

A top-down construction scheme for irregular pyramids

Romain Goffe, Luc Brun, Guillaume Damiand

▶ **To cite this version:**

Romain Goffe, Luc Brun, Guillaume Damiand. A top-down construction scheme for irregular pyramids. International Conference On Computer Vision Theory And Applications (VISAPP'09), Feb 2009, Lisboa, Portugal. pp.163-170. hal-00362234v3

HAL Id: hal-00362234
https://hal.science/hal-00362234v3

Submitted on 26 Apr 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A top down construction scheme for irregular pyramids

Romain Goffe[1], Luc Brun[2], and Guillaume Damiand[3]

[1] SIC-XLIM, Université de Poitiers, CNRS, UMR6172, F-86962, Futuroscope Chasseneuil, France
goffe@sic.univ-poitiers.fr
[2] GREYC, ENSICAEN, CNRS, UMR6072, 6 Boulevard du Maréchal Juin, F-14050, Caen, France
luc.brun@greyc.ensicaen.fr
[3] LIRIS, Université de Lyon, CNRS, UMR5205, F-69622, Villeurbanne, France
guillaume.damiand@liris.cnrs.fr

Abstract Hierarchical data structures such as irregular pyramids are used by many applications related to image processing and segmentation. The construction scheme of such pyramids is bottom-up. Such a scheme forbids the definition of a level according to more global information defined at upper levels in the hierarchy. Moreover, the base of the pyramid has to encode any single pixel of the initial image in order to allow the definition of regions of any shape at higher levels. This last constraint raises major issues of memory usage and processing costs when irregular pyramids are applied to large images. The objective of this paper is to define a top-down construction scheme for irregular pyramids. Each level of such a pyramid is encoded by a combinatorial map associated to an explicit encoding of the geometry and the inclusion relationships of the corresponding partition. The resulting structure is a stack of finer and finer partitions obtained by successive splitting operations and is called a top-down pyramid.

Key words: Segmentation; Irregular pyramid; Topological model;Combinatorial map;

1 Introduction

Quadtrees [DRH80, JR94] and regular pyramids [JR94] belong to the first hierarchical data structures introduced within the computer vision framework. Both models are based on psycho-visual properties: focus of attention, for data structure based on recursive split such as quadtrees, and successive processings by neural layers, for bottom-up regular pyramids. Segmentation using quadtree data structures is based on a recursive subdivision of a basic shape (e.g. a square).

On the other hand, regular or matrix pyramids are defined as a stack of images with decreasing resolutions. An entity (square or pixel) defined at a given level of a pyramid is associated to a connected set of entities below, called a reduction window [BCR90]. Both encoding schemes induce several drawbacks on the segmentation process [BCR90].

The irregular pyramid framework introduced by Meer and Montanvert [Mee89, MMR91] partially solves these drawbacks: the stack of partitions is encoded as a stack of successively reduced graphs. Irregular pyramids [Mee89, MMR91, JM92, BK03] may only be built using a bottom-up construction scheme.

However, a bottom-up scheme requires an explicit encoding of the base level image in order to define regions with any shapes at higher levels. Moreover, in a bottom-up pyramid, each newly created region has no prior information about its parents (defined at a later stage). This last constraint prevents the management of the regions from depending on the properties of their parents in the pyramid.

The objective of this paper is the definition of a top-down hierarchical data structure by extending the model of two-dimensional topological maps. For many applications related to image segmentation, it is critical to minimize memory requirements, mainly for those processing large images. A top-down approach rules out the constraint of the explicit storage for the base: only split regions are kept in memory. Besides, it offers a perceptual advantage as major features of an image are discerned first in the pyramid, contrary to bottom-up models.

We first recall in Section 2, the basics of the different models used to define our top-down irregular pyramid framework. Then, Section 3 defines our model of top-down pyramid. We present in Section 4 its construction scheme. Section 5 details the basic operations used to build a new level of the pyramid. We finally provide, in Section 6, several experiments which allow to evaluate the computational times and memory requirements of our model.

2 Recalls

2.1 Combinatorial maps

A combinatorial map encodes all the subdivisions and incidence relationships of a topological space [Lie89]. In two dimensions, it is composed of vertices, edges and faces, respectively defined as cells of 0, 1 and 2 dimensions and noted i-cells. The border of an i-cell is a set of $(j < i)$-cells. Two i-cells are said *incident* if one belongs to the border of the second while they are said *adjacent* if they are both incident to the same $(j < i)$-cell. The *degree of an i-cell* is the number of adjacent $(i + 1)$-cells and a *dangling edge* is an edge incident to a degree 1 vertex. Adjacency relations are represented by operators noted β_i and applied to darts, as we will call the abstract basic elements resulting from a complete decomposition of the image (Figure 1).

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

1. *$\mathcal{D}$ is a finite set of darts;*
2. *β_1 is a* permutation[1] *on $\mathcal{D}$;*

[1] A *permutation* is a one to one mapping from S onto S.

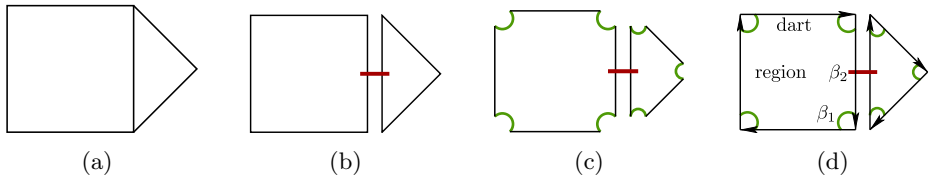


Figure 1: Construction of a 2-map by successive decompositions. (a) Original image; (b) Disconnected faces; (c) Disconnected edges; (d) 2-map: darts are the basic elements represented by arrows, β_1 relations are represented with arcs and β_2 with bold segments.

3. β_2 *is an* involution[2] *on* $\mathcal{D}$.

Intuitively, a combinatorial map may be understood as a planar graph where relations on edges are explicitly defined by β_i operators. Darts allow to differentiate the two extremities of an edge and thus, are assimilated to half-edges. Each dart belongs to a single vertex, edge and face of the map. The β_1 permutation links each dart of a face to the next dart encountered while turning clockwise around the face (Figure 1). The β_2 involution links each dart of an edge to the other dart of the edge which has an opposite orientation (shared edge between the square and the triangle in Figure 1). Two darts linked by β_i are said i-sewn and two 2-sewn darts belong to two adjacent faces.

2.2 Topological maps

Because 2-maps can only represent the topology of connected objects, we introduce the notion of topological maps [BMD03, DBF04], an extension of combinatorial maps that uses three different models to encode: topological relations, geometrical information and inclusion relationships between regions.

Topology is based on a 2-map which is *minimal* according to its number of cells (Figure 3(a)). Although combinatorial maps only represent the topology of the space, geometric elements can be added easily. This association is called *embedding*. The geometry relies on the *interpixel framework* where an image is considered as a subdivision of a two-dimensional space in a set of 2-cells, 1-cells and 0-cells, respectively called *pixels*, *linels*, and *pointels* (Figure 2). Each border between two regions is thus defined as a set of linels. Since each dart corresponds to an oriented boundary, the embedding of a dart defines an order over the set of linels belonging to this border. The set of linels composing a dart can be represented explicitly as a sequence of linels or implicitly, using a two-dimensional matrix of the size of the image [BMD03, DBF04] (Figure 3(b)).

A *region* is a set of darts delimited by a β_1-loop. Each one has a *representative dart* which allows to retrieve a dart of a given region (e.g. used as a starting point to

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

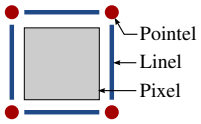


Figure 2: Representation of the interpixel framework: an image is composed of pixels, linels and pointels.

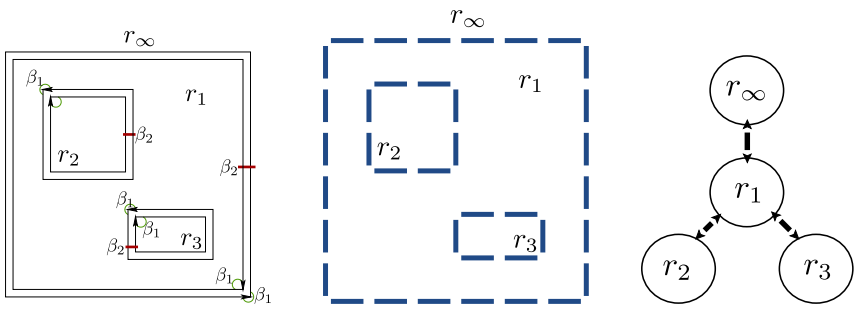


Figure 3: The three different models composing a topological map. (a) A 2-map representing the topology with darts and β_i relationships; (b) A geometrical matrix that points out active linels and pointels; corresponding elements are linked together (for visibility reasons, dotted lines represent only up/down relations between regions); (c) A tree of regions for inclusion relations.

find the external border of the region). A set of adjacent regions is called a *connected component* and the union of all the regions create a topologically closed space since we represent the *infinite* region [3] which encodes the background of the image. A region included into another one is called a hole and defines an internal border for the including region. An *inclusion tree of regions* represents the inclusion relationships of the structure: the father of any region within the tree is defined as the one which includes it in the image (Figure 3(c)).

A topological map is a suitable model for image processing which has been proven *complete* (represents both topology and geometry), *minimal* (retiring any element would change the topology) and *unique* (two topologically equivalent partitions have the same map) [DBF04]. In practice, the minimality is required to decrease the number of cells and minimize memory usage and the completeness insures that we can encode partitions with regions of any geometry.

2.3 Pyramids

Simple graph pyramids, first introduced by Meer and Montanvert [Mee89, MMR91], then developed by Jolion [JM92] are defined as a stack of simple graphs successively

[3] For visibility reasons, the infinite region may not be represented in some figures.

reduced. Within the segmentation framework, each graph of such a pyramid encodes a partition. Due to the limitation of the simple graph data structure, many issues are encountered when we have to update these graphs after splitting operations.

Combinatorial pyramids [BK03] are built from an initial combinatorial map successively reduced by a sequence of contractions and removal operations. These operations are ruled by a contraction kernel (forest of the initial combinatorial map) and a removal kernel (forest of the dual combinatorial map). These structures are *bottom-up* and the initial combinatorial map (the base) is the most detailed level: the embedding of each dart of the base corresponds to a linel. Therefore, the receptive field of any dart may be retrieved from its receptive field and the embedding of the darts defined at the base [BK03]. Besides, using forests avoids disjunctions of connected components when performing merging operations: two connected components are linked by a bridge if one is included into the other [BK06]. The model of bottom-up combinatorial pyramid has been generalized to encode all n-dimensional, orientable or not and with or without boundary subdivisions [Lie89, SDL06].

Contrary to bottom-up methods, based on an explicit encoding of the base of the pyramid, a top-down approach allows to encode only the upper levels, resulting in a major memory reduction. Moreover, the focus of attention, encoded by the top-down scheme, can adapt the segmentation of each region according to the features of its parents (e.g. with medical images, the segmentation of cells in a tissue depends on the tissue itself).

Within a top-down scheme, we have to give up on bridges to represent inclusion relationships. Indeed, the management of the additional connections encoded by the bridges during splitting operations may induce cumbersome computations. For example, the insertion of an edge at the two endpoints of a bridge may create an artificial face which has to be detected and removed. This is why bridges are replaced by the use of inclusion tree of regions.

Moreover, since the top-down construction scheme avoids an explicit encoding of the base, the geometry of the pyramid's partitions cannot be implicitely encoded at the base level. The borders of the partition have thus to be explicitely encoded. These last arguments justify the use of topological maps as the basis of our top-down model.

3 Model for top-down pyramids

Each level of our top-down pyramid model is encoded by a topological map, defined by: a 2-map for the topology, an encoding for the geometrical embedding of darts and a tree of regions for inclusion relationships.

Definition 2 (Top-down topological pyramid)**.** *Let* $(n, m, k) \in \mathbb{N}^3$*. A top-down pyramid* P *is defined by* $P = \{G^k\}$ *where,* $\forall k$*,* $0 \leq k \leq m$*:*

1. $G^k = (\mathcal{D}^k,\ \beta_1^k,\ \beta_2^k)$ *is a topological map;*

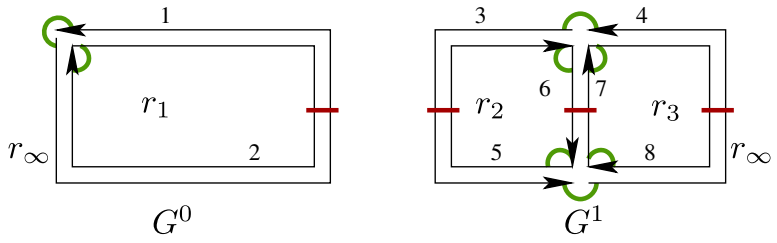


Figure 4: A top-down topological pyramid P, composed of two levels G^0 and G^1. Numbers designate darts, β_1 relations are represented by arcs and β_2 by segments. G^1 is deduced from G^0 by splitting region r_1 into two regions r_2 and r_3.

2. G^{k+1} is deduced from G^k by performing splitting operations.

Since any level of the top-down pyramid results from splitting operations (2), every region of G^k has a descendant in G^{k+1} and every region of G^k has *at most* one antecedent in G^{k-1} (same for darts). Thus, the model is a causal structure [GCM06] and defines a hierarchy of regions.

As a hierarchical data structure, the model has to represent objects and relations through the levels of the pyramid. So, each dart and region of a map G^k is connected to its parent in G^{k-1} and its child in G^{k+1} (also called ascendant/descendant or up/down). Note that, although each element (dart or region) has a single descendant, there is no loss of information: we can retrieve for each element the corresponding set of elements in a lower level. Indeed, the set of children of a given element is connected so, up and down relations allow to start from a descendant and to find all the neighbors which have the same "up". Neither elements from the top level nor elements newly created on a level have an antecedent and elements belonging to the base do not have a descendant but several elements may have the same parent (Table 1).

Table 1: Parent/child relations in pyramid P from Figure 4. (a) Between darts; (b) Between regions.

$\mathcal{D}$	d_{up}	d_{down}
1	-	4
2	-	5
3	1	-
4	1	-
5	2	-
6	-	-
7	-	-
8	2	-

(a)

$\mathcal{R}$	r_{up}	r_{down}
1	-	2
2	1	-
3	1	-

(b)

In the following, P denotes a top-down pyramid composed of $m+1$ levels num-

bered from 0 to m (m is called the *depth* of the pyramid). Globally, an exponent k refers to level $k+1$ of the pyramid: G^k is the map of level $k+1$, $\mathcal{D}^k$ (resp. $\mathcal{R}^k$) is the set of all darts (resp. regions) composing level $k+1$. An edge may be noted (d, d') where d and d' are the 2-sewn darts which compose it and $r^k(d)$ denotes the region of dart d in level k.

4 Construction scheme

This section outlines the global operations constructing a pyramid. The construction is incremental: it starts from the top and adds new levels one by one at the bottom. It is composed of three main steps: the construction of the first (top) level, the creation of a new level by copying the bottom and the segmentation of the level that has just been added.

Several methods can be considered to build the first level of a pyramid but only two are considered so far. The first method creates a map composed of a single region enclosing the image and the infinite region for the outside of the image. The second method extracts a first topological map from a segmentation of the image in few regions.

We create a new level by duplicating the last one of the pyramid. This is why, we build a map equivalent to the bottom (same topology, same geometry and same tree of regions), link corresponding elements between the two levels, and finally, add this map at the bottom of the pyramid (Algorithm 1 and Figure 5(b)).

Algorithm 1: Duplication of a level

Data: A pyramid P of depth $m+1$.
Result: A pyramid P of depth $m+2$.
create a new void map G^{m+1};
copy the geometry of G^m into G^{m+1};
foreach *dart* $d_{up} \in G^m$ **do**
    create a new dart d_{down} in G^{m+1} ;
    set d_{up} as parent of d_{down};
    sew (in G^{m+1}) d_{down} with the corresponding down darts of $\beta_1(d_{up})$ and $\beta_2(d_{up})$;
foreach *region* $r_{up} \in G^m$ **do**
    create a new region r_{down} in G^{m+1};
    set r_{up} as parent of r_{down};
    establish inclusions in G^{m+1} by setting the relations of r_{down} like those of r_{up};
set G^m as parent of G^{m+1};

The last step of the construction process is the segmentation of the level that has just been duplicated. This segmentation is based on splitting and merging operations

that transform the level (Algorithm 2).

Algorithm 2: Segmentation of a level

```
Data: A level G^k of a pyramid P.
Result: P with a new segmentation on level G^k.
foreach region r ∈ G^k do
1     if splitting criterion(r) is true then
2         split(r);
3 merge(G^k, merging criterion);
4 simplify G^k;
5 compute the new tree of regions;
```

Algorithm 2 is composed of five main steps:

- `line 1`: The splitting criterion indicates if a region has to be segmented. It is used upstream from the construction process as an optimization since it moves uninteresting areas further apart (notion of *focus of attention* [JR94]);
- `line 2`: This step decomposes region r into a set of square-unit regions, each one enclosing a single pixel (Figure 5(c)). This operation is detailed in Section 5.1;
- `line 3`: The merging criterion determines if two adjacent regions should be merged. In order to preserve the causality property, we restrict the merging operation to new regions resulting from the split of a *same* region (Figure 5(d)). Therefore, two different regions of a level G^p will never be merged in a level G^q, $p < q$. This operation is detailed in Section 5.2;
- `line 4`: The simplification step removes all remaining 2-degree vertices;
- `line 5`: Because new regions are created, the tree for inclusion relations has to be rebuilt. Indeed, it would be too expensive to keep it up-to-date as many regions are created from a level to another.

Steps `line 4` and `line 5` rely on the algorithms defined for topological maps [DBF04]. Figure 5 illustrates a simple example of the building process.

5 Basic operations

5.1 Splitting operation

As mentioned in Section 4, our splitting step decomposes a region into a set of basic regions, each one enclosing a single pixel. Later, a merging step will merge these regions: since any couple of adjacent regions may be merged, any subdivision of the initial region can be encoded. The operation insures that each created region is both topologically and geometrically correct (Figure 6).

Assuming that the region to split is denoted r, Algorithm 3 describes the splitting operation which may be divided into the following steps:

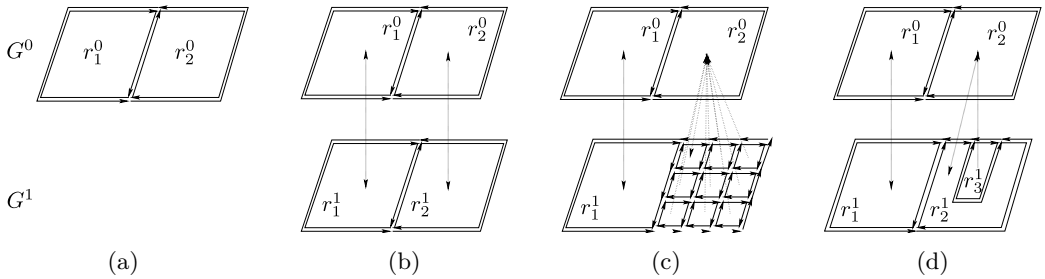


Figure 5: Main steps of the construction process of a top-down pyramid. (a) Initial step: the pyramid is composed of a single level G^0; (b) Duplicate and link: G^1 is a copy of G^0 and corresponding elements are linked together (for visibility reasons, dotted lines represent only up/down relations between regions); (c) Split: r_2^0 is split into a set of square-unit regions enclosing a single pixel; (d) Merge and simplify: some of the created regions are merged and draw r_2^1 and r_3^1.

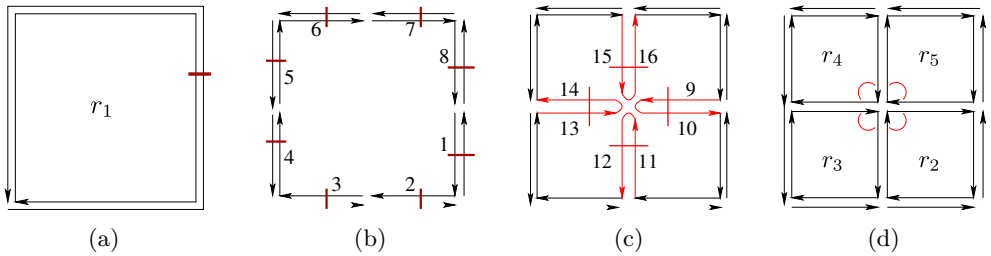


Figure 6: Decomposition in four steps of the splitting process applied on a region r_1 enclosing four pixels. (a) Initial region; (b) Splitting edges: darts 1 to 8 are stored into *list* (external border of r_1); (c) Insertion of 4 dangling edges while testing darts 1, 3, 5 and 7; (d) Sewing correctly dangling edges while testing darts 9 and 13.

Algorithm 3: Splitting region

Data: A region r.
Result: Region r is split into a set of basic regions enclosing a single pixel.

split edges of r into unit edges;
create *list* containing every dart of r;
while $\exists\ d \in list | d$ *is unmarked* **do**
    $l \leftarrow getLinel(d)$;
    if $l^{\perp}$ *is not activated* **then**
        insert $edge(d_i, d_j)$ on d;
        add(d_i, d_j) at the end of *list*;
    else if $\beta_2(d) = \beta_1(d)$ **then**
        1-sew correctly d and $\beta_2(d)$ around the pointel p incident to $\beta_2(d)$;
    mark(d);

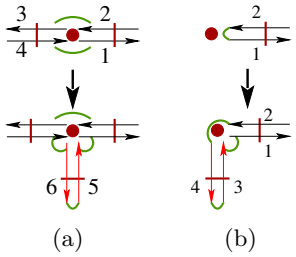


Figure 7: Insertion of a dangling edge on dart 1. (a) Insertion of edge $(5, 6)$ on a degree 2 vertex; (b) Insertion of edge $(3, 4)$ on a degree 1 vertex.

- `line 1`: All the edges belonging to the borders of r are split into one-linel long edges to allow further edge insertions;
- `line 2`: We create a *list* which initially contains all the darts resulting from the previous decomposition of external and internal borders of the region;
- `line 3`: We retrieve the geometry associated to the dart. As all edges have been split, the embedding of each dart of *list* is a single linel;
- `line 4`: The external border of a region is clockwise oriented. Let us denote by l_d and $l_d^{\perp}$, the oriented linels encoding respectively the embedding of the dart d and the next linel encountered after l_d when turning counter-clockwise around the pointel associated to d (e.g. in Figure 6, $l_2^{\perp} = l_1$). Since the β_1 permutation connects two consecutive darts in a clockwise orientation around a face, only $l_d^{\perp}$ needs to be considered at this stage. If other darts remain around the vertex incident to $\beta_2(d)$, they will be considered during further iterations.
- `line 5`: Actually, an edge insertion on a dart d consists in adding two one-linel long darts d_i and d_j (whose embedding is perpendicular to l_d) on the pointel p incident to d. Two configurations may happen as described in Figure 7;
- `line 6`: In order to sew the 2 darts of the dangling edge $e = (d, \beta_2(d))$, we geometrically look for edges perpendicular to e as illustrated in Figure 8. This operation 1-sew the two darts according to its number (one or two) of perpendicular edges. At least one perpendicular edge exists (inserted during the previous iteration when processing $\beta_2(d)$). If four edges are incident to p, processing e will sew two edges around p and the two others will be sewn in a further iteration (Figure 6).

This process ensures that each linel within the initial region is added. Moreover, no dangling edges remain: the initial region is initially minimal (i.e. without dangling edges) and if one is inserted, it is added to the list and then processed and correctly sewn. Consequently, the splitting operation produces a set of square-unit regions corresponding to the initial region.

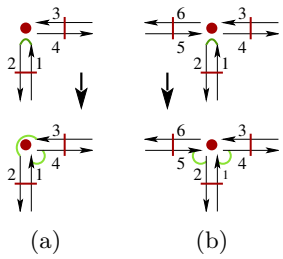


Figure 8: 1-sewing operations when processing dangling edge $(1, 2)$. (a) One perpendicular edge: 1-sew $(1, 4)$ and $(3, 2)$; (b) Two perpendicular edges: 1-sew $(1, 4)$ and $(5, 2)$.

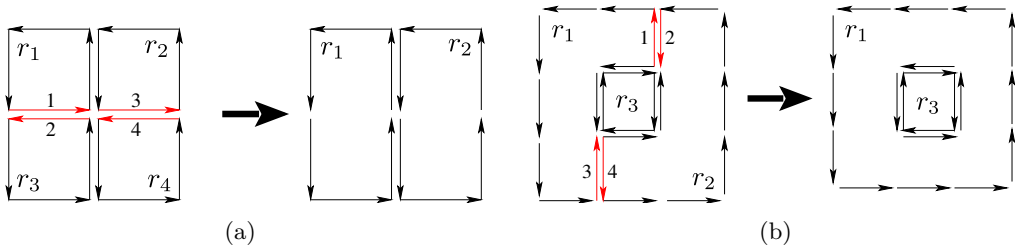


Figure 9: Configurations encountered during the merging operation between two adjacent regions. (a) Simple adjacency between (r_1, r_3) and (r_2, r_4): the corresponding edges $(1, 2)$ and $(3, 4)$ are removed if the merging criterion is verified; (b) Multi-adjacency between (r_1, r_2): if $(1, 2)$ has been removed, $(3, 4)$ is removed independently of the merging criterion.

5.2 Merging operation

Once the splitting process is done, we need to traverse all the created square-unit regions to segment the level according to our merging criterion. Actually, the merging operation is a global process which operates on a list of darts corresponding to the edges that were inserted by the splitting operation. This solution was selected for optimization matter as it avoids a complete traversal of all the regions. Each edge contained in this list is incident to a couple of adjacent regions (r, r').

Two configurations may be encountered: simple or multi adjacency. In the first case, the edge is removed if the merging criterion is satisfied for (r, r') (Figure 9(a)). In case of multi-adjacency, r and r' share several edges. If the merging criterion determines that (r, r') must be merged, all the shared edges have to be removed (Figure 9(b)). Algorithm 4 details the whole process:

- `line 1`: This list allows to traverse the regions created by the splitting operation instead of traversing the whole map. Note that we only need to store one dart d per edge (the second one is $\beta_2(d)$);
- `line 2`: The first condition aims to detect cases of multi-adjacency. The merging

Algorithm 4: Merging operation

```
1 Data: A list of darts corresponding to all the edges inserted on G^k during
        splitting operation.
  Result: G^k segmented according to merging criterion.
  foreach dart d ∈ list do
2     if r(d) = r(β2(d)) or merging criterion(r(d), r(β2(d)))is true then
3         turn off getLinel(d) (geometry);
4         relabel the darts of r(β2(d));
5         remove d and β2(d) (topology);
```

criterion is a test between two adjacent regions along the current edge (i.e. $(d, \beta_2(d))$);

- `line 3`: Let the geometry know that the linel is not active any more;
- `line 4`: This step updates the darts previously composing $r(\beta_2(d))$ as now belonging to $r(d)$;
- `line 5`: Removal of the two darts, according to the method in [DL03].

The only constraint applied to the splitting and merging operations is to preserve the causality of the structure: merging is thus restricted to the basic regions generated by the split of a same region. Therefore, within these regions, our merging operation is unrestricted and may group into a single region, any connected set of pixels. Any partition of the initial region may thus be encoded by our split and merge process. Contrary to quadtrees, our splitting operation is independent of any geometrical constraint.

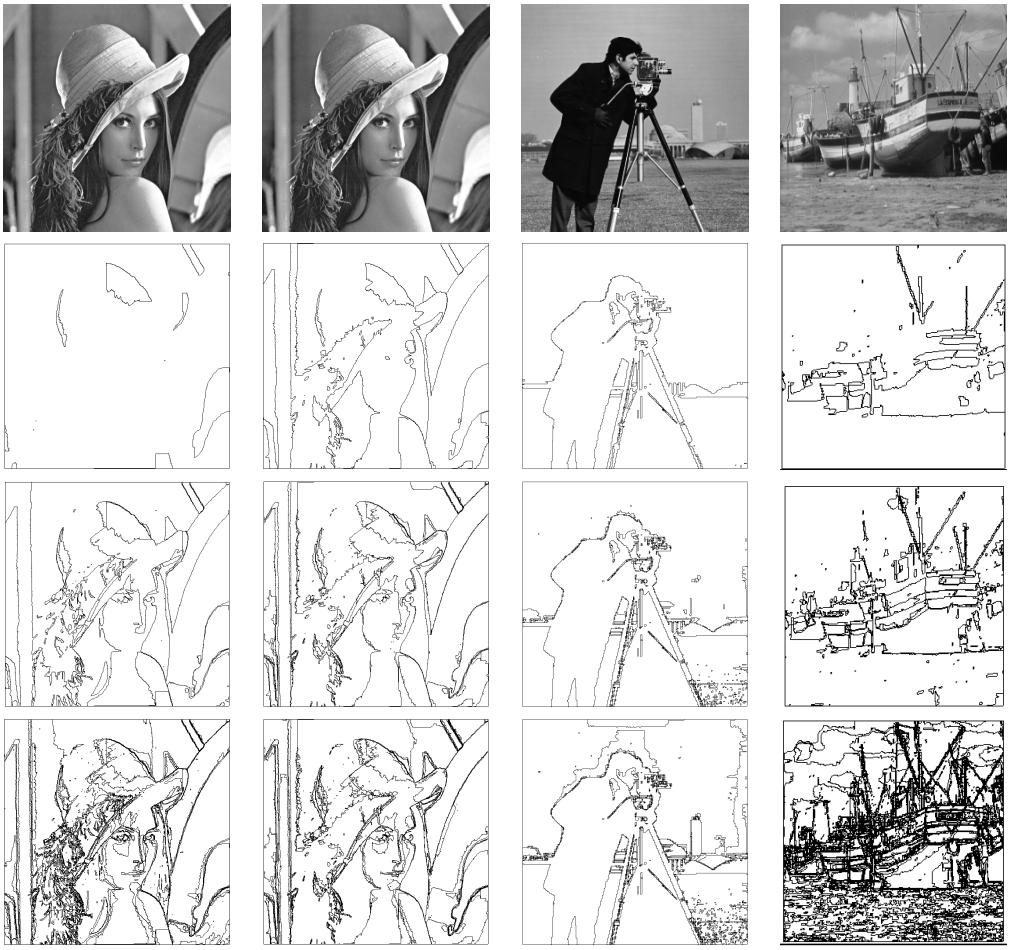
6 Results and analysis

This top-down model has been implemented in C++ and results have been computed on a personal computer with a CPU AMDX2 3800+ (2GHz) and 1GB of RAM on a Linux system.

The splitting and merging criteria used to obtain the different segmentations in Figure 10 are defined as follow: let (r, r') a couple of adjacent regions, M, m and A denoting the maximum, minimum and average gray level of r. The symbol tr denotes the threshold used by the merging criterion.

In columns (a) and (d) of Figure 10, the merging criterion is a comparison between the average gray levels of two adjacent regions: (r, r') is merged if $|A - A'| < tr$. In columns (b) and (c), tr is proportional to the standard deviation of the parent region of (r, r'): the more homogeneous a parent region is, the more our segmentation algorithm will search for fine details within its sons.

The splitting criterion avoids regions with less than ≤ 10 pixels and those which are going to be "completely" merged: r is split if $M - m > tr$. In columnms (a) and



(a) (b) (c) (d)

Figure 10: Visualization of the top-down construction process: first row is the original image followed by levels G^1, G^2 and G^3 (G^0 is a single region). Segmentations are based on user-defined thresholds (a,d) or rely on the standard deviation of the parent region (b,c).

(d), the values of tr are respectively set to 80/40/20 and 100/40/10 for the levels $G^1/G^2/G^3$ of the images Lena and Boat. Our splitting and merging operations are independent of any specific criteria: more elaborated ones, using geometrical, colorimetric or topological features, may be designed without modifying our model.

Table 2: Statistics of the top-down construction from the image of Lena (512*512) in Figure 10, column (b).

	G^1	G^2	G^3
darts	600	7 728	19 090
regions	134	1 624	3 953
total memory (KB)	306	808	1 604
splitting (s)	2.23	1.42	1.29
merging (s)	0.37	0.27	0.27
total level construction (s)	3.11	2.05	1.94

Table 2 gives the number of elements, the memory usage and processing times for each topological map composing the pyramid of the image Lena. The number of darts and regions strongly increases from a level to another as the merging threshold differentiates more regions. It directly impacts the memory size of the associated topological map. Indeed, the topology of a map requires most of the memory, except for low segmented maps where geometry could require more. The construction time of a level remains constant because, although more regions are split, they are smaller in number of pixels.

7 Conclusion

This paper defines a model of top-down hierarchical data structure based on topological maps. Topological maps are based on three models: a combinatorial map encoding multiple adjacency of regions, an explicit encoding of the geometry of the regions border and an encoding of the inclusion relationships. Such a model provides a complete description of a partition and is adapted to splitting operations. Our top-down pyramid is based on an initial topological map successively refined by splitting operations.

This structure is particularly well suited for applications in segmentation that process large images: a top-down construction scheme allows to store, at each step of the algorithm, only the currently split regions and we avoid the storage of very fine partitions (first levels of bottom-up irregular pyramids). Besides, can use global properties of upper levels to refine the segmentation in lower levels and we retrieve the dual relation between quadtrees and matrix pyramids with top-down and bottom-up approaches.

In our future work, we plan to study different encoding of the geometry and of the inclusion relationships of topological maps. We also plan to use other splitting methods such as [BMD03] and to compare our results with results obtained from other kinds of pyramids. Finally, we should define segmentation operations which fully exploit the top-down structure of the pyramid.

Acknowledgements

This research is part of the FoGrImMi project, supported by the ANR foundation under grant ANR-06-MDCA-008-01/FOGRIMMI.

References

[BCR90] M. Bister, J. Cornelis, and A. Rosenfeld. A critical view of pyramid segmentation algorithms. *Pattern Recognition Letter.*, 11(9):605–617, September 1990.

[BK03] Luc Brun and Walter Kropatsch. Combinatorial pyramids. *IEEEICIP*, 2:33–37, 2003.

[BK06] Luc Brun and Walter Kropatsch. Contains and inside relationships within combinatorial pyramids. *Pattern Recognition.*, 39(4):515–526, 2006.

[BMD03] Luc Brun, Myriam Mokhtari, and Jean Philippe Domenger. Incremental modifications on segmented image defined by discrete maps. *Journal of Visual Communication and Image Representation*, 14:251–290, 2003.

[DBF04] G. Damiand, Y. Bertrand, and C. Fiorio. Topological model for two-dimensional image representation: definition and optimal extraction algorithm. *Computer Vision and Image Understanding*, 93(2):111–154, February 2004.

[DL03] G. Damiand and P. Lienhardt. Removal and contraction for n-dimensional generalized maps. In *DGCI*, pages 408–419, Naples, Italy, November 2003.

[DRH80] R.C. Dyer, A Rosenfeld, and S Hanan. Region representation: Boundary codes from quadtrees. *ACM: Graphics and Image Processing*, 23(3):171–179, March 1980.

[GCM06] Laurent Guigues, Jean Pierre Cocquerez, and Hervé Le Men. Scale-sets image analysis. *Comput. Vision*, 68(3):289–317, 2006.

[JM92] Jean-Michel Jolion and Annick Montanvert. The adaptative pyramid: A framework for 2d image analysis. *CVGIP*, 55(3):339–348, May 1992.

[JR94] Jean-Michel Jolion and Azriel Rosenfeld. *A Pyramid Framework for Early Vision: Multiresolutional Computer Vision.* Kluwer Academic Publishers, Norwell, MA, USA, 1994.

[Lie89] P. Lienhardt. Subdivision of n-dimensional spaces and n-dimensional generalized maps. In *Symposium on Computational Geometry*, pages 228–236, Saarbrücken, Germany, 1989.

[Mee89] P. Meer. Stochastic image pyramids. *CVGIP*, 45:269–294, 1989.

[MMR91] Annick Montanvert, Peter Meer, and Azriel Rosenfeld. Hierarchical image analysis using irregular tessellations. *IEEETPAMI*, 13(4):307–316, April 1991.

[SDL06] C. Simon, G. Damiand, and P. Lienhardt. nd generalized map pyramids: definition, representations and basic operations. *Pattern Recognition*, 39(4):527–538, April 2006.