Implicit encoding of Combinatorial Pyramids

Brun Luc[1], Walter Kropatsch[2]

[1]Labroratoire d'Études en Informatiques
I.U..T Léonard de Vinci
51.100 Reims
France
[2]Institute for Computer-aided Automation
Pattern Recognition and Image Processing Group
Vienna Univ. of Technology
Austria
luc.brun@univ-reims.fr, krw@prip.tuwien.ac.at

Abstract *An irregular pyramid consists of a stack of successively reduced graphs. Each smaller graph is deduced from the preceding one by the contraction or the removal of a set of edges. Using a fixed decimation ratio we need approximately $\mathcal{O}(\log(\textit{image size}))$ graphs to encode the whole pyramid. A combinatorial map encodes a planar graph thanks to two permutations encoding the edges and their orientation around the vertices. We present in this article an encoding of a combinatorial pyramid which allows to fold the whole pyramid in the base level layer and provides at the same time a measure of the importance of every pixel. Any reduced combinatorial maps of the pyramid maybe directly retrieved from this encoding if needed.*

1 Introduction

Regular image pyramids have been introduced as a stack of images with decreasing resolutions. Since then, regular pyramids have been widely used in image segmentation and image analysis. However, the rigidity of regular pyramids induces several drawbacks such as the shift-dependence problem and the limited number of regions encoded at a given level of the pyramid [2]. Irregular pyramids overcome these negative properties while keeping the main advantages of their regular ancestors. These pyramids are defined as a stack of successively reduced graphs. Each graph is built from the graph below by selecting a set of vertices named surviving vertices and mapping each non surviving vertex to a surviving one [9]. If the initial graph is planar its reduced versions are also planar. Moreover, given an image, if each vertex of the initial graph is associated to one pixel, the set of initial vertices mapped to a surviving vertex defines a region of the image.

The boundaries between two adjacent regions are encoded by the edges of the reduced graphs. Using simple graphs (without multiple edges between vertices nor self-loops) multiple boundaries between two regions are mapped into only one edge. This drawback may be overcome by using the Dual graph reduction scheme [7]. Within this framework, the reduction operation is performed in two steps: First, the contraction of a set of edges identifies a set of vertices. This operation may create redundant edges such as empty self-loops or double edges [7]. These redundant edges are characterized in the dual of the graph and removed by a set of edge removals. Using such a reduction scheme each edge in the reduced graph corresponds to one boundary between two regions. Moreover, inclusion relationships may be differentiated from adjacency ones in the dual graph.

Combinatorial Pyramids inherit all the useful properties from the dual graph pyramids with the addition that they also preserve the local orientation of edges around vertices and faces.

The remaining of this paper is as follows: We present in section 2 the combinatorial map model together with its main properties. In section 3 we define the contraction and removal operations within the combinatorial map framework. In Section 4 we present a folding of the Combinatorial Pyramid based on an encoding of the maximal level where an edge survives. Using such a folding the relevance of an edge according to a particular reduction scheme is stored in the base level combinatorial map. We thus combine local and global information within a same representation. Moreover, this folding allows both to recover the whole pyramid or one of its graph from the base level.

2 Combinatorial Maps

Combinatorial maps and generalized combinatorial maps define a general framework which allows to encode any subdivision of nD topological spaces orientable or non-orientable with or without boundaries. An exhaustive comparison of combinatorial maps with other boundary representations such as cell-tuples and quad-edges is presented

*This Work was supported by the Austrian Science Foundation under P14445-MAT.

in [8]. Recent trends in combinatorial maps apply this framework to the segmentation of 3D images [1, 3] and the encoding of hierarchies.

The remaining of this paper will be based on 2D combinatorial maps which we simply call combinatorial maps. A combinatorial map may be seen as a planar graph encoding explicitly the orientation of edges around a given vertex. Figure 1 demonstrates the derivation of a combinatorial map from a plane graph. First edges are split into two half edges called *darts*, each dart having its origin at the vertex it is attached to. The fact that two half-edges (darts) stem from the same edge is recorded in the reverse permutation α. A second permutation σ encodes the set of darts encountered when turning counterclockwise around a vertex (see e.g. the σ-orbit $(-8, -3, 11, 4)$ encoding the central vertex in Figure 1).

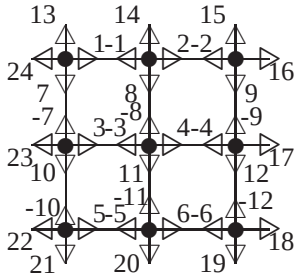


Figure 1: A 3×3 grid encoded by a combinatorial map

The symbols $\alpha^*(d)$ and $\sigma^*(d)$ stand, respectively, for the α and σ orbits of the dart d. More generally, if d is a dart and π a permutation we will denote the π-orbit of d by $\pi^*(d)$.

A combinatorial map G is the triplet $G = (\mathcal{D}, \sigma, \alpha)$, where $\mathcal{D}$ is the set of darts and σ, α are two permutations defined on $\mathcal{D}$ such that α is an involution:

$$\forall d \in \mathcal{D} \quad \alpha^2(d) = d \tag{1}$$

Note that, if the darts are encoded by positive and negative integers, the involution α may be implicitly encoded by sign (Figure 1).

Given a combinatorial map $G = (\mathcal{D}, \sigma, \alpha)$, its dual is defined by $\overline{G} = (\mathcal{D}, \varphi, \alpha)$ with $\varphi = \sigma \circ \alpha$ (Figure 2). The orbits of the permutation φ encode the set of darts encountered when turning around a face (see e.g. the φ-orbit $(1, 8, -3, -7)$ encoding the top-left face in Figure 1). Note that, using a counter-clockwise orientation for permutation σ, each dart of a φ-orbit has its associated face on its right.

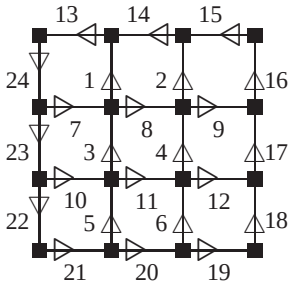


Figure 2: Dual of the combinatorial map represented in Figure 1

2.1 Properties of Combinatorial maps

As mentioned above the dual combinatorial map may be simply computed by composing the permutations σ and α. This dual map may thus be implicitly encoded. This implicit encoding allows to reduce both the memory requirements and the execution times since only one data structure needs to be stored and processed. Moreover the combinatorial map formalism is formally defined in any dimensions. Algorithms defined within the $2D$ combinatorial pyramid framework may thus be extended to higher dimensions(see e.g. [8]). This hypothesis is confirmed by some recent results from Damiand and Lienhardt [6]. Finally, the combinatorial map formalism encodes explicitly the orientation of edges around each vertex. This additional feature allows to encode fine relationships on the partition such as the sequence of regions encountered when turning with a given orientation around one region.

3 Combinatorial Pyramids

As in the dual graph pyramid scheme (Section 1) a combinatorial pyramid is defined by an initial combinatorial map successively reduced by a sequence of contraction or removal operations. In order to preserve the connectivity among the elements chosen to survive to the higher level we forbid the removal of bridges and the contraction of self-loops. A self-loop in the initial combinatorial map corresponds to a bridge in its dual and vice-versa. In the same way, a contraction operation in the initial combinatorial map is equivalent to a removal operation performed in its dual. Therefore, the exclusion of bridges and self-loops from respectively removal and contraction operations corresponds to a same constraint applied alternatively on the dual combinatorial map and the original one.

In order to avoid the contraction of self-loops, the set of edges to be contracted must form a forest of the initial combinatorial map. The graph of a combinatorial map is a forest if it does not contain a cycle. A set of edges to be contracted satisfying the above requirement is called a contraction kernel:

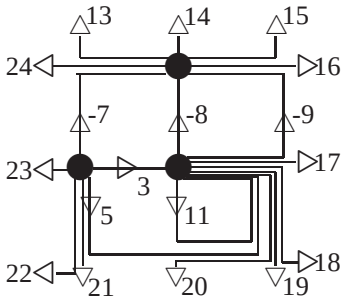
Definition 1 Contraction Kernel

Given a connected combinatorial map $G = (\mathcal{D}, \sigma, \alpha)$ the set $K \subset \mathcal{D}$ will be called a contraction kernel iff K is a forest of G.

The set $\mathcal{SD} = \mathcal{D} - K$ is called the set of surviving darts.

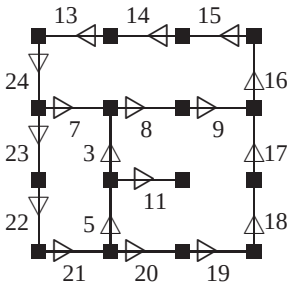
Given a contraction kernel K, we denote by $\mathcal{CC}(K)$ its set of connected components. Since K is a forest, each $\mathcal{T} \in \mathcal{CC}(K)$ is a tree. Intuitively, a tree $\mathcal{T} \in \mathcal{CC}(K)$ collapses into one vertex, a connected set of vertices of the initial combinatorial map.

The contraction of a combinatorial map by a contraction kernel may induce the creation of redundant edges. These edges named double edges and direct-self-loops are respectively characterized by $\varphi^2(d) = d$ and $\sigma(d) = \alpha(d)$ where d is one of the darts of the edge $\alpha^*(d)$. Both double edges and direct self-loops may be removed by using a removal kernel defined as a forest of the dual combinatorial map. This last constraint insures that no self-loop will be contracted in the



(a) $G_1 = G/K_1$

Figure 3: Reduction of the initial grid displayed in Figure 1 by the contraction kernel K_1



(a) $\overline{G_1} = \overline{G/K_1}$

Figure 4: Dual of the contracted grid represented in Figure 3.

dual combinatorial map and thus that no bridge may be removed in the initial one. The contracted combinatorial map may thus be simplified using parallel edge removals.

3.1 Example of contraction and simplification

Figure 3 illustrates a contraction of the initial combinatorial map represented in Figure 1 by a contraction kernel K_1 defined by the trees $\alpha^*(1,2)$, $\alpha^*(4,12,6)$ and $\alpha^*(10)$. Since each initial vertex is incident to a contracted edge this forest spans the initial combinatorial map and we obtain 3 surviving vertices encoding 3 regions.

Double edges encode a same adjacency relationship between two vertices. For example, the edges $\alpha^*(8)$ and $\alpha^*(9)$ in Figure 3 encode both an adjacency relationship between the top vertex and the center one. Such edges correspond to an artificial split of a boundary between two regions (see Figure 4). On the other hand, direct self-loops, such as $\alpha^*(11)$ (Figure 3) may be interpreted in the dual combinatorial map as inner-boundaries of a face (Figure 4).

Figure 5 shows the combinatorial map deduced from Figure 3 and simplified by the removal kernel K_2 defined in the dual combinatorial map by the trees: $\alpha^*(15,14,13,24)$, $\alpha^*(9)$, $\alpha^*(11,3)$, $\alpha^*(19,18,17)$ and $\alpha^*(22,21)$. Note that given a sequence of double edges, the choice of the surviving edge is arbitrary. For example, a choice of the tree $\alpha^*(20,19,18)$ instead of $\alpha^*(19,18,17)$ would lead to an equivalent simplified combinatorial map with a surviving edge equal to $\alpha^*(17)$ instead of $\alpha^*(20)$.

3.2 Connecting Walks

The creation of the reduced combinatorial map from a contraction or a removal kernel is performed in parallel by using connecting walks. Given a combinatorial map $G = (\boldsymbol{\mathcal{D}}, \sigma, \alpha)$, a kernel K and a surviving dart $d \in \mathcal{SD} = \boldsymbol{\mathcal{D}} - K$, the connecting walk associated to d is either equal to:

$$CW(d) = d, \varphi(d), \ldots, \varphi^{n-1}(d)$$

with $n = Min\{p \in \mathbb{N}^* \mid \varphi^p(d) \in \mathcal{SD}\}$, if K is a contraction kernel and

$$CW(d) = d, \sigma(d), \ldots, \sigma^{n-1}(d)$$

with $n = Min\{p \in \mathbb{N}^* \mid \sigma^p(d) \in \mathcal{SD}\}$ if K is a removal kernel.

Given a kernel K and a surviving dart $d \in \mathcal{SD}$, such that $CW(d) = d.d_1 \ldots d_p$, the successor of d within the reduced combinatorial map $G' = G/K = (\mathcal{SD}, \sigma', \alpha)$ is retrieved from $CW(d)$ by the following equations:

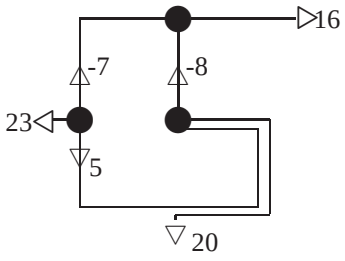
$$\begin{array}{lcll} \varphi'(d) & = & \varphi(d_p) & \text{if } K \text{ is a contraction kernel} \\ \sigma'(d) & = & \sigma(d_p) & \text{if } K \text{ is a removal kernel} \end{array} \quad (2)$$

Note that, if K is a contraction kernel, the connecting walk $CW(d)$ allows to compute $\varphi'(d)$. The σ-successor of d within the contracted combinatorial maps is retrieved from $CW(\alpha(d)) = \alpha(d).d'_1, \ldots, d'_p$. Indeed, we obtain by using equations 1 and 2: $\varphi'(\alpha(d)) = \sigma'(\alpha(\alpha(d))) = \sigma'(d) = \varphi(d'_p)$.

4 Folding and unfolding the pyramid

Connecting walks allow us to reduce a combinatorial map either by contraction or removal kernels. Starting from an initial combinatorial map G_0 and given a sequence of kernels $K_1, \ldots, K_n$ we can thus build the sequence of reduced combinatorial maps $G_0, G_1, \ldots, G_n$ encoding explicitly the pyramid as a stack of successively reduced combinatorial maps. The aim of this section is to define an implicit encoding of the pyramid by additional attributes stored in the initial combinatorial map G_0.

Each connecting walk defined in one combinatorial map G_i, $i \in \{1, \ldots, n-1\}$ connects a dart in G_{i+1} to a sequence



(a) $(G/K_1) \setminus K_2$

Figure 5: Reduction of the contracted combinatorial map displayed in Figure 3 by the removal kernel K_2

of darts in G_i. Such a sequence corresponds to the notion of reduction window [2, 9] within the Pyramid framework. The transitive closure of this father-child relationships defines the notion of receptive field. Within the Combinatorial Pyramid framework this notion is encoded by connecting dart sequences which connect one dart in a combinatorial map $G_i = (\mathcal{SD}_i, \sigma_i, \alpha)$ to a sequence of darts defined in the base level combinatorial map $G_0 = (\boldsymbol{\mathcal{D}}, \sigma, \alpha)$. These sequences are defined recursively by $CDS_0(d) = d$ for any $d \in \boldsymbol{\mathcal{D}}$ and:

For each i in $\{1, \ldots, n\}$ and each dart $d \in \mathcal{SD}_i$:

- If K_i and K_{i-1} have the same type, i.e. if both kernels are contraction kernels or removal ones:
  $CDS_i(d) = CDS_{i-1}(d_1).\ldots.CDS_{i-1}(d_p)$
- If K_i and K_{i-1} have different types:
  $CDS_i(d) = d_1.CDS^*_{i-1}(\alpha(d_1)).\ldots.d_p.CDS^*_{i-1}(\alpha(d_p))$

where $CW_i(d) = d_1 \ldots, d_p$ and $CDS^*_{i-1}(\alpha(d_j))$, $j \in \{1, \ldots, p\}$ denotes the sequence $CDS_{i-1}(\alpha(d_j))$ without its first dart.

Connecting dart sequences may be understood as the transitive closure of the hierarchical relation defined by connecting walks. Both sequences should thus satisfy similar properties. Indeed, given one dart $d \in \mathcal{SD}_i$, such that $CDS_i(d) = d_1, \ldots, d_p$ we have shown that [5]:

- If K_i is a contraction kernel:

$$\varphi_i(d) = \begin{cases} \varphi(d_p) & \text{if } d_p \text{ has been contracted} \\ \sigma(d_p) & \text{if } d_p \text{ has been removed} \end{cases} \quad (3)$$

- If K_i is a removal kernel:

$$\sigma_i(d) = \begin{cases} \varphi(d_p) & \text{if } d_p \text{ has been contracted} \\ \sigma(d_p) & \text{if } d_p \text{ has been removed} \end{cases} \quad (4)$$

Note that equations 3 and 4 are similar to equations 2 defined for connecting walks.

We additionally showed that given one dart $d \in \mathcal{SD}_i$ such that $CDS_i(d) = d_1.\ldots.d_p$, we have $d = d_1$ and $\{d_2, \ldots, d_p\} \subset \cup_{j=1}^{i} K_j$. The first dart of a connecting dart sequence is thus the only one which survives up to level i. Moreover, we have shown that the connecting dart sequences defined at level i define a partition of the initial set of darts $\boldsymbol{\mathcal{D}}$. The connecing walk satisfy a similar property.

4.1 Folding the Pyramid

Given the set of connecting dart sequences defined at level i, one can use equations 3 and 4 to retrieve the reduced combinatorial map G_i. However, using the recursive construction scheme defined above, the definition of connecting dart sequences at level i is based both on connecting dart sequences defined at level $i-1$ and an explicit encoding of the level $i-1$. Indeed, the connecting walks used to build the connecting dart sequences at level i are defined on G_{i-1}. The recursive construction of connecting dart sequences at level i requires thus an explicit encoding of the pyramid form level 0 to level $i-1$. This recursive construction scheme may be avoided by using the following theorem [5]:

Theorem 1 *Given a combinatorial map $G_0 = (\boldsymbol{\mathcal{D}}, \sigma, \alpha)$, a sequence of contraction kernels or removal kernels $K_1, K_2 \ldots, K_n$, the relation between the successive darts of a connecting dart sequence $CDS_i(d) = (d.d_1, \ldots, d_p)$, with $i \in \{1, \ldots, n\}$ and $d \in \mathcal{SD}_i$ is as follows:*

$$d_1 = \begin{cases} \varphi(d) & \text{if } K_i \text{ is a contraction kernel} \\ \sigma(d) & \text{if } K_i \text{ is a removal kernel} \end{cases}$$

and for each j in $\{2, \ldots, p\}$

$$d_j = \begin{cases} \varphi(d_{j-1}) & \text{if } d_{j-1} \text{ has been contracted} \\ \sigma(d_{j-1}) & \text{if } d_{j-1} \text{ has been removed} \end{cases}$$

One connecting dart sequence defined at level i may thus be traversed if we know the initial dart in $\mathcal{SD}_i$ defining it and if we determine the operation which has reduced each dart of the connecting dart sequence. The above remark suggests an encoding of the pyramid by two functions:

1. one function $state$ from $\{1, \ldots, n\}$ to the binary states $\{Contracted, Removed\}$ which specifies the type of each kernel.
2. One function $level$ defined for all darts in $\boldsymbol{\mathcal{D}}$ such that $level(d)$ is equal to the maximal level where d survives:

   $$\forall d \in \boldsymbol{\mathcal{D}}\ level(d) = Max\{i \in \{1, \ldots, n+1\} \,|\, d \in \mathcal{SD}_{i-1}\}$$

   a dart d surviving up to the top level has thus a level equal to $n+1$. Note that the level of an initial vertex (i.e. a pixel) may be implicitly defined as the minimal level of its darts.

4.2 Unfolding one level of the Pyramid

Given the function $level$, the sequence of kernels and the set of surviving darts may be retrieved by the following equations:

$$\forall i \in \{1, \ldots, n\} \begin{cases} K_i &= \{d \in \boldsymbol{\mathcal{D}} \,|\, level(d) = i\} \\ \mathcal{SD}_i &= \{d \in \boldsymbol{\mathcal{D}} \,|\, level(d) > i\} \end{cases} \quad (5)$$

Moreover, given a dart $d \in \mathcal{SD}_i$ and using both functions $state$ and $level$, the traversal of a connecting dart sequence defined by Theorem 1 may be written as follows:

$$d_1 = \begin{cases} \varphi(d) & \text{If } state(i) = Contracted \\ \sigma(d) & \text{If } state(i) = Removed \end{cases}$$

and for each j in $\{2, \ldots, p\}$

$$d_j = \begin{cases} \varphi(d_{j-1}) & \text{If } state(level(d_{j-1})) = Contracted \\ \sigma(d_{j-1}) & \text{If } state(level(d_{j-1})) = Removed \end{cases} \quad (6)$$

Given a connecting dart sequence $CDS_i(d) = d.d_1 \ldots, d_p$, if we define d_{p+1} by $\varphi(d_p)$ if d_p is contracted and $\sigma(d_p)$ is d_p is removed, d_{p+1} is either equal to $\varphi_i(d)$ or $\sigma_i(d)$ according to K_i (equations 3 and 4). We have thus in both cases, $d_{p+1} \in \mathcal{SD}_i$ and $level(d_{p+1}) > i$ (equation 5). This last remark allows us to determine the last dart of a connecting dart sequence as the one whose successor defined by equation 6 has a level strictly greater than i. The above considerations lead to the design of the function survive (Algorithm 1) which traverses the connecting dart sequence of a dart $d \in \mathcal{SD}_i$ and return the successor of its last dart according to equation 6. Algorithm 1 performs a call to the function survive for each dart $d \in \mathcal{SD}_i$. If K_i is a contraction

kernel, function survive is called with parameters i and $\alpha(d)$ and traverses $CDS_i(\alpha(d))$. Using equation 3, the successor of the last dart of $CDS_i(\alpha(d))$ is equal to $\varphi_i(\alpha(d)) = \sigma_i(d)$ (equation 1, Algorithm 1 line 4). In the same way, if K_i is a removal kernel, the successor of the last dart of $CDS_i(d)$ is equal to $\sigma_i(d)$ (equation 4, Algorithm 1 line 6).

```
 1 For each d ∈ SD_i
 2 {
 3  if (state(i)==Contracted)
 4       σ_i(d) =survive(i,α(d))
 5  else
 6       σ_i(d) =survive(i,d)
 7 }
 8 dart survive(int i,dart d)
 9 {
10  if (level(d)>i)
11       return d;
12  if(state(level(d))== Contracted)
13       return survive(i,φ(d));
14  return survive(i,σ(d));
15}
```

Algorithm 1: *Unfolding of level $G_i = (\mathcal{SD}_i, \sigma_i, \alpha)$.*

We have shown, that the complexity of Algorithm 1 is equal to $\mathcal{O}(|\boldsymbol{\mathcal{D}}|)$ where $|\boldsymbol{\mathcal{D}}|$ denotes the cardinal of the initial set of darts $\boldsymbol{\mathcal{D}}$. We also designed a parallel version of Algorithm 1. The complexity of this last algorithm is equal to $\mathcal{O}(\log(|CDS_i^{max}(d)|))$ where $|CDS_i^{max}(d)|$ denotes the length of the longest connecting dart sequence defined at level i.

4.3 Unfolding the whole Pyramid

Let us consider the top of a pyramid at level n, one dart $d \in \mathcal{SD}_n$ and the sequence of darts $\mathcal{SC}_n(d)$ defined by:

$$\mathcal{SC}_n(d) = \begin{cases} CDS_n(d)\varphi_n(d) & \text{if } K_n \text{ is a contraction kernel} \\ CDS_n(d)\sigma_n(d) & \text{if } K_n \text{ is a removal kernel.} \end{cases}$$

Note that a call to the function `survive(n,d)` (Algorithm 1) traverses the sequence $\mathcal{SC}_n(d)$. We have shown that all the σ or φ successors of a dart $d' \in \mathcal{SC}_n(d)$ belong to $\mathcal{SC}_n(d)$. More precisely [4]:

$$\forall d \in \mathcal{SD}_n, \forall b \in \mathcal{SC}_n(d), \forall j \in \{0, \ldots, level(b)-1\}$$
$$\begin{cases} \varphi_j(b) = b' & \text{if } b \text{ has been contracted} \\ \sigma_j(b) = b' & \text{if } b \text{ has been removed} \end{cases} \quad (7)$$

where b' is the first dart after b in $\mathcal{SC}_n(d)$ with a level greater than j.

In other word, the σ or φ successors at level j of a dart b belonging to a sequence $\mathcal{SC}_n(d)$ may be read in this sequence by selecting the first dart with a level greater than j encountered after b in $\mathcal{SC}_n(b)$. Note that if b is contracted equation 7 provides all its φ successor. The σ successors of this dart may be determined by scanning the sequence $\mathcal{SC}_n$ which contains $\alpha(b)$. Indeed, the scanning of this sequence provides the value of $\varphi_j(\alpha(b)) = \sigma_j(b)$ (equation 1) for all $j \in \{0, \ldots, level(b)-1\}$.

The computation of the whole pyramid (Algorithm 2) from the functions $level$ and $state$ is based on an update of the function survive (Algorithm 1). Note that the set of darts traversed by both algorithms remains the same (lines 13-14, Algorithm 1 and 15-18 Algorithm 2). The main difficulty induced by equation 7 is that the successors of a dart b contained in $\mathcal{SC}_n(b)$ will be traversed after b by the function `updated_survive` and are thus unknown when this function traverses b. To overcome this difficulty, we use an array $prec$ whose entry j contains for each dart traversed by `updated_survive` the last encountered dart with a level greater than j. We have thus using equation 7:

$$\forall j \in \{0, \ldots, level(prec[j])-1\}$$
$$\begin{cases} \varphi_j(prec[j]) = b & \text{if } prec[j] \text{ has been contracted} \\ \sigma_j(prec[j]) = b & \text{if } prec[j] \text{ has been removed} \end{cases}$$

where b is the dart currently traversed by the algorithm.

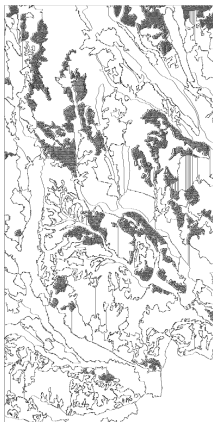
This array if updated before traversing each new dart (lines 28-29 Algorithm 2).

```
1   For each dart d in SD_n
2   {
3       updated_survive(d)
4   }
5   updated_survive(dart d)
6   {
7       dart d';
8       dart prec[n+1];
9
10      for(j=1;j<level(d);i++)
11          prec[j]=d;
12      d'=d;
13      do
14      {
15          if(state(level(d'))==Contracted)
16              d' = φ(d);
17          else
18              d' = σ(d);
19
20          for(j=1;j<level(d');j++)
21          {
22              if(state(level(prec[j]))
23                          == Contracted)
24                  φ_j(prec[j]) = d';
25              else
26                  σ_j(prec[j]) = d';
27          }
28          for(j=1;j<level(d');j++)
29              prec[j]=d'
30      }
31      while(level(d')≤ n)
32  }
33
```

Algorithm 2: *Updated version of the function survive which computes all the successors of the darts contained in $\mathcal{SC}_n(d)$*

Note that since the set of connecting dart sequences at level n define a partition of $\boldsymbol{\mathcal{D}}$ (Section 4), Algorithm 2 considers all the darts of the pyramid and compute the successors of each dart only once.

(a) Connected components



(b) Folding of the pyramid

4.4 Memory requirements

Since both darts of an edge are simultaneously contracted or removed the function *level* may be encoded on only one dart of each edge (e.g. the positive dart if α is encoded by the sign). Using a pyramid with n level, the level of each dart may be encoded with $\log_2(n)$ bits. The amount of memory required to store the function *state* being negligible beside the one required by the function level, the memory requirement of our implicit encoding is equal to:

$$\frac{1}{2}|\mathcal{D}|\log_2(n) \text{ bits.}$$

On the other hand an explicit encoding of the permutation σ for each level requires $|\mathcal{D}|\log_2(|\mathcal{D}|)\frac{k}{k-1}$ bits where k is the reduction factor.

Both quantities may be compared by using the relationship $\frac{|\mathcal{D}|}{k^n} = 1$ where we suppose that the top of the pyramid is reduced to a single edge. The amount of memory required by an explicit encoding of the pyramid may then be rewritten as:

$$2|\mathcal{D}|\log_2(|\mathcal{D}|) = 2|\mathcal{D}|n\log_2(k)$$

The amount of memory required by the implicit and explicit encodings increase respectively as a logarithmic and a linear function of the height n of the pyramid.

Fig. 4.4 shows the folding of a pyramid on a satellite image[1]. The connected components of this image are encoded by a combinatorial pyramid made of 3 levels G_0, G_1 and G_2. The base level G_0 encodes the planar sampling grid, the combinatorial map G_1 encodes the regions of the partition after the contraction operation and the top of the pyramid G_2 is deduced from G_1 by removing the redundant edges induced by the contraction operation. The image in Fig. 4.4(b) is obtained by storing the level of two darts in each pixel of the initial sampling grid. These levels are then stored on 12 bits in order to obtain a 24 bits color image. Note that many informations on the partition such that the interior of regions or the boundaries appear in this image of levels. Using alternatively contraction and removal operations, the function *state* may deduced from the parity of the levels.

[1] Data provided by the Institute of Surveying, Remote Sensing and Land Information, BOKU Vienna

5 Conclusion

A combinatorial pyramid can be folded into its base level combinatorial map by two functions $state()$ and $level()$. Using these functions and the local orientation of edges around each vertex any reduced combinatorial map may be unfolded directly from the base level. Our method stores in the base level combinatorial map the relevance of each edge according to a particular decimation process. This combination of local and global information in the same representation should improve the performances of matching algorithms applied to images or to the skeleton of shapes.

References

[1] Yves. Bertrand, Guillaume. Damiand, and Christophe. Fiorio. Topological map: Minimal encoding of 3d segmented images. In Jean Michel. Jolion, Walter. Kropatsch, and Mario. Vento, editors, 3^{rd} *Workshop on Graph-based Representations in Pattern Recognition*, pages 64–73, Ischia(Italy), May 2001. IAPR-TC15, CUEN.

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

[3] Jean Pierre. Braquelaire, Piere. Desbarats, and Jean Philippe. Domenger. 3d split and merge with 3-maps. In Jean Michel. Jolion, Walter. Kropatsch, and Mario. Vento, editors, 3^{rd} *Workshop on Graph-based Representations in Pattern Recognition*, pages 32–43, Ischia(Italy), May 2001. IAPR-TC15, CUEN.

[4] Luc. Brun. *Traitement d'images couleur et pyramides combinatoires*. Habilitation à diriger des recherches, Université de Reims, 2002. URL : http://www.univ-reims.fr/membre/luc/ARTICLES/hdr.pdf

[5] L.. Brun and Walter. Kropatsch. The construction of pyramids with combinatorial maps. Technical Report 63, Institute of Computer Aided Design, Vienna University of Technology, lstr. 3/1832,A-1040 Vienna AUSTRIA, June 2000. URL : www.prip.tuwien.ac.at/ftp/pub/publications/trs/tr63.ps.gz

[6] Guillaume. Damiand and Pascal. Lienhardt. Removal and contraction for n-dimensional generalized maps. In Horst. Wildenauer and Walter. Kropatsch, editors, *Proceedings of the Computer Vision Winter Workshop*, pages 208–221, Bad Ausse Austria, February 2002.

[7] Walter G.. Kropatsch. Building Irregular Pyramids by Dual Graph Contraction. *IEE-Proc. Vision, Image and Signal Processing*, Vol. 142(No. 6):pp. 366–374.

[8] P.. Lienhardt. Topological models for boundary representations: a comparison with n-dimensional generalized maps. *Computer-Aided Design*, 23(1):59–82, 1991.

[9] Annick. Montanvert, Peter. Meer, and Azriel. Rosenfeld. Hierarchical image analysis using irregular tessellations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):307–316, APRIL 1991.