faces at each step of the algorithm.

This data structure provides also any editing operations such as for instance segment removing, boundary face removing contour insertion or contour smoothly. Since the data structure is not altered by editing, the editing step is no longer a final step of the segmentation process but can be followed by any other one. It is thus possible to edit a region and then to recursively split it in order to locally refine the segmentation.

Finally it is possible to address regions by using high level topological features. From the topological map it is for instance very easy to get the set of faces included into a finite face or the set of faces being at a bounded distance of another one.

V. Experiments and discussions

In this section we describe an example of image segmentation performed with our environment. The working image is the well known test image usually called *lenna* (see Figure 19-a). This image presents several difficulties for the purpose of image segmentation. Some frontiers, such as for instance the top of the hat or the top of the shoulder are very tenuous. Moreover the image, which size is 512×480 pixels, has a large number of different colors, each one being relatively few frequent. Some parts, like the hairs, correspond to the high frequencies when other ones, like the lips correspond to low frequencies.

The segmentation is initiated by running a split algorithm on the whole image. This split algorithm based on a partition of the feature space is described in [7]. The result of the split step is shown in figure 19-b. A lot of small faces corresponding to noise have been created by the split algorithm. Thus we perform a merge step by using algorithm described in Figure 4. In that case, the list L of faces is initialized with all faces displayed in Figure 19-b. The result of the merge step is displayed in figure 19-c. The produced regions correspond to the main area of the image. We have used a high threshold for our merge score in order to remove insignificant faces induced by the clustering step. As a matter of fact, our clustering algorithm, may create a face enclosing a group of pixels with features highly represented in the whole image even if, from a local point of view this face is not significant. The high threshold of our merge score remove these faces and keeps only the main area of the image. Since our thresholds for split and merge algorithms are relative to the face to be segmented, the segmentation algorithm will become more and more precise in regard to the homogeneity of the initial face. Thus, finer details such as the lips will be segmented later from more homogeneous faces.

As sketched above splits and merges can be iterated without any restriction. Moreover the data structure allows us to recursively initiate a segmentation on a sub-region, using local criteria. In order to achieve the segmentation of the hat the region including the missing part of the hat border (see Figure 19-d) is selected by picking with the mouse anywhere in the region area. In Figure 19-e is displayed the result of the split algorithm on the face which contains the hat. Figure 19-f shows the partition of this face after the merge step. The frontier of the hat is now entirely build but presents several irregularities. These irregularities will be treated later.

In the following step we are going to refine the big region including the right part of the hat, the face, and the shoulder (see Figure 20-a). We thus applied a split step followed by a merge step on this face. The partition of this face is displayed in Figure 20-b. Now we focus on the part of the image representing the young women. Thus we remove from the data structure all the regions corresponding to the background.

The interface we have developed is still very simple (selection of a face or of a segment). In spite of this simplicity it is very easy to get the segmented image displayed in Figure 20-c. Note that since the edition of the segmented image and the refinement of faces are done with the same data structure, faces can be further refine or merged after an edition step. We have kept the region above the hat in order to refine the frontier of the hat. Thus we remove the segment delimiting the left part of the top of the hat which appears too much rough. (see Figure 20-d). The feature space of the resulting face being more homogeneous, the clustering algorithm will give more accurate results. The result of a split step followed by a merge step applied on this face is displayed in Figure 20-e. Once undesirable regions have been removed, we get the image presented in Figure 20-f. The next step consists in refining the face of the woman. The result of the split and merge algorithm applied on this face is displayed in Figure 21-a while the result of the editing step is displayed in Figure 21-b. We also refine the face containing the feathers and the hairs, the resulting faces are displayed in Figure 21-c and Figure 21-d. Finally the region corresponding to the right part of the hat and the woman's forehead is selected in order to close the lower part of the hat. The result of the split and merge algorithm is displayed in Figure 21-e and the result of the editing step is displayed in Figure 21-f.

Only a few number of minor editing of the geometry of segments have been performed to improve the final result such as on the segment separating the shoulder and the face.

VI. Conclusion

In this paper we have described a data structure for image segmentation allowing a unified representation of both topology and geometry of the segmented image. Contrary to other usual models, this models is not devoted to a specific segmentation method and allows any combinations of any methods.

This model has been implemented in standard C under Unix and X-Window. It provides a powerful framework for interactive segmentation. This implementation constitutes the kernel of a segmentation platform used to develop meta-segmentation algorithms. Another direction of research we are currently exploring is the segmentation of an image sequence with topological invariant.

References

[1] B. Bahnu and B.A. Parvin. Segmentation of natural scenes. *Pattern Recognition*, 20(5):487–496, 1987.

[2] D.H. Ballard and Jack Sklansky. A ladder-structures decision tree for recognizing tumors in chest radiographs. *IEEE Transactions on computers*, 25(5):503–513, may 1976.

[3] J.R Beveridge, J. Griffith, R. Kohler, A.R. Hanson, and E.M. Riseman. Segmenting images using localized histograms and region merging. *International Journal of Computer Vision*, 2:311–347, 1989.

[4] J.P. Braquelaire and J.P. Domenger. An effective representation of segmented images. Submitted, 1996.

[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. Split, merge and grouping : a new approach for color image quantization. Submitted to Computer Vision Journal, may 1996.

[8] L. Brun and J.P. Domenger. Incremental modifications on segmented image defined by discrete maps. Submitted, 1996.

[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] P. Chen and T. Pavlidis. Segmentation by texture using a co-occurence matrix and a split and merge algorithm. *Computer Graphics, and Image Processing*, 10:172–182, 1979.

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

[12] 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.

[13] 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.

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

[15] C. Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation*. Thèse de doctorat, Université Montpellier II, 24 novembre 1995.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

[22] U. Levi, G. Montanari. A grey-weighted skeleton. *Information Control*, 17:62–91, 1970.

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

[24] 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.

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

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

[27] A. Rutovitz and J. L. Pfaltz. Data structures for operations on digital images. *Pictorial Pattern Recognition*, pages 105–133, 1968.

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

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

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

[31] R. I. Taylor and P. H. Lewis. Colour image segmentation using bondary relaxation. *IEEE*, pages 721–724, 1992.

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

Appendix

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Fig. 5. *Definition of region 1 by maximal blocks*

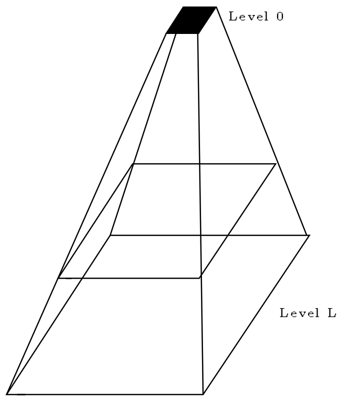


Fig. 6. *An M-pyramid associated to an $2^L \star 2^L$ original image*

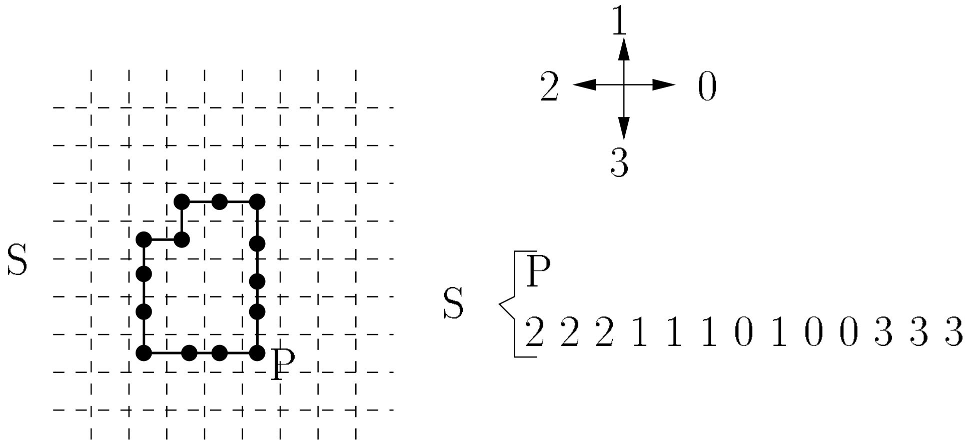


Fig. 7. *Definition of a region with a 4-connected path in integer plane*

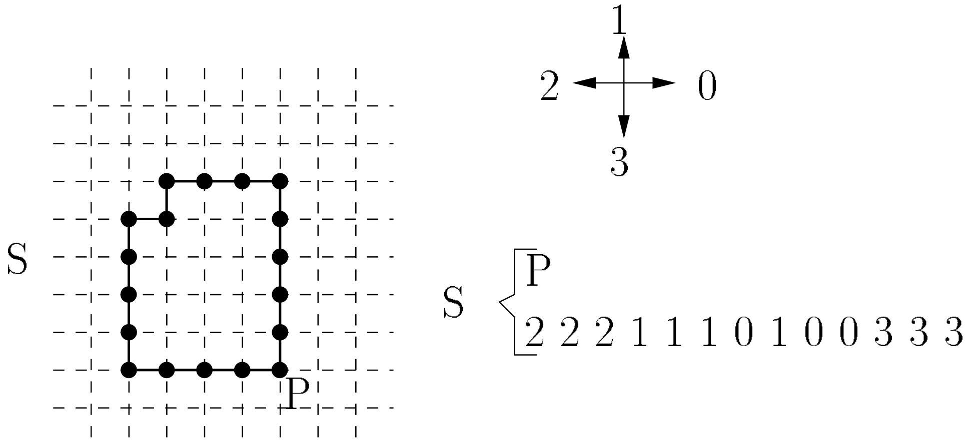


Fig. 8. *Same region than the one of Figure7 but defined in the half-integer plane*

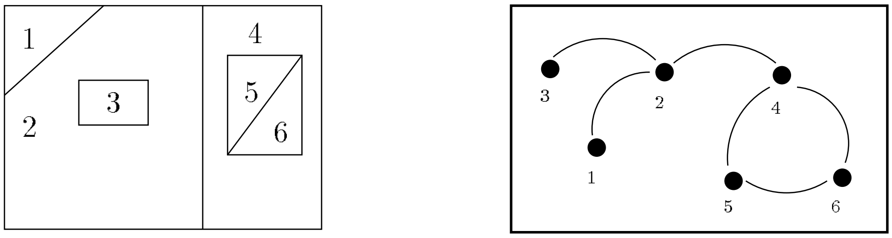
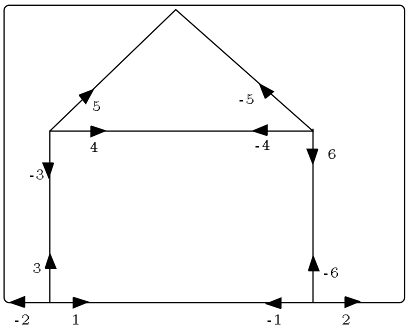


Fig. 9. *Region adjacency graph of a set of regions*

Fig. 10. *Merge of two nodes in the Region adjacency graph*

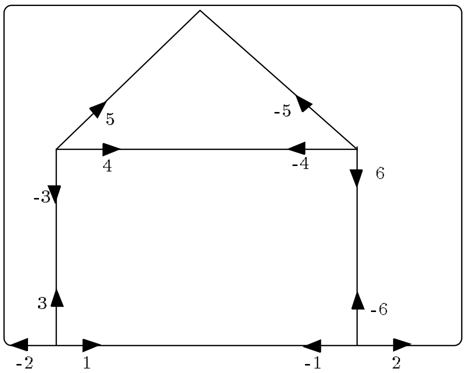


σ

$\sigma = (3,-2,1)(-6,-1,2)(-5,-4,6)(5,-3,4)$

$\forall b \in \{1,\ldots,6\} \alpha(b) = -b$ and $\alpha(-b) = b$

Fig. 11. *Example of permutations σ and α*



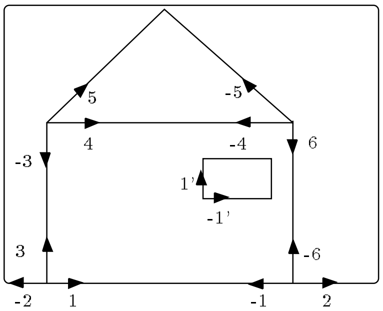
σ

$\sigma = (3,-2,1)(-6,-1,2)(-5,-4,6)(5,-3,4)$

$\varphi = (-4,5)(4,6,-1,3)(1,2)(-5,-3,-2,-6)$

$\varphi(-4) = \sigma \circ \alpha(-4) = \sigma(4) = 5$

Fig. 12. *Example of permutations φ and α*



$\phi = (-4,5)(4,6,-1,3)(1,2)(-5,-3,-2,-6)$

$\phi' = (1')(-1')$

$f^{\infty}_{\phi} = \lambda(1)$

$f^{\infty}_{\phi'} = \lambda(-1')$

σ

$Parent(\lambda(-1')) = \lambda(3)$

$Child(\lambda(3)) = \{\lambda(-1')\}$

Fig. 13. *Example of included regions*

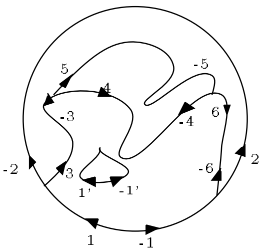


Fig. 14. *This figure gives a permutation φ equal to the one of Figure 13*

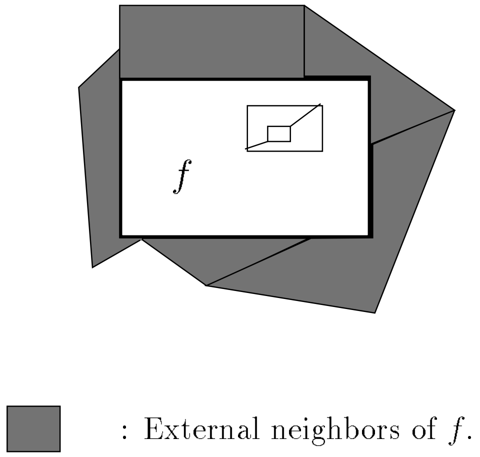


Fig. 15. *An example of external neighborhood*

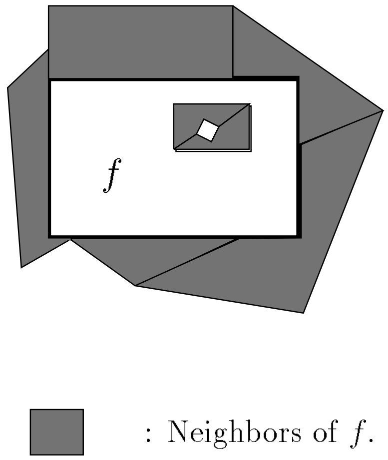


Fig. 16. *An example of neighborhood*

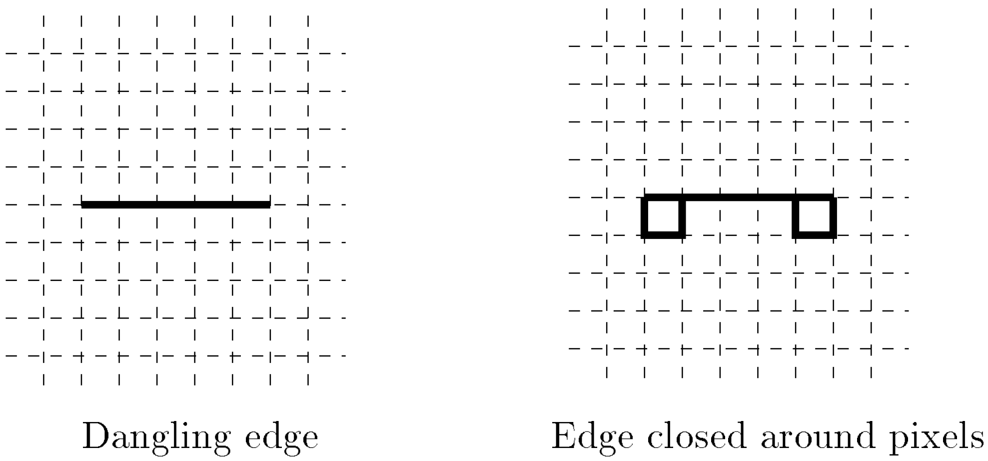
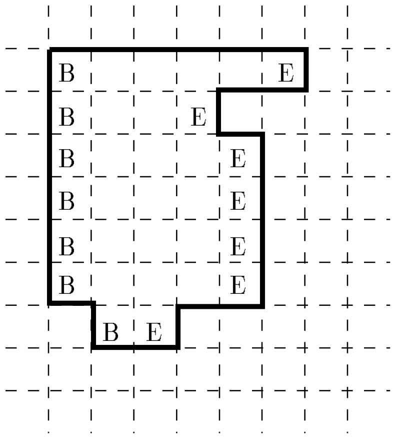


Fig. 17. *Closure of a dangling edge*



B : Begin point
E : End point

Fig. 18. *Marked points of a region for purpose of reconstruction*

(a) (b) (c) (d) (e) (f)

Fig. 19. An example of segmentation session.

Fig. 20. An example of segmentation session – continuing.

Fig. 21. An example of segmentation session – continuing.