Merge-and-simplify operation for compact combinatorial pyramid definition

Guillaume Damiand, Florence Zara

Merge-and-simplify Operation for Compact Combinatorial Pyramid Definition

Guillaume Damiand[a,*], Florence Zara[b]

[a] *CNRS, LIRIS, UMR5205, Universit de Lyon, F-69622, Villeurbanne, France*
[b] *Université de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205, F-69622, Villeurbanne, France*

Abstract

Image pyramids are employed for years in digital image processing. They permit to store and use different scales/levels of details of an image. To represent all the topological information of the different levels, combinatorial pyramids have proved having many interests. But, when using an explicit representation, one drawback of this structure is the memory space required to store such a pyramid. In this paper, this drawback is solved by defining a compact version of combinatorial pyramids. This definition is based on the definition of a new operation, called "merge-and-simplify", which simultaneously merges regions and simplifies their boundaries. Our experiments show that the memory space of our solution is much smaller than the one of the original version. Moreover, the computation time of our solution is faster, because there are less levels in our pyramid than in the original one.

Keywords: Hierarchical image representation; combinatorial pyramids; 2D combinatorial map.

1. Introduction

Image analysis consists to extract meaningful information from images. An efficient approach is inspired by the psychovisual properties of focus of attention. It analyzes the content of images at different scales/levels of details. This approach requires a relevant data structure to store the different levels of information. In this context, *image pyramids* are used for years in digital image processing (Meer (1989); Bister et al. (1990); Montanvert et al. (1991); Jolion and Montanvert (1992)).

The first pyramids were defined as sequences of images with decreasing resolutions. Each new level of a pyramid was computed using a same regular grouping pattern. These pyramids are called *regular*. But the segmentation process turned out more efficient using different adaptive grouping (Bister et al. (1990)). The *irregular* pyramid was thus defined, where each level is a partition of the image pixels in *regions*. Such a pyramid requires a topological representation to describe the partitions, and basic operations to build the data structure.

Montanvert et al. (1991) proposed to code the sequence of partitions as a sequence of successive reduced graphs. Each vertex of a graph corresponds to a region, and an edge in the graph exists between two vertices when the two corresponding regions are adjacent in the image. In Kropatsch (1995); Kropatsch and Macho (1995), *dual graphs* were defined: two graphs are stored, a graph and its dual.

Later on, specific data structures have been proposed to obtain a topological representation of the space. A *2D combinatorial map* is a mathematical model of the representation of space subdivision. It is based on planar map (Edmonds (1960); Tutte (1963); Jacques (1970)). Combinatorial maps have many advantages justifying their use: they are fully ordered; they enable the representation of multi-incidence relations; they allow us many topological operations; they allow us local modifications; they are independent of the geometry, *i.e.* they can represent curved objects as well as linear objects.

For these reasons, 2D combinatorial maps, or some variants, were used as basic data-structure in several image processing methods Braquelaire and Brun (1998); Fiorio (1996); Kthe (2002); Dufourd (2007). They were also used to define irregular pyramids for image processing, called *combinatorial pyramids* (Brun and Kropatsch (2000, 2003b,a,c); Simon et al. (2005, 2006); Simon and Damiand (2006); Fourey and Brun (2009)). A combinatorial pyramid is a sequence of progressively reduced 2D combinatorial maps, plus some links between consecutive levels. To represent such a pyramid, an explicit representation stores each level as a distinct combinatorial map. The main advantage is to provide a direct access to each level of the pyramid. Its main drawback is to have an high complexity in memory space.

In this paper, our main contribution is the definition of a *compact combinatorial pyramid* that requires less memory space than previous explicit representations. Our solution is based on the definition of a new operation called *merge-and-simplify*, which is our second main contribution. This operation allows us to simultaneously merge some regions in a given combinatorial map, and simplify the region boundaries, which was not possible with original removal operations.

The outline of this paper is the following. Section 2 introduces all the basic notions used in this paper. In Section 3, we define our new operation and we show the links with the original operations. Then, this new operation is used in Section 4 to define our new compact combinatorial pyramid. The experiments presented in Section 5 show the interest of our solution: we both decrease the memory space and the computation time of the pyramid used for the image segmentation. In Section 6,

*Corresponding author: Tel.: +33-472-431-434; fax: +33-472-431-536;
Email addresses: guillaume.damiand@liris.cnrs.fr (Guillaume Damiand), florence.zara@liris.cnrs.fr (Florence Zara)

we conclude and give some future work.

2. Preliminary notions

Some preliminary notions are introduced about combinatorial maps, operations on these maps, and combinatorial pyramids. In our mathematical notation, capital letters are used for sets, and lower case letters otherwise. For a mapping f from X to Y, the notation $f(E) = \{f(e) | e \in E\}$ is used, with $E \subseteq X$. f is a *permutation* if $X = Y$; f is an *involution* if $X = Y$ and $\forall e \in X$, $f(f(e)) = e$.

2.1. 2-maps and their topological operations

The subdivision of a 2D topological space is a partition of the space into 3 subsets whose elements are *cells* of 0, 1 and 2 dimension (respectively called *vertices*, *edges* and *faces)*. *Border* relations are defined between these cells, where the border of an i-cell is a set of $(j<i)$-cells. Two cells are *incident* when one belongs to the border of the second, and two i-cells are *adjacent* if they are both incident to the same $(i-1)$-cell.

2D combinatorial maps. A combinatorial map encodes a space subdivision and all the incidence relations between the different cells of the space. Consequently, it represents the topology of this space. It is based on a unique basic type of element, called *dart*, together with relations between these darts. A 2D combinatorial map (called 2-map) is a set of *darts*, which can be seen as *half-edges*, plus two mappings defined on these darts (*cf.* Def. 1) (Lienhardt (1991); Damiand and Lienhardt (2014)). In this paper, 2-map is used without boundary. So, each edge is always described by 2 darts, and there is a specific unbounded face describing the external part of the object.

Definition 1 (2-map)**.** *A 2-dimensional combinatorial map is a triplet $M = (D, \beta_1, \beta_2)$ where: (1) D is a finite set of darts; (2) β_1 is a permutation on D; (2) β_2 is an involution on D.*

For a dart d, β_1 is the next dart after d in the same face than d. $\beta_0 = \beta_1^{-1}$ is the inverse relation: this is the dart before d in the same face. For a dart d, β_2 is the other dart of the same edge. Note that this definition of 2-map was extended to be able to represent objects with boundaries, and objects in higher dimension (Damiand and Lienhardt (2014)).

A 2-map is equivalent to the half-edge data structure (Weiler (1985); Mäntylä (1988)) or the DCEL data structure (de Berg et al. (2000)) - DCEL for Doubly Connected Edge List), where half-edges are equivalent to darts. Moreover, others equivalent solutions exist, for example the *corner table* data structure (Rossignac (2001)), where corners are equivalent to half-edges. The difference between these solutions are the naming of the elements (darts, half-edges, corners); the mappings represented (permutation around vertices or around faces); the way these mappings are stored (pointers or indices). We use the notation with darts and β (rather than α, σ and φ notations also used by many authors when dealing with planar maps) as it is the notation both used in the reference book about combinatorial maps (Damiand and Lienhardt (2014)) and in the Combinatorial maps package (Damiand (2011)) in CGAL which is the main C++ library of computational geometry.

i-cells. Thanks to the darts and the β links, a 2-map describes the subdivision of a 2D object in cells. All the cells of the object are represented by some specific subset of darts. An edge is the set of two darts linked by β_2; a face is the set of all darts obtained by using β_1, until going back to the original dart; a vertex is the set of all darts obtained by using $\beta_1 \circ \beta_2$, until going back to the original dart. Two cells C_1 and C_2 are *incident* if $C_1 \cap C_2 \neq \emptyset$, or in other words, if the two cells share at least a common dart. Two i-cells C_3 and C_4 are *adjacent* if $\exists C_5$, an $(i-1)$-cell, incident both to C_3 and C_4.

Example. Fig. 1a gives an example of 2-map. It has 22 darts (drawn by oriented segments), 15 darts (numbered). Two darts linked by β_1 are drawn consecutively (for example $\beta_1(1) = 2$), and two darts linked by β_2 are drawn in parallel, with reverse orientations, with a small gray segment over the two darts (for example $\beta_2(1) = 5$). This 2-map has 9 vertices (for example vertex $C_0(2) = \{2, 5, 15\}$, the set of all darts having this vertex as origin), 11 edges (for example edge $C_1(1) = \{1, 5\}$) and 4 faces (for example face $C_2(1) = \{1, 2, 3, 4\}$). $C_0(2)$ and $C_1(1)$ are incident because $C_0(2) \cap C_1(1) = \{5\} \neq \emptyset$. $C_2(1)$ and $C_2(5) = \{5, 6, 7, 8\}$ are adjacent because $C_1(1)$ is incident to both 2-cells.

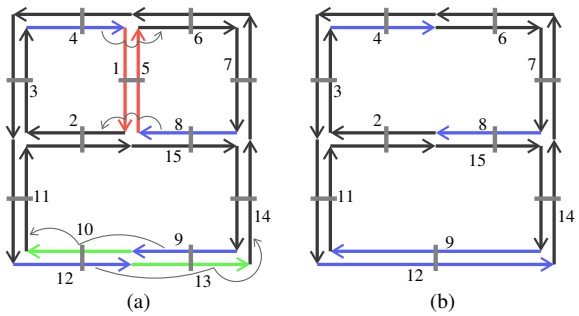


Figure 1: (a) A 2-map. (b) The 2-map obtained from (a) by the removal of edge $\{1, 5\}$ and the removal of vertex $\{10, 13\}$.

Dual of a 2-map. Given a 2-map $M = (D, \beta_1, \beta_2)$, its *dual* is the 2-map $\overline{M} = (D, \beta_1 \circ \beta_2, \beta_2)$. Vertices of M are faces in $\overline{M}$ (and reciprocally); while edges stay unchanged in M and $\overline{M}$.

Removal operations. **We remind that cells are sets of darts in 2-maps.** This is one interesting specificity of 2-maps. Thus, to define an operation that modifies a 2-map, only darts and β links have to be modified and the cells are automatically updated thanks to their definition.

It is possible to simplify a 2-map by removing some of its edges or some of its vertices. When an i-cell C is removed, this merges the two $(i+1)$-cells incident to C in one $(i+1)$-cell. The edge removal operation is given in Def. 2.

Definition 2 (1-cell removal)**.** *Let $M = (D, \beta_1, \beta_2)$ be a 2-map and E be one of its edge*[1]*. Let $D^N = \beta_0(E) \setminus E$. The 2-map resulting from the 1-removal of E in M is $R_1(M, E) = (D', \beta'_1, \beta'_2)$, defined by:*

[1] By definition of edges in 2-maps, E is a set of darts.

- $D' = D \setminus E$;
- $\forall d \in D', \beta'_2(d) = \beta_2(d)$;
- $\forall d \in D' \setminus D^N, \beta'_1(d) = \beta_1(d)$;
- $\forall d \in D^N, \beta'_1(d) = (\beta_1 \circ \beta_2)^k \circ \beta_1(d)$,
  k is the smaller positive integer s.t. $(\beta_1 \circ \beta_2)^k \circ \beta_1(d) \notin E$.

The main principle of this definition is to remove the 2 darts of the removed edge E, and to redefine the β_1 links of darts neighbors of E by β_0. It is performed by jumping over E and turning around its extremity, until obtaining a non removed dart.

Vertex removal operation is based on a same principle. But contrary to edges, it is not possible to remove any vertex. Indeed, it must have at most two edges incident to the vertex, otherwise it is not possible to decide which edges to merge after the removal. Such a vertex is called *removable*.

For a removable vertex, the removal operation is given in Def. 3. In this definition, both β_1 and β_2 links of darts neighbors of the removed vertex have to be redefined.

Definition 3 (0-cell removal)**.** *Let $M = (D, \beta_1, \beta_2)$ be a 2-map and V be one of its removable vertex*[2]*. The 2-map resulting from the 0-removal of V in M is $R_0(M, V) = (D', \beta'_1, \beta'_2)$, defined by:*

- $D' = D \setminus V$;
- $\forall j \in \{1, 2\}, \forall d \in D'$:
  - *if* $d \notin \beta_j^{-1}(V) \setminus V$, $\beta'_j(d) = \beta_j(d)$;
  - *otherwise,* $\beta'_j(d) = \beta_j \circ (\beta_1)^k(d)$,
    k is the smaller positive integer s.t. $\beta_j \circ (\beta_1)^k(d) \notin V$.

Fig. 1 illustrates these operations. Starting from the 2-map given in Fig. 1a, edge $\{1, 5\}$ (red darts) and vertex $\{10, 13\}$ (green darts) have been removed. The 2-map depicted in Fig. 1b was obtained, where β_1 of darts $4, 8, 9$ and 12 were modified (blue darts), as well as β_2 of darts 9 and 12. These β_2 links must be updated because darts 10 and 13 were removed by the vertex removal operation, and these darts were linked respectively with darts 9 and 12 by β_2 links in the original map.

Removal operations are extended directly to set of cells: it is possible to remove simultaneously any set of removable vertices and edges, when the set of darts of these cells are disjointed. It is also possible to simplify a 2-map by using the contraction operation, which is the dual of the removal operation. We do not use this operation in this work (Brun and Kropatsch (1999, 2003b); Damiand and Lienhardt (2003, 2014)).

Reducible vertices. If a vertex incident to only one edge (a loop, separating two faces) is removed, this also removes totally the edge and the 2 faces. To avoid such a removal, a property on the vertex to remove can be added: it must be 0-removable and incident to exactly two different edges. Such a vertex is called *0-reducible*.

[2]By definition of vertices in 2-maps, V is a set of darts.

2.2. Combinatorial pyramids

A combinatorial pyramid is a sequence of 2D combinatorial maps $(M_0, \ldots, M_l)$ where the 2-map M_i is called level i of the pyramid; level 0 is called the *basis*; level l the *top*; l is the *height* of the pyramid. Each map (except the basis) is a simplification of the 2-map in the previous level (Brun and Kropatsch (2000, 2003a,c); Simon et al. (2006)). According to the application and the semantic associated with the 2-maps, it is possible to define each level M_i in different ways, but the computation of a new level is usually done by using only one type of operation (removal or contraction of only one type of cells). Usually, the basis describes all pixels, then one level corresponds to the merging of some regions, and a second level corresponds to the simplification of the region boundaries.

For example in Brun and Kropatsch (2003a), the vertices of the basis of the pyramid correspond to the pixels of the image. Then, merging two adjacent regions is achieved by contracting the corresponding edge; and simplify the region boundaries is done by removing edges with some specific properties (double edges and empty-self loops).

Dual representation is used in Simon and Damiand (2006). Pixels of the image are represented by faces in the basis of the pyramid. Then, regions are merged by removing edges; and simplify the region boundaries is done by removing vertices.

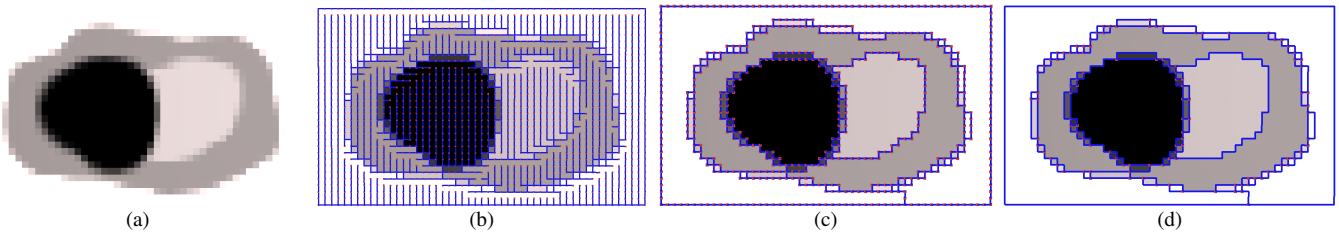
When a new level is added in a pyramid, the 2-map has to be ever *connected* in order to have a correct definition of cells. Indeed, imagine a 2-map made of two squares, one inside the other. This 2-map has two connected components. The geometrical object represented is a square with an hole: that is, an object with three faces (the square, the hole and the unbounded face). But the 2-map has 4 faces, two for each square, which is not correct. Adding an edge between the two squares keep the 2-map connected and give us the correct number of faces.

When a set of edges E in a 2-map M is removed, the map stays connected if E is a *forest*[3] in the dual 2-map $\overline{M}$ (Brun and Kropatsch (2003b)). For a combinatorial pyramid where each level is connected, it is possible to retrieve all the i-cells in a level i that are merged in a given i-cell in a level $j > i$ (*cf.* Brun and Kropatsch (2003c); Simon et al. (2005)).

Encoding combinatorial pyramids. Several possibilities exist to encode more or less explicitly a combinatorial pyramid according to the expected space/time complexity (Simon et al. (2006)). In an *explicit* representation, each level of the pyramid is represented in a distinct 2-map; each dart stores 2 additional pointers to the corresponding darts in the previous and the next levels. In an *implicit* representation, only the basis of the pyramid is represented as a 2-map and additional information allows us to retrieve removed and contracted cells for each level.

Each representation has its own advantages and drawbacks. The implicit representation is the best one in term of space complexity. But it is the worst representation for time complexity since the traversal of any level needs to iterate through

[3]A forest is a set of trees, thus a set of edges without cycle, but not necessarily connected.



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

Figure 2: (a) An image. (b) Result obtained after the removal of some edges in the 2-map describing all the pixels of this image, these edges being a forest in the dual 2-map. (c) Result obtained after the removal of all dangling edges of (b). (d) Result obtained after the removal of all 0-reducible vertices.

darts of the basis of the pyramid, jumping over all the darts removed/contracted. The explicit representation is the one which requires the most memory space since each level is represented plus the links between the level. But this is the best representation for operations since it provides a direct access to each level, and operations can be applied directly to each level without any dependencies with other levels.

In our work, an explicit representation is used, since our application needs fast access to each level of the pyramid. In this framework, our main contribution is the definition of a new operation, called *merge-and-simplify*, which allows us to decrease the height of a combinatorial pyramid by avoiding the need to use different levels for merging the regions and for simplifying the region boundaries.

2.3. 2-maps versus graphs

Many image processing papers that consider hierarchical partitions use trees or graphs to describe the hierarchy (*cf.* for example Montanvert et al. (1991); Jolion and Montanvert (1992); Kropatsch (1995); Arbelaez et al. (2011); Cousty et al. (2018)). But a graph is not able to describe multiple adjacencies; it does not keep the order of the edges around a vertex; it does not represent the faces but only vertices and edges; and it is not unique as two different images could be represented by the same graph. Some solutions were proposed to solve some of these drawbacks, like dual-graphs (Kropatsch and Macho (1995)). But they are not topological representative of the images as it is possible to have two topological different images with the same dual graphs. Moreover, to maintain the correspondence between the 2 graphs, each operation performed has to be applied twice: once to the primal graph and another to the dual one.

Contrary to solutions based on graphs, a 2D combinatorial map is a mathematical model of the representation of space subdivision (Damiand and Lienhardt (2014)): it allows us to fully define the topology of a planar partition by describing all the cells and all the incidence and adjacency relations between the cells, allowing multiple adjacencies relations. Moreover, several operations exist on 2-maps allowing to iterate through the elements of a 2-map, or to modify the subdivision. All of these reasons make 2-maps an efficient model to build, to represent and to update an image partition in regions, as shown by numerous work based on this model (for example Braquelaire and Brun (1998); Fiorio (1996); Kthe (2002); Dufourd (2007); Brun and Kropatsch (2000, 2003b,a,c); Simon et al. (2005, 2006); Simon and Damiand (2006); Fourey and Brun (2009)).

The choice of one (hierarchical) data structure to describe partitions depends on the applications. If you only need to deal with regions and some criteria (like size, shape, color, etc.), a list of regions with attributes is enough. If one algorithm requires to access to regions adjacent to a given one, and needs to update these adjacency relations, a region adjacency graph will probably be a good choice. Whereas, if you need to describe the full topology of the partition, with several operations to update your model providing topological control on the modifications, 2-maps is probably the best suitable data-structure.

3. Our merge-and-simplify operation

In this section, a new operation on 2-maps called *merge-and-simplify* is defined.

3.1. Original operation

When faces are merged in a 2-map, for example during an image segmentation process based on region growing, edges separating faces to merge must be 1-removed. As seen in the previous section, the disconnection of the 2-map in several connected components can be avoided by imposing that the set of removed edges is a forest in the dual 2-map.

This is a sufficient condition but not a required one. Indeed, when two regions to merge are adjacent along at least two consecutive edges, only one out of the two edges will be removed. Effectively, adding the two edges in the set of removed edges will create a cycle in the dual 2-map. Due to this condition, there are many edges inside faces resulting of the merging that must be removed since they do not describe any region boundary, as illustrated in Fig. 2b. This can be achieved by iteratively removing all the dangling edges (Simon and Damiand (2006)). At the end, the 2-map obtained in Fig. 2c contains no more dangling edges: each edge either belongs to the boundary of a face, or is a bridge necessary to keep the 2-map connected. But now this map contains too many vertices that must be removed. This is the last step of the simplification which consists to remove iteratively each 0-reducible vertex. The 2-map shown in Fig. 2d is obtained which can no more be simplified.

3.2. New operation: merge-and-simplify

Our idea is to define a new operation, called *merged-and-simplify*, which applies directly both the region merging and the two simplification steps (of dangling edges and of 0-reducible vertices). This new operation (given in Def. 4) takes as input a 2-map plus a set of edges. The dangling edges and the 0-reducible vertices to remove are automatically computed by it.

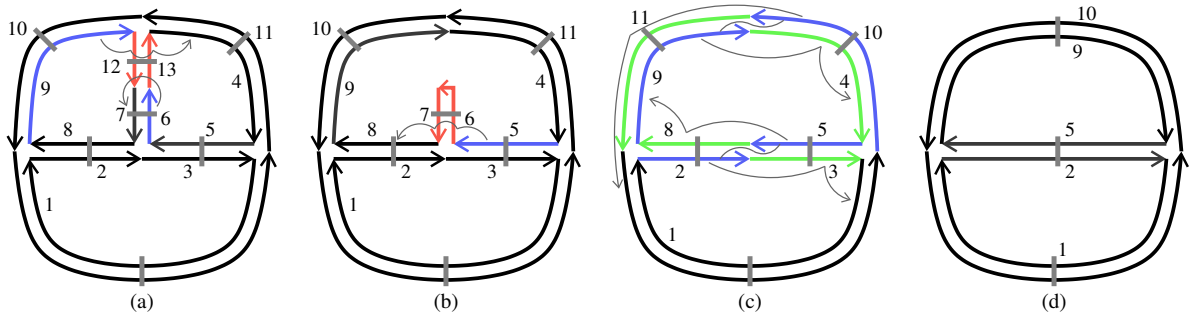


Figure 3: Illustration of the different steps to simplify a 2-map when using the original removal operation. (a) Initial configuration, one edge $\{12, 13\}$ is marked to remove. (b) Result of the removal of edge $\{12, 13\}$. Now edge $\{6, 7\}$ is dangling and thus marked to remove. (c) After the removal of edge $\{6, 7\}$, the two vertices $\{3, 8\}$ and $\{4, 11\}$ are 0-reducible. (d) Result of the removal of the two vertices $\{3, 8\}$ and $\{4, 11\}$.

Definition 4 (Merge-and-simplify)**.** *Let $M = (D, \beta_1, \beta_2)$ be a 2-map and K_1 be a set of darts, union of edges*[4] *of M. Let us define:*

- $K_1 = K_1 \cup E_0 \ldots \cup E_e$*, where each E_i is a dangling edge in $R_1(M, K_1 \cup E_0 \ldots \cup E_{i-1})$;*
  and there is no dangling edge in $R_1(M, K_1 \cup E_0 \ldots \cup E_e)$;
- $K_0 = V_0 \cup \ldots \cup V_v$*, where each V_i is a 0-reducible vertex in $R_0(R_1(M, K_1), V_0 \cup \ldots \cup V_{i-1})$; and there is no 0-reducible vertex in $R_0(R_1(M, K_1), V_0 \cup \ldots \cup V_v)$;*
- $N_1 = \beta_0(K_0 \cup K_1) \setminus (K_0 \cup K_1)$*;*
- $N_2 = \beta_2(K_0) \setminus (K_0 \cup K_1)$*.*

The 2-map resulting from the merge-and-simplify of M by K_1 is $MS(M, K_1) = (D', \beta'_1, \beta'_2)$, defined by:

- $D' = D \setminus (K_0 \cup K_1)$*;*
- $\forall d \in D' \setminus N_1$*,* $\beta'_1(d) = \beta_1(d)$*;*
- $\forall d \in N_1$*,* $\beta'_1(d) = d' = (f_k \circ \ldots \circ f_0 \circ \beta_1)(d)$*,*
  where $f_i = \beta_1$ if $(f_{i-1} \circ \ldots \circ f_0 \circ \beta_1)(d) \in K_0$,
  or $f_i = \beta_1 \circ \beta_2$ if $(f_{i-1} \circ \ldots \circ f_0 \circ \beta_1)(d) \in K_1$,
  and k being the smaller positive integer s.t. $d' \in D'$;
- $\forall d \in D' \setminus N_2$*,* $\beta'_2(d) = \beta_2(d)$*;*
- $\forall d \in N_2$*,* $\beta'_2(d) = \beta_2 \circ \beta_0 \circ \beta'_1(d)$*.*

The first part of the definition adds in K_1 all the dangling edges. Note that edge E_i is dangling in $R_1(M, K_1 \cup E_0 \ldots \cup E_{i-1})$, *i.e.* in the 2-map obtained by removing all edges in K_1 plus all dangling edges $E_0, \ldots, E_{i-1}$. Such a definition is required because edge E_i is possibly not dangling in the map obtained by removing all edges in K_1, but can become dangling after the removal of some dangling edges.

In the second part, K_0 is defined as the set of all the 0-reducible vertices. Similarly than for K_1, vertex V_i is added if it is 0-reducible in the 2-map where all the edges in K_1 are removed and all the previous 0-reducible vertices are removed.

N_1 is the set of non removed darts, neighbors by β_0 of one removed vertex or removed edge. These darts will have their β_1 modified by the operation. N_2 is the set of non removed darts neighbors of one remove vertex by β_2, having their β_2 modified.

Then, similarly than the removal operation, the merge-and-simplify operation defines a new 2-map (D', β'_1, β'_2) which redefines β'_1 and/or β'_2 for darts concerned by the operation. The difference is the paths followed for these redefinitions. Contrary to the original operation, a removed vertex can now be traversed starting from a neighbor dart of a removed edge. For this reason, there are 2 cases in the path traversal, depending if the last dart belongs to a 0-removed or a 1-removed cell. In each case, the definition of the original operation is used to follow correctly the path of removed darts: using either β_1 to jump over removed vertices, or $\beta_1 \circ \beta_2$ to turn over removed edges.

3.3. Comparison between original and new operations

Let us compare the new merge-and-simplify operation with the original one, which starts first to remove edges, then to remove vertices. In the example given in Fig. 3, we start from an initial 2-map (Fig. 3a) where an edge is marked to remove, by an external process (for example an image segmentation criterion). After the removal of this edge, the 2-map shown in Fig. 3b is obtained which contains one dangling edge and which is removed. The resulting 2-map (Fig. 3c) contains now two 0-reducible vertices which are marked to remove. Fig. 3d shows the final result obtained after all the simplifications. To store these different steps into an explicit pyramid, 4 different levels are needed, 42 darts in total, and 11 β links are redefined.

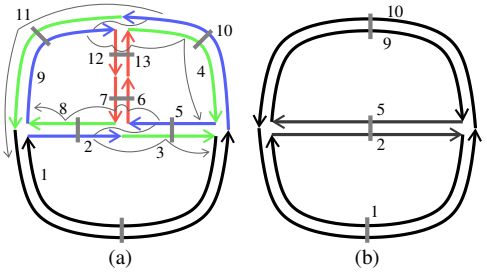


Figure 4: Same simplification than the example given in Fig. 3, but using now the merge-and-simplify operation. (a) Initial configuration: edge $\{12, 13\} \in K_1$, is the only input of the operation. Edge $\{6, 7\}$ is dangling in $M_{R_1}(K_1)$ and is thus added in K_1. Vertices $\{3, 8\}$ and $\{4, 11\}$ are 0-reducible in $M_{R_1}(K_1)$ and are thus added in K_0. (d) Result of the merge-and-simplify operation, where edges in K_1 then vertices in K_0 are removed.

[4] By definition of edges in 2-maps, each edge is a set of darts.

Let us reconsider the same example but using now the new merge-and-simplify operation. As shown in Fig. 4, the starting configuration is the same: an initial 2-map with only one edge marked to remove. But contrary to the previous simplification, dangling edge and 0-reducible vertices are directly marked to remove in the initial 2-map (Fig. 4a). After applying the merge-and-simplify operation,the simplified 2-map given in Fig. 4b is directly obtained in one step.

In this example, let us consider the path of darts traversed for the definition of $\beta'_1(5)$. $5 \in N_1$ (this is a neighbor dart of a removed edge). $f_0 = \beta_1 \circ \beta_2$ because $\beta_1(5) \in K_1$; $f_1 = \beta_1$ because $8 = \beta_1 \circ \beta_2 \circ \beta_1 \in K_1$, and $k = 1$ because $9 = \beta_1 \circ \beta_1 \circ \beta_2 \circ \beta_1 \in D'$. Thus $\beta'_1(5) = 9$ is defined. To define $\beta'_2(5)$, the definition of $\beta'_1(5)$ is reused (in order to avoid to redefine the path in Def. 4), go backward by β_0 to retrieve the last dart of the path in the same edge, then use β_2. For dart 5, $\beta'_1(5) = 9$, $\beta_0(9) = 8$ and $\beta_2(8) = 2$: $\beta'_2(5) = 2$ is defined (and reciprocally).

Thanks to this new operation, this simplification can be stored into an explicit pyramid which only has 2 different levels with 20 darts in total, and 8 β links (*versus* 4 levels, 42 darts and 11 β for the original version) are redefined.

Given a dart $d \in N_1$, it is straightforward to prove that $\forall i$, $0 \leq i < k$, $(f_i \circ \ldots \circ f_0 \circ \beta_1)(d) \in (K_0 \cup K_1)$. Indeed, if we suppose $\exists i$, $0 \leq i < k$, $d_i = (f_i \circ \ldots \circ f_0 \circ \beta_1)(d) \notin (K_0 \cup K_1)$, then $d_i \in D' = \backslash(K_0 \cup K_1)$, which contradicts k being the smaller positive integer s.t. $d' \in D'$. In other words, all darts traversed during the definition of β' are removed by the operation, except the first and the last one.

3.4. Link with sequence of removal operations

Let $M = (D, \beta_1, \beta_2)$ be a 2-map and K_1 be a set of edges of M. We can prove that $M' = MS(M, K_1) = (D', \beta'_1, \beta'_2)$ is isomorphic to $M'' = R_0(R_1(R_1(M, K_1), E), V)$, with $E = E_0 \cup \ldots \cup E_e$, where each E_i is a dangling edge in $R_1(M, K_1 \cup E_0 \ldots \cup E_{i-1})$; and there is no dangling edge in $R_1(M, K_1 \cup E_0 \ldots \cup E_e)$; and $V = V_0 \cup \ldots \cup V_v$, where each V_i is a 0-reducible vertex in $R_0(R_1(M, K_1), V_0 \cup \ldots \cup V_{i-1})$.

In other words, the same result is obtained by using the merge-and-simplify operation than using the sequence of 3 operations: (1) edge removal of edges in K_1; (2) edge removal of all dangling edges; (3) vertex removal of all 0-reducible vertices. This property ensures us the validity of our new operation. The principle of the proof of equivalence is the following. Starting from the definition of removal operations, which redefine β links, these definitions can be re-written by considering all the different cases of darts, depending if they belong to a vertex or an edge removed. At the end, the redefinition done when using the new operation is the same that the one obtained by applying successively the three simplifications.

3.5. Merge-and-simplify algorithm

Given a 2-map M and a set of darts K_1 (union of edges), the merge-and-simplify operation produces $MS(M, K_1)$. Let see one algorihm to compute this 2-map from M and K_1.

A first naive solution is to copy M into M', then to progressively simplify M' in 3 steps: (1) remove all edges in K_1; (2) remove all dangling edges; (3) remove all 0-reducible vertices. Hopefully, it is possible to build $MS(M, K_1)$ directly in linear time in number of darts of M, without having to start to copy M requiring thus less memory allocation/de-allocation.

The principle of our method is (i) to mark all edges and all vertices to remove directly on M (thanks Algorithm 1); (ii) to build directly $MS(M, K_1)$ by copying only non marked darts.

Algorithm 1: Merge-and-simplify: mark darts.

Input: A 2-map M;
A set of darts K_1, union of edges.
Output: $MS(M, K_1)$

```
1 mark all darts in K_1 by m_1;
2 foreach dart d of M do
3     while c_1(d) is dangling, ignoring all darts marked by m_1 do
4         mark c_1(d) by m_1;
5         d ← one non marked dart adjacent to c_1(d);
6 foreach dart d of M do
7     if c_0(d) is 0-reducible, ignoring all darts marked by m_0 or by m_1 then
8         mark c_0(d) by m_0;
```

Algorithm 1 starts by marking all given edges (line 1). Then, an iteration through each dart d is performed. A test is done to know if $c_1(d)$, the edge containing d, is dangling in the 2-map obtained from M when removing all marked edges only by ignoring all darts marked by m_1. When $c_1(d)$ is dangling, d is removed to the dart of an edge adjacent to $c_1(d)$ (line 5). Indeed, removing $c_1(d)$ can transform the adjacent edge into a dangling edge. Similarly for vertices, a test is performed to know if $c_0(d)$ is 0-reducible in the 2-map obtained from M when removing all marked edges and all marked vertices by ignoring all darts marked by m_0 or m_1.

After the marking of removed darts, non marked darts can be copied, and Def. 4 can be used to link these darts together, by following the paths given in the definition.

4. Our compact combinatorial pyramids

The new merge-and-simplify operation enables a direct definition of *compact combinatorial pyramid*.

Definition 5 (Compact combinatorial pyramid)**.** *A 2-dimensional compact combinatorial pyramid is a sequence of 2-maps $(M_0, \ldots, M_l)$ where: $\forall i$, $1 \leq i \leq l$, $M_i = MS(M_{i-1}, K_i)$, with K_i a set of darts in M_{i-1}, being an union of edges forming a forest in $\overline{M_{i-1}}$.*

Let us compare a compact combinatorial pyramid P with a classical version P', which is defined by using only removal operations. In this case, each level M_i in P ($1 \leq i \leq l$) corresponds to 3 levels in P': M'_j, M'_{j+1}, M'_{j+2}, with $j = 3(i-1)+1$. $M'_j = R_1(M'_{j-1}, K_i)$ is the level where given edges are removed. $M'_{j+1} = R_1(M'_j, E_{i0} \cup \ldots \cup E_{il_i})$, where each E_{io} is a dangling edge in $R_1(M'_j, E_{i0} \cup \ldots \cup$

$E_{i(o-1)}$). $M'_{j+2} = R_0(M'_{j+1}, V_{i0} \cup \ldots \cup V_{im_i})$, where each V_{io} is a 0-reducible vertex in $R_0(M'_{j+1}, V_{i0} \cup \ldots \cup V_{i(o-1)})$.

Three different levels are needed to (1) remove the given edges which are a forest in the dual 2-map; (2) remove all the dangling edges; (3) remove all the 0-reducible vertices. (2) and (3) can not be applied simultaneously by the classical removal operation because the set of concerned darts are not independent. (1) and (2) can be regrouped in a same level because they both use edge removal. In this case, only 2 levels is needed in P' for each level in P (except for the base level).

With the first solution, the height of P' is three times the height of P which is our compact version. With the second solution, the height of P' is twice the height of P.

Reducing the height of the pyramid is interesting for two reasons: (1) it decreases the memory space required to represent the pyramid when using an explicit representation, since they are less darts to represent; (2) it decreases the time complexity of operations that traverses the pyramid, since this complexity is often based on the number of levels of the pyramid traversed.

It is important to remind that information described by the compact combinatorial pyramid is the same than the one described by the original version. Indeed, the meaningful levels are the ones without dangling edges, nor 0-reducible vertices, and these levels are the same in both pyramid versions.

5. Experiments

All experiments were run on an Intel®i7-4790 CPU, 4 cores @ 3.60GHz with 32 GB RAM. Both original and compact versions of the combinatorial pyramids have been implemented by using the CGAL implementation of combinatorial maps (Damiand (2011)). Fig. 5 shows two images used for the experiments (images from Unsplash (`https://unsplash.com/`), taken *resp.* by C. Hu, D. Dobrila, J. Alexander, K. Toth, M. Asthoff and T. Naccarato). The sizes of images used are 1834×2000 (Img1), 2200×1313 (Img2), 3000×2000 (Img3), 3456×5184 (Img4), 5616×3744 (Img5) and 1280×853 (Img6).

Figure 5: Two images used for the experimentations (Left) Img1. (Right) Img2.

A combinatorial pyramid was built, starting from a 2-map that represents all image pixels as basis. In this level each region has one pixel. Then a new level was added by merging all pair of adjacent regions such that the difference between the two regions mean gray level is smaller than a given threshold τ. We started with $\tau = 5$, and multiplied it by 2 for each new segmentation level. The process was ended when the top level of the pyramid has only 2 regions (one for the image pixels, the second one corresponding to the unbounded face).

Fig. 6 presents 2 levels obtained for Img3. Fig. 7 shows the sum of number of darts of all levels, except the basis, for the pyramids computed from the 6 images. Each point represents one image: its x-coordinate is the number of darts of the original pyramid and its y-coordinate is the number of darts of the compact version. The fitting line has a slope of 0.34: the number of darts used by a compact pyramid can be approximated by 0.34 times the number of darts of the corresponding original pyramid. Moreover, Table 1 gives the number of darts used by the compact combinatorial pyramid in percentage of the number of darts of the classical one: in average, a compact combinatorial pyramid uses only 32.2% of darts of the corresponding classical pyramid, which is coherent with the slope of the fitting line obtained in Fig. 7. In these results, the number of darts of the basis were not summed up as it has a specific structure: this is a set of regular squares glued together (one square per pixel). Due to this specificity, it is inefficient to represent the basis using a classical data-structure for 2-maps (with darts and pointers): it is only implicitly represented.

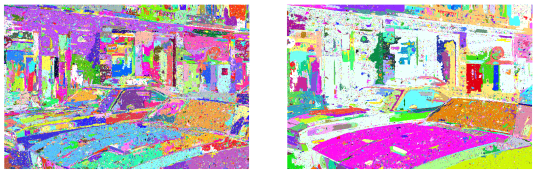


Figure 6: Two levels of the compact pyramid computed from Img3 (one random color per region). (Left) level 2. (Right) level 3 with less regions.

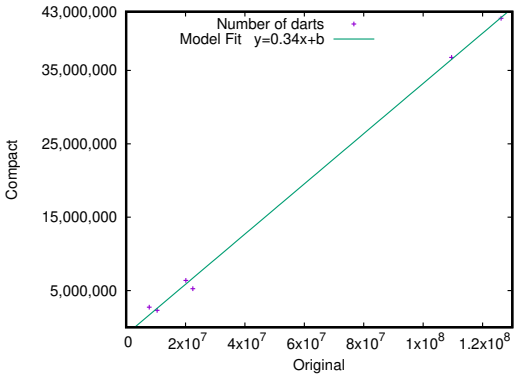


Figure 7: Number of darts (without basis) of the different pyramids. A point represents one image: its x-coordinate (*resp.* y-coordinate) is the number of darts of the original (*resp.* compact) pyramid.

Table 1: Number of darts of the compact combinatorial pyramid in percentage of the number of darts of the classical one.

Img1	Img2	Img3	Img4	Img5	Img6	Average
31.7%	22.0%	23.4%	33.6%	33.4%	35.1%	32.2%

Table 2 gives the computation time (in seconds) to build the combinatorial pyramids for the 6 images for both versions. These computation were quite good, taking into account the big size of images. This table also gives the time used by the construction of the compact version in percentage of the time spent by the classical version. In average, the compact method uses 64.6% of the time of the classical construction method. Fig. 8

Table 2: User time in seconds for constructing (a) the classical or (b) the compact combinatorial pyramid. (c) Time used by the construction of the compact version in percentage of the time spent by the classical version.

	Img1	Img2	Img3	Img4	Img5	Img6
(a)	47.3s	29.9s	69.6s	303.8s	351.4s	14.1s
(b)	30.4s	19.8s	45.9s	189.7s	226.6s	9.1s
(c)	64.3%	66.2%	65.9%	62.4%	64.5%	64.6%

shows these computation times using the same principle that the one used in Fig. 7 for the number of darts. Each point represents one image: its x-coordinate (*resp.* y-coordinate) is the computation time to construct the original (*resp.* compact) pyramid. The fitting line has a slope of 0.63: the computation time can be approximated to build a compact pyramid by 0.63 times the computation time to build the corresponding original pyramid.

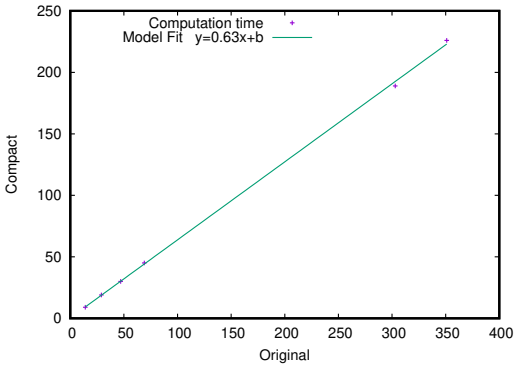


Figure 8: Computation time to build the different pyramids. A point represents one image: its x-coordinate (*resp.* y-coordinate) is the computation time to construct the original (*resp.* compact) pyramid.

Our experiments show that using the compact version allows us to decrease the number of darts of about 64% and decrease also the computation time to build the pyramid of about 27%, without introducing any drawback. This shows the interest of using the compact version instead of the original one.

6. Conclusion

In this paper, a new operation on 2-maps is defined, merge-and-simplify, which allows us to merge some faces then to simplify the dangling edges and the 0-reducible vertices in a single operation. This operation is equivalent to apply successively three times the original removal operations. Our new operation can be used in order to define compact combinatorial pyramids which decrease the height of previous combinatorial pyramids. As illustrated in our experiments, this decreases the memory space used to store such a pyramid in an explicit representation, and it decreases also the computation time of the pyramid.

In our future work, we are going to define *connecting walks*, *connecting dart sequences* and *receptive fields* with our new framework: the three central notions when we want to retrieve information in a given level based on information on previous levels. We also would like to extend the merge-and-simplify operation in higher dimension.

References

Arbelaez, P., Maire, M., Fowlkes, C., Malik, J., 2011. Contour detection and hierarchical image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 33, 898–916.

de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O., 2000. Computational Geometry: Algorithms and Applications. Springer-Verlag.

Bister, M., Cornelis, J., Rosenfeld, A., 1990. A critical view of pyramid segmentation algorithms. Pattern Recognition Letter. 11, 605–617.

Braquelaire, J.P., Brun, L., 1998. Image segmentation with topological maps and inter-pixel representation. Journal of Visual Communication and Image representation 9, 62–79.

Brun, L., Kropatsch, W., 1999. Dual contraction of combinatorial maps, in: 2^{nd} IAPR-TC-15 Workshop on Graph-based Representations, Österreichische Computer Gesellschaft, Haindorf, Austria. pp. 145–154.

Brun, L., Kropatsch, W., 2000. Irregular pyramids with combinatorial maps, in: Amin, A., Ferri, F.J., Pudil, P., I nesta, F.J. (Eds.), Advances in Pattern Recognition, Joint IAPR International Workshops SSPR'2000 and SPR'2000, Springer, Berlin Heidelberg, New York, Alicante, Spain. pp. 256–265.

Brun, L., Kropatsch, W., 2003a. Combinatorial pyramids, in: Suvisoft (Ed.), IEEE International conference on Image Processing (ICIP), Barcelona. pp. 33–37.

Brun, L., Kropatsch, W., 2003b. Contraction kernels and combinatorial maps. Pattern Recognition Letters 24, 1051–1057.

Brun, L., Kropatsch, W., 2003c. Receptive fields within the combinatorial pyramid framework. Graphical Models 65, 23 – 42. Special Issue: Discrete Topology and Geometry for Image and Object Representation.

Cousty, J., Najman, L., Kenmochi, Y., Guimarães, S., 2018. Hierarchical segmentations with graphs: Quasi-flat zones, minimum spanning trees, and saliency maps. Journal of Mathematical Imaging and Vision 60, 479–502.

Damiand, G., 2011. Combinatorial maps, in: CGAL User and Reference Manual. 3.9 ed. `http://www.cgal.org/Pkg/CombinatorialMaps`.

Damiand, G., Lienhardt, P., 2003. Removal and contraction for n-dimensional generalized maps, in: Proc. of 11th International Conference on Discrete Geometry for Computer Imagery, Springer Berlin/Heidelberg, Naples, Italy. pp. 408–419.

Damiand, G., Lienhardt, P., 2014. Combinatorial Maps: Efficient Data Structures for Computer Graphics and Image Processing. A K Peters/CRC Press.

Dufourd, J.F., 2007. Design and formal proof of a new optimal image segmentation program with hypermaps. Pattern Recogn. 40, 2974–2993.

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

Fiorio, C., 1996. A topologically consistent representation for image analysis: the frontiers topological graph, in: Proc. of International Conference on Discrete Geometry for Computer Imagery, Lyon, France. pp. 151–162.

Fourey, S., Brun, L., 2009. A first step toward combinatorial pyramids in n-D spaces, in: Graph-based Representations in Pattern Recognition, Springer, Venice, Italy. pp. 304–313.

Jacques, A., 1970. Constellations et graphes topologiques, in: Combinatorial Theory and Applications, pp. 657–673.

Jolion, J.M., Montanvert, A., 1992. The adaptive pyramid: A framework for 2d image analysis. Computer Vision, Graphics and Image Processing 55, 339–348.

Kropatsch, W., 1995. Building irregular pyramids by dual-graph contraction. Vision, Image and Signal Processing 142, 366–374.

Kropatsch, W., Macho, H., 1995. Finding the structure of connected components using dual irregular pyramids, in: Proc. of International Conference on Discrete Geometry for Computer Imagery, pp. 147–158, *invited lecture*.

Kthe, U., 2002. Xpmaps and topological segmentation - a unified approach to finite topologies in the plane, in: Proc. of International Conference on Discrete Geometry for Computer Imagery, Bordeaux, France. pp. 327–350.

Lienhardt, P., 1991. Topological models for boundary representation: a comparison with n-dimensional generalized maps. Computer Aided Design 23, 59–82.

Mäntylä, M., 1988. An Introduction to Solid Modeling. Computer Science Press.

Meer, P., 1989. Stochastic image pyramids. CVGIP 45, 269–294.

Montanvert, A., Meer, P., Rosenfeld, A., 1991. Hierarchical image analysis using irregular tessellations. IEEE Transactions on Pattern Analysis and Machine Intelligence 13, 307–316.

Rossignac, J., 2001. 3D compression made simple: Edgebreaker with zipandwrap on a corner-table, in: Proc. of International Conference on Shape Modeling and Applications, pp. 278–283.

Simon, C., Damiand, G., 2006. Generalized map pyramid for multi-level 3d image segmentation, in: Proc. of 13th International Conference on Discrete Geometry for Computer Imagery, Springer Berlin/Heidelberg, Szeged, Hungary. pp. 530–541.

Simon, C., Damiand, G., Lienhardt, P., 2005. Receptive fields for generalized map pyramids: The notion of generalized orbit, in: Proc. of 12th International Conference on Discrete Geometry for Computer Imagery, Springer Berlin/Heidelberg, Poitiers, France. pp. 56–67.

Simon, C., Damiand, G., Lienhardt, P., 2006. nd generalized map pyramids: Definition, representations and basic operations. Pattern Recognition (PR) 39, 527–538.

Tutte, W., 1963. A census of planar maps. Canad. J. Math. 15, 249–271.

Weiler, K., 1985. Edge-based data structures for solid modelling in curved-surface environments. Computer Graphics and Applications 5, 21–40.