Basic Morphological Operators Applied On Partitions

Serge BEUCHER
CMM - Mines ParisTech
(March 26, 2013)

Abstract

    This document describes morphological operators designed for partition images. Two approaches are addressed. In the first one, each cell of the partition is considered independently of the others and various anti-extensive operators can be defined on it. However, these operators can be applied directly on the whole partition. There is no need to extract each cell to transform it. The second approach, on the contrary, considers a partition as a representation of a Delaunay graph. Morphological operators can be defined on this graph. This document explains how these operators can be implemented directly on the partition image.
These operators have been implemented with the MAMBA Image library.

1. Introduction

    Partitions (also called partition images or mosaic images) are very common in image analysis and in Mathematical Morphology. They can be found as the result of various morphological operators: watershed transforms, hierarchical segmentations, labellings, residual transforms, especially with the associated functions of a residual operator.

The classical way for handling partitions considers that each tile or cell of the partition is a vertex of the valued Delaunay graph associated to this partition. This representation allows to define basic morphological transforms (erosion and dilation) on this neighbourhood graph. In this case, the partition is not modified. Only the valuations of the tiles are changed. These operators have been seldom used in practice. It is surely due to the fact that they seem to require the Delaunay graph associated to the partition to be obtained. However, it is possible to define them directly on the partition image rather quickly. We shall discuss this point in the second part of this paper.

The second approach is totally the opposite. Indeed, when a partition is available, it is very interesting to have the capability to apply basic operators to each cell of the partition as if it was alone and not adjacent to the other cells. However, as a partition is often represented as a numerical function where a single grey level labels all the pixels which belong to the same cell, it is not simple to realize efficiently, that is rapidly, such transforms. Moreover, some operators are not always meaningful. It is the case, in particular, when dealing with extensive transforms (as dilations for instance).

We shall explain in the first part of this document which operators can be defined and how they can be obtained.

These operators have been implemented with the MAMBA IMAGE library [1]. The source code is given in the appendix.

2. Definitions

Let us start with some definitions.

2.1. Partition image and background cells

Let $\{Y_k\}$, $k \in [1, K]$ be a partition of the space E:

$$\forall Y_k, Y_{k'}, \ k \neq k' : Y_k \cap Y_{k'} = \varnothing$$
$$\bigcup_{k \in [1,K]} Y_k = E$$

K is the number of connected components (tiles) of the partition.

A 2D partition image is represented by a numerical image f defined as:

$$x \in E : \ f(x) = v_k \text{ if } x \in Y_k$$

v_k is an integer value also called the label of the cell Y_k. If Y_k and $Y_{k'}$ are adjacent cells, we have $v_k \neq v_{k'}$.

This definition simply means that all the pixels belonging to the same cell have the same value but different cells may have the same label provided that they are not adjacent. Note that, thanks to this definition (less restrictive than the definition where each cell has a unique label), any greyscale image can be considered as a partition image (Fig. 1).

Therefore, as we may have different tiles with the same label i, we will note:

$$X_i = \bigcup_{v_k = i} Y_k$$

the union of all the cells Y_k of the partition which share the same label i. We have:

$$E = \bigcup_{i \in [0,n]} X_i$$

n is the highest label used in the partition.

The set X_0 containing all the cells with a label equal to 0 will play a particular role in the following. This set is called the background of the partition. It is perfectly allowed that a partition does not contain any background. This background allows to define and to use extensive operators.
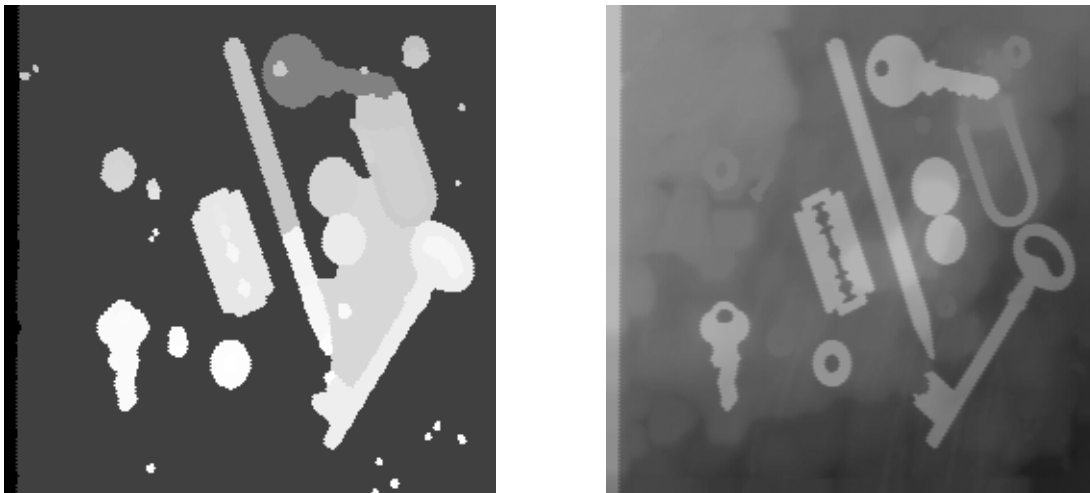

Fig.1: *Examples of partition images. On the left, example of a "classical" partition. However, any grey level image can be considered as a partition (on the right).*

2.2. Indicator function

Another useful definition in the sequel is the following:

Let M be a set included in E. We denote m the valued indicator function of M defined as:

$$m(x) = n \text{ if } x \in M$$
$$m(x) = 0 \text{ if not}$$

2

n is the highest label used in the partition.

3. Operators applied to the cells of a partition

This section will explain how to define and to realize basic operators applied on each cell of the partition. Due to the fact that all the cells are adjacent, the only operators which should be defined must be anti-extensive otherwise the cells would overlap after transformation. However, we shall see that it is possible to apply extensive operators on partitions provided that a background exists in this partition.

3.1. Basic cells erosion

It is obviously possible to compute this operation cell by cell. Each cell is eroded and a new function f' is then built from each eroded cell thus generating a new partition where each previous cell is separated from the others by a background region. However, this procedure would be extremely slow. Let us describe faster ways to obtain this result.

We want to define an operator ε_P applied on the function f representing the partition $\{Y_k\}$ which produces a new function f' representing the partition $\{Y'_k\}$ where each cell Y'_k is the erosion with a structuring element B of the initial cell Y_k :

$$x \in E : f'(x) = v_k \text{ if } x \in Y_k \ominus \check{B}$$

Note that B must contain its origin otherwise the erosion would not be anti-extensive (albeit we know that, if this condition is not fulfilled, this case can be considered as a mere translation of the transformed partition).

Note also that the initial background cells (the set X_0) are also eroded. However, as every point x which does not belong to any eroded cell is given the value 0 in f', the background set X_0' of the final partition is larger than the initial set X_0:

$$X_0 \subset X_0'$$

We may also remark that:

$$x \in E : f'(x) = i \text{ if } x \in \left(X_i \ominus \check{B} \right)$$

where X_i is the union of all the cells Y_k with a label equal to i. Indeed, every cell Y_k belonging to X_i does not touch the other cells of X_i. Therefore, we have:

$$X_i \ominus \check{B} = \bigcup_{v_k = i} \left(Y_k \ominus \check{B} \right)$$

Let us prove that f' is equal to:

$$f' = \varepsilon_P(f) = f \wedge m$$

with m, valued indicator function of the set:

$$M = \left\{ x : \left(f \ominus \check{B} \right)(x) = \left(f \oplus \check{B} \right)(x) \right\}$$

M is the mask of the points of E where the (classical) erosion of f by B is equal to the dilation.

Denoting $Z_i(f)$ the threshold of f at level i, we have:

$$Z_i(f) = \{x \in E : f(x) \geq i\}$$

That is:

$$Z_i(f) = \bigcup_{j \geq i}(X_j)$$

and:

$$Z_i(f') = \bigcup_{j \geq i} \left(X_j \ominus \check{B} \right)$$

We have also:

$$X_j = Z_j(f) \setminus Z_{j+1}(f) = Z_j(f) \cap Z_{j+1}^c(f)$$

3

Therefore, we can write:

$$Z_i(f') = \bigcup_{j \geq i} \left[\left[Z_j(f) \cap Z_{j+1}^c(f) \right] \ominus \overset{\vee}{B} \right]$$

$$Z_i(f') = \bigcup_{j \geq i} \left[\left(Z_j(f) \ominus \overset{\vee}{B} \right) \cap \left(Z_{j+1}^c(f) \ominus \overset{\vee}{B} \right) \right]$$

$$Z_i(f') = \bigcup_{j \geq i} \left[\left(Z_j(f) \ominus \overset{\vee}{B} \right) \cap \left(Z_{j+1}(f) \oplus \overset{\vee}{B} \right)^c \right]$$

and:

$$Z_i(f') = \bigcup_{j \geq i} \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \right]$$

where ε and δ are simply another notations for the erosion and the dilation.
The mask M can also be defined as the threshold at 0 of the difference between the erosion and the dilation (as the structuring element B contains its origin, we have always $\varepsilon(f) \leq \delta(f)$).

$$M = Z_0(\varepsilon(f) - \delta(f))$$

This threshold can be written (see [5], appendix A for more details):

$$Z_0(\varepsilon(f) - \delta(f)) = \bigcup_j \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \right]$$

Let us write the threshold of the function f'. We have indeed:

$$Z(m) = M \ \ (i > 0)$$

Therefore:

$$f' = f \wedge m \ \ \Rightarrow \ \ Z_i(f') = Z_i(f) \cap Z_i(M) = Z_i(f) \cap M$$

Note that when i is equal to 0, all the thresholds are equal to E. Therefore:

$$Z_i(f') = Z_i(f) \cap \left[\bigcup_j \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \right] \right]$$

$$Z_i(f') = \bigcup_j \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \cap Z_i(f) \right]$$

We can split this expression into two parts. The first one corresponds to the sections higher than or equal to i. The second part corresponds to the sections lower than i.

$$Z_i(f') = \left[\bigcup_{j \geq i} \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \cap Z_i(f) \right] \right] \cup \left[\bigcup_{j < i} \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \cap Z_i(f) \right] \right]$$

$$Z_i(f') = \underbrace{\left[\bigcup_{j \geq i} \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \cap Z_i(f) \right] \right]}_{(a)} \cup \underbrace{\left[\bigcup_{j < i} \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \cap Z_i(f) \right] \right]}_{(b)}$$

In term (a), we always have $Z_j(\varepsilon(f)) \subset Z_i(f)$ as the erosion is increasing and anti-extensive. This term can be reduced to:

$$(a) = \bigcup_{j \geq i} \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \right]$$

Regarding term (b), we always have $Z_{j+1}^c(\delta(f)) \cap Z_i(f) = \varnothing$ as the dilation is increasing and extensive. So (b) is equal to the empty set:

$$(b) = \varnothing$$

Finally, we obtain:

$$Z_i(f') = \bigcup_{j \geq i} \left[Z_j(\varepsilon(f)) \cap Z_{j+1}^c(\delta(f)) \right] \ \ \text{Q.E.D.}$$

The erosion f' of the cells of the partition image f by a structuring element B containing its origin is therefore obtained by the following sequence of operations:

- Compute the morphological gradient of f with B.
- Extract the pixels with a gradient equal to 0 (by thresholding). This produces a mask M.
- Then compute the indicator function m of this set M.
- We obtain finally $f' = f \wedge m$.

Fig.2 illustrates this operator. The corresponding MAMBA operation (named ***cellErode***) is given in the appendix.
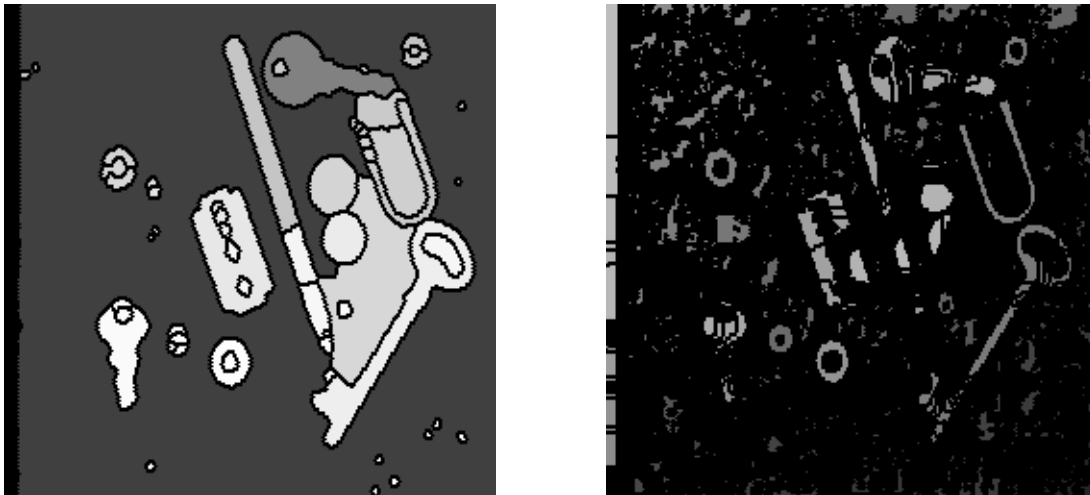

Fig.2*: Cells erosion (size 1, hexagonal structuring element) of the previous partition images (see Fig. 1)..*

3.2. Cells dilations and openings

Defining the dilation of the cells of a partition is not easy as there is generally no "room" between the different cells to insert the dilations. The only exception concerns partitions with a background. In this case, it is possible to dilate each cell and to keep only parts of the dilated cells which fall into the background. If we denote by b the indicator function of the background X_0, we could define the dilation δ_P as:

$$\delta_P = \left[\left(f \oplus \overset{\vee}{B}\right) \wedge b\right] \vee f$$

However, this definition is quite dangerous because, if the structuring element B is large, the dilation of some cells, those with a high valuation, will invade the background region to the detriment of the dilations of the cells with a lower valuation.

To cope with this problem, the only allowed structuring element B must be the elementary structuring element (square or hexagon). This is equivalent to use a geodesic dilation. At each step, the elementary operation is performed:

$$\delta_P = \left[\left(f \oplus \overset{\vee}{B}\right) \wedge b(f)\right] \vee f$$

Where b(f) is the indicator function of the background of f. Then the next step of dilation is achieved by replacing f by δ_P and by iterating the operation.

This transformation however is not very useful. Therefore it has not be implemented.

There is another case where the dilation of the cells can be performed with any structuring element B. This is allowed in the definition of the opening g_P of each cell by B. This operator is simply defined by:

$$\gamma_P = \varepsilon_P \oplus B$$

Indeed, in this case, the dilation by $\overset{\vee}{B}$, transposed structuring element, is always contained in the non background initial cells.

The corresponding MAMBA operation (named ***cellsOpen***) is given in the appendix. Fig. 3 shows an example of opening of a partition.
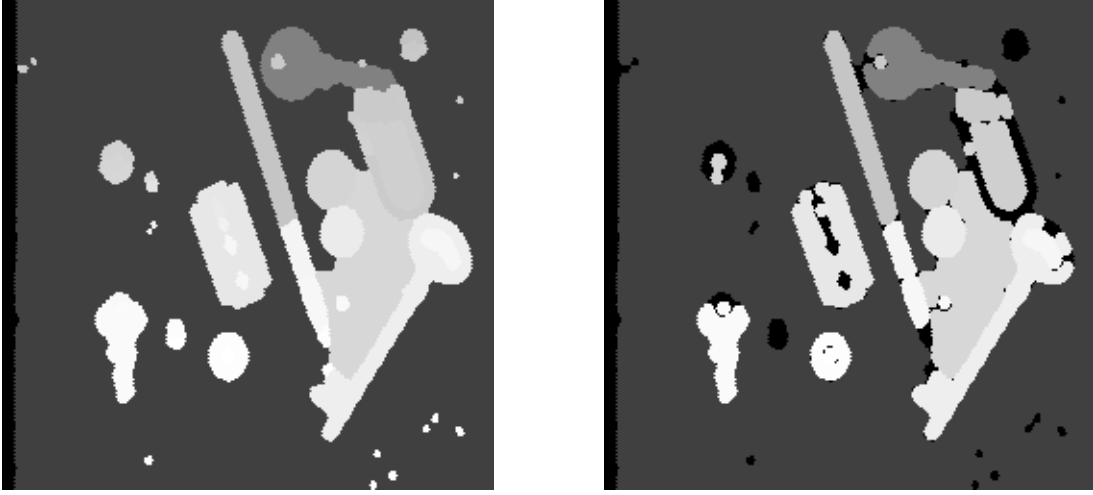
Fig. 3*: Hexagonal opening of size 3 (right) of the cells of the partition (left).*

3.3. Elementary operators applied on cells

Most of the morphological operators use combinations of basic transforms defined on simple structuring elements composed by doublets of points in a given direction. It is, for instance, the case in the MAMBA library. Considering a structuring element L made of two points, extremities of a vector a, two basic operators are defined:

$$\varepsilon^{L}[f(x)] = \inf(f(x), f(x + a))$$

and

$$\delta^{L}[f(x)] = \sup(f(x), f(x + a))$$

Combining these two operators (respectively called *infNeighbor* / *inFarNeighbor* and *supNeighbor* / *supFarNeighbor* in MAMBA) leads to the definition of many morphological transformations. These two operators can also be written:

$$\varepsilon^{L}[f(x)] = f(x) \quad \text{if } f(x) \le f(x + a)$$
$$\varepsilon^{L}[f(x)] = f(x + a) \quad \text{if not}$$

$$\delta^{L}[f(x)] = f(x) \quad \text{if } f(x) \ge f(x + a)$$
$$\delta^{L}[f(x)] = f(x + a) \quad \text{if not}$$

However, when defining the erosion ε_{P} of the cells of a partition, the basic transformation which is used can be written:

$$\varepsilon_{P}^{L}[f(x)] = f(x) \quad \text{if } f(x) = f(x + a)$$
$$\varepsilon_{P}^{L}[f(x)] = 0 \quad \text{if not}$$

The definition of this operator is similar to the two previous ones. Replacing the two inequalities by a strict equality when comparing the values of the two pixels amounts to consider any pixel with a value different from the value of the pixel at the origin of the structuring element as a pixel belonging to the outside of the cell. This is precisely what we want.

A dual operator η_{P}^{L} can also be defined. We can write:

$$\eta_{P}^{L}[f(x)] = f(x) \quad \text{if } f(x) \neq f(x + a)$$
$$\eta_{P}^{L}[f(x)] = 0 \quad \text{if not}$$

These two operators have been programmed with MAMBA. They are named ***equalNeighbor*** and ***nonEqualNeighbor*** and are given in the appendix.

These two operators allow to define hit-or-miss and thinnings transforms applied to each non background cell of the partition.

Let (T, T') be a doublet of structuring elements, with $T \cap T' = \varnothing$. We can define:

$$\varepsilon_{P}^{T}[f(x)] = f(x) \quad \text{if } f(x) = f(x'), \forall x' \in T_{x}$$

6

$$\varepsilon_P^T[f(x)] = 0 \quad \text{if not}$$
$$\eta_P^{T'}[f(x)] = f(x) \quad \text{if } f(x) \neq f(x'), \forall x' \in T_x'$$
$$\eta_P^{T'}[f(x)] = 0 \quad \text{if not}$$

The Hit-or-Miss transform ω_P of the partition P by (T, T') is therefore:

$$\omega_P^{T,T'} = \varepsilon_P^T \wedge \eta_P^{T'}$$

We see that, in order to get non zero and anti-extensive transforms, the common origin of both structuring elements T and T' must belong to T.

A partition thinning τ_P is defined by:

$$\tau_P = I - \omega_P$$
$$\tau_P(f) = f - \omega_P(f)$$

These operators (**cellsHMT**, **cellsThin** and **cellsFullThin**) have been implemented and are given in the appendix. Fig. 4 gives the result of a complete thinning operator applied on a partition image when an homotopic structuring element is used (D element, see [3]).
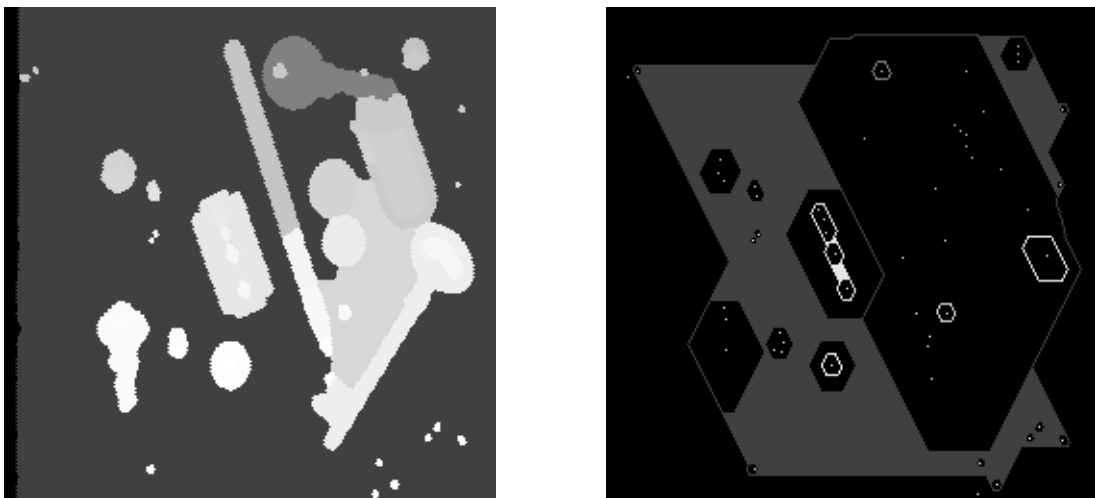
Fig. 4: *Complete D thinning operator applied to partition cells. When a cell is not simply connected, this operator cannot reduce it to a single point (homotopy preservation).*

3.4. Distance function

A fast computation of the distance function $d_P(f)$ of each non background cell can be achieved easily.
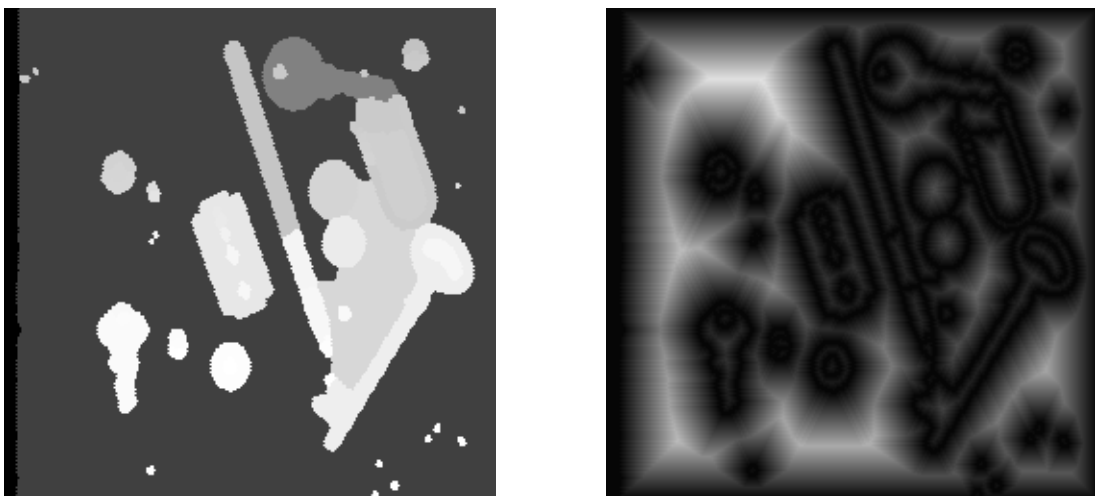

Fig. 5: *Distance function of the cells (right) of the partition (left).*

We simply need to perform a size one erosion of the partition image. Then, after a thresholding of the eroded partition, we compute the distance function of the binary set. The final distance function is obtained by adding the indicator function of the initial non background cells (Fig. 5).

Here is the detailed algorithm:

- Compute the size 1 cells erosion, $\varepsilon_P(f)$
- Extract from the eroded partition the non zero pixels: $X = \{x : \varepsilon_P(f(x)) \neq 0\}$
- Compute the distance function of X, $d(X)$
- Define the indicator function k: $k(x) = 1$ if $f(x) > 0$
- Compute the cells distance function d_P: $d_P(f) = d(X) + k$

This operator is named ***cellsComputeDistance*** in the appendix.

3.5. Cells reconstructions

Cells reconstruction is a very important tool when dealing with partitions. Two main operators are needed. The first one consists in reconstructing among the different cells of a partition those which are marked. The marker set (binary) M allows to select some cells in the initial partition f. The final partition $R_P(f, M)$ is composed of these marked cells (their label value is preserved) and of background cells. The second operator aims at replacing the label value of some cells by a new one. The cells to be modified are pointed by a marker, the new value being the grey level of the marker. These two operators are in fact very similar and they are built the same way. We can consider that the first operator is equivalent to the second one where each marker value is identical to the label value of the corresponding marked cell. If each cell of the partition had a unique value (as many labels as cells), these two transformations could be implemented easily through the definition of a lookup table where the initial labels would be replaced by new ones according to the values indicated by the markers. However, with the extended definition of a partition given above, the implementation is more complex.
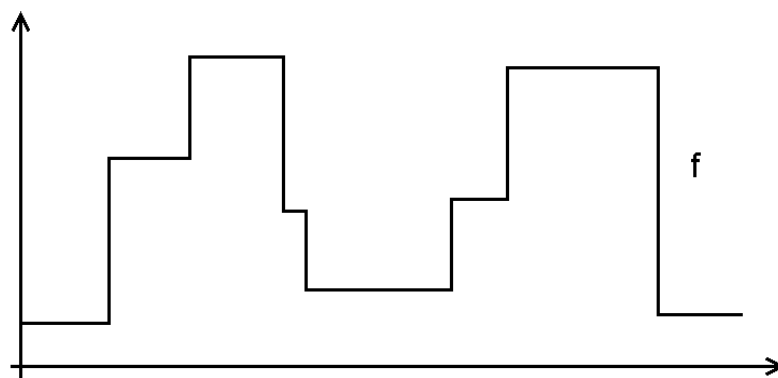

Fig. 6*: Profile of a partition image f in a given direction.*

Indeed, using the geodesic reconstruction can be a solution but if, in the one hand, the size of the cells is very small (one pixel wide for instance) and/or, in the other hand, the markers are placed on the boundary of the cells, the geodesic reconstruction will not work because either this reconstruction will invade non marked adjacent cells or some cells will not be reconstructed despite the fact that they are marked. One solution to this issue consists in using directional reconstructions. This operator already exists in the MAMBA library

(*buildNeighbor*). Fig. 6 to 14 illustrate how to use it. Note that, as the process is directional, the illustrations which are monodimensional describe nevertheless the exact algorithm as it uses *buildNeighbor* which is a true monodimensional operator.

Let us consider a profile in a given direction α of a partition f (Fig. 6). The first step of the algorithm will consist in detecting all the grey levels transitions in a given direction by means of a directional gradient g_a for instance (Fig. 7):

$$g_a = \left(f \oplus \overset{\vee}{L}_a \right) - \left(f \ominus \overset{\vee}{L}_a \right)$$

Where L_a is a doublet of adjacent points in direction α.
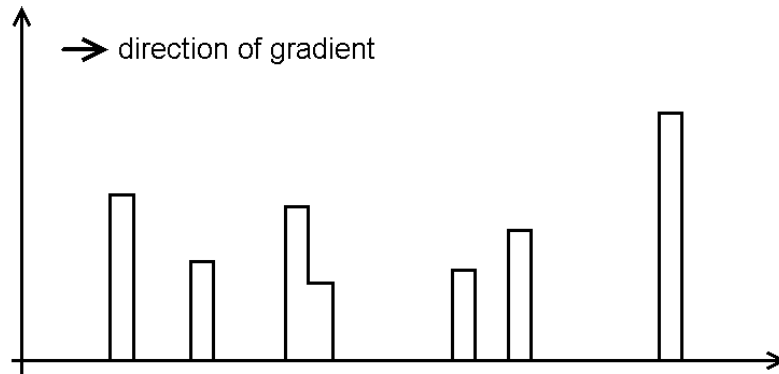
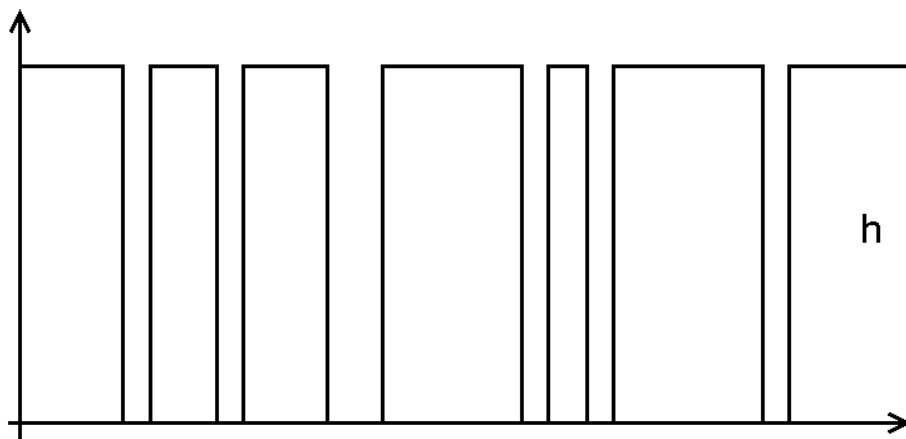*Fig. 7: Directional gradient of the previous profile.*

Fig. 8: Mask function h corresponding to the zeros of the previous gradient image.

Note that the transition is detected whatever its orientation (upward or downward). This can be observed in the previous figure when the thickness of a cell is equal to 1. In this case, two adjacent transitions appear.

Then, a mask function h is defined corresponding to the zero values of the previous one (Fig. 8):

$$h(x) = n \ \text{if } g_a(x) = 0$$
$$h(x) = 0 \ \text{ if not.}$$

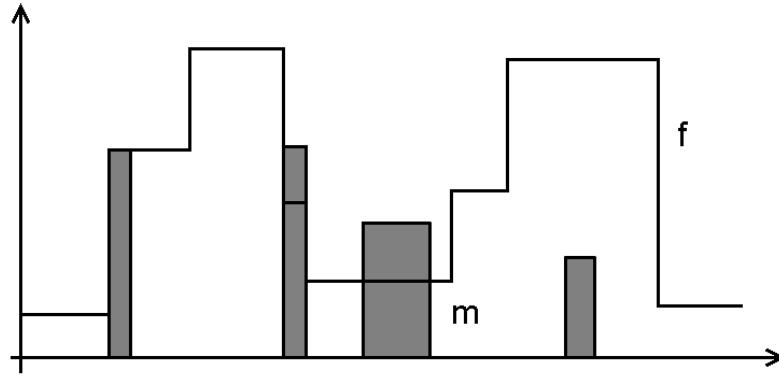(Remind that n is the highest label value of the partition).

Fig. 9: *Initial profile and marker function m. This function marks the cells which will be rebuilt and gives their new values.*

Consider now the marker set M, that is the set which indicates the cells of the partition to be rebuilt and, more precisely, the associated valued function m where each marker is given a grey value corresponding to the new label of every marked cell (Fig. 9).
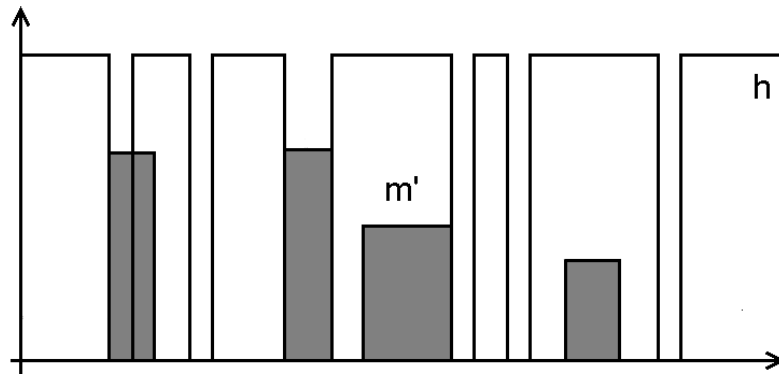

Fig. 10: *Dilation of the previous markers to insure that the directional reconstruction will be initiated.*
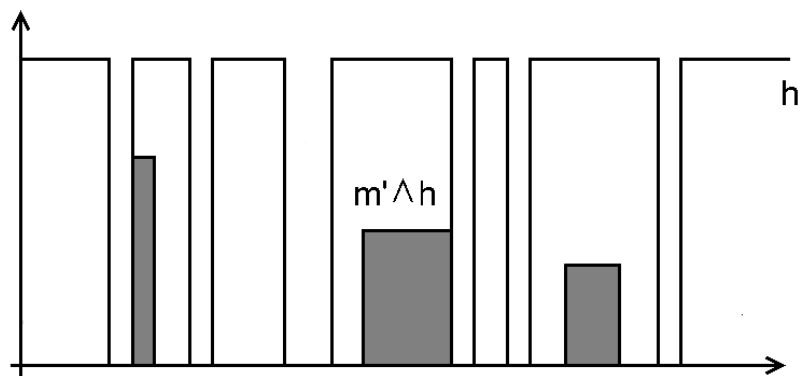

Fig. 11: *Intersection of the previous markers with the mask (reconstruction will occur inside the mask).*

It is necessary to perform an initial directional dilation of m in order to be sure that the following directional reconstruction will occur in any case, especially when a marker is placed at the boundary of a cell. The direction of the dilation is the same as the direction of the previous directional gradient. Note that this dilation will never mark unwanted cells for the reason explained below. We obtain a new marker function $m' = m \oplus \overset{\vee}{L}_\alpha$ (Fig.10).

Computing the function $(m' \wedge h)$ allows to keep only parts of m' which are less than h (Fig. 11).

A directional geodesic reconstruction $R_\alpha(h, m' \wedge h)$ of the function h can then be achieved using the function $m' \wedge h$. The direction used is obviously the direction of the previous gradient (Fig. 12). The intersection with h prevents the possible marking of adjacent cells which have been addressed above.
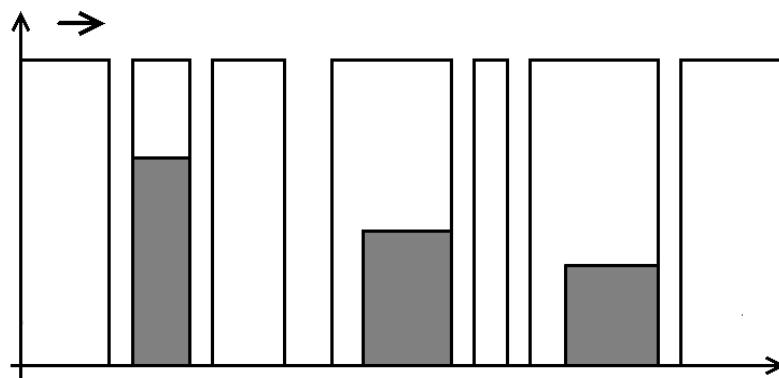

Fig. 12: Directional reconstruction of the mask.

Note that some markers may have disappeared in $m' \wedge h$. These markers correspond to thin cells in the current direction α. However, it is not at all a problem as we will see in the sequel. Note also that the purpose of the directional gradient is to provide a gap between adjacent cells in order to prevent the propagation of the reconstruction into non marked cells. This gap also prevents the initial directional dilation of the markers to overlap on the adjacent non marked cells.
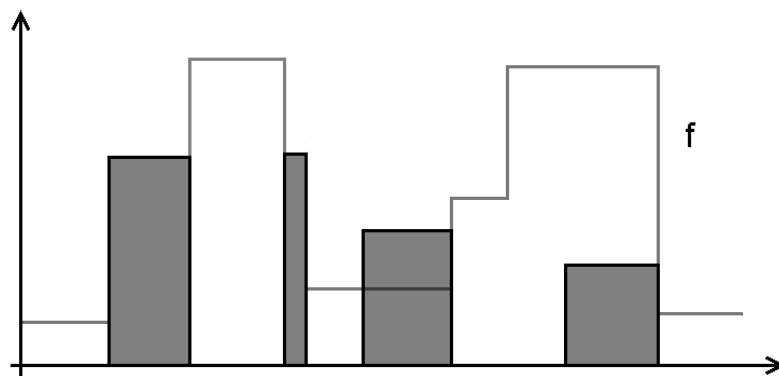

Fig. 13: Reintroduction of the markers which are too thin to be taken into account in the reconstruction.

At the end of this step, we just have to reintroduce the markers which have been suppressed during the process (Fig. 13). In fact, these markers have not contributed to the reconstruction in the current direction. This reintroduction is simply achieved by a supremum with the reconstruction:

$$R_a(h, m' \wedge h) \vee m$$

The resulting image can be used as a new marker for iterating the entire process in another directions until idempotence (Fig. 14).

Fig. 14: *Final reconstruction when all the directions are used iteratively.*

Note that it is not necessary that the marker function be composed of flat connected components. Indeed, if different values appear in a marker, the corresponding marked cell will be given a new value equal to the maximum of the marker values.

Two operators are implemented in Mamba. The first one, **cellsBuild** extracts from an initial partition image cells which are marked by a marker function and labels them with the label of the marker. The second one, **cellsExtract**, simply extracts cells which are marked by a binary marker.
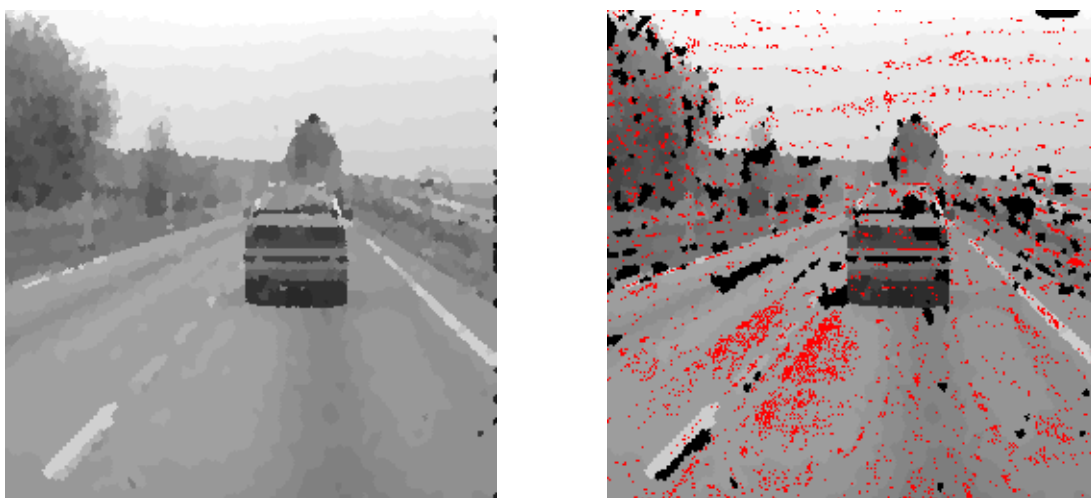

Fig.15: *Cells of the left partition image which are marked (marker set in red) are extracted, unmarked ones become background cells (right).*

Fig. 15 illustrates this transformation applied to extract marked cells of a partition image.
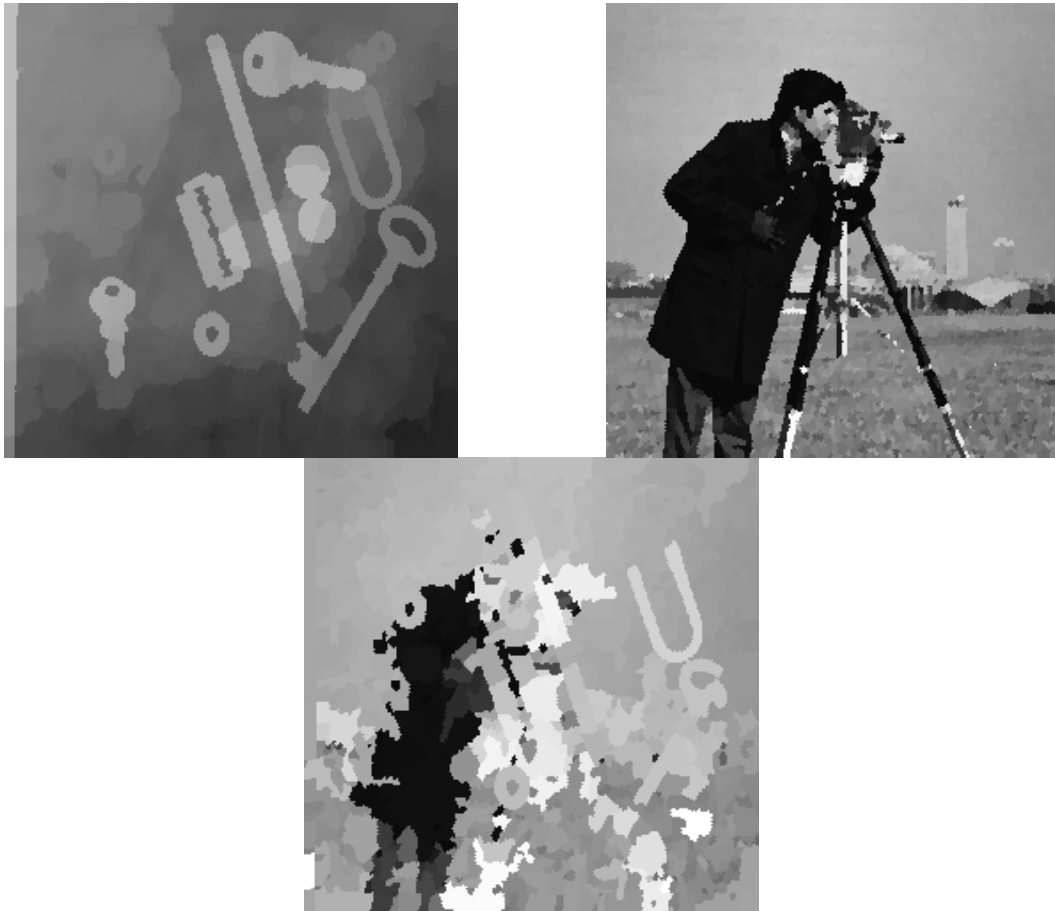
Fig. 16*: The upper right partition image is used as a marker to rebuild and to value the cells of the upper left partition. The result of the operation is the lower image.*

Fig. 16 shows a cells reconstruction of a partition image which uses as markers the cells of another partition image.
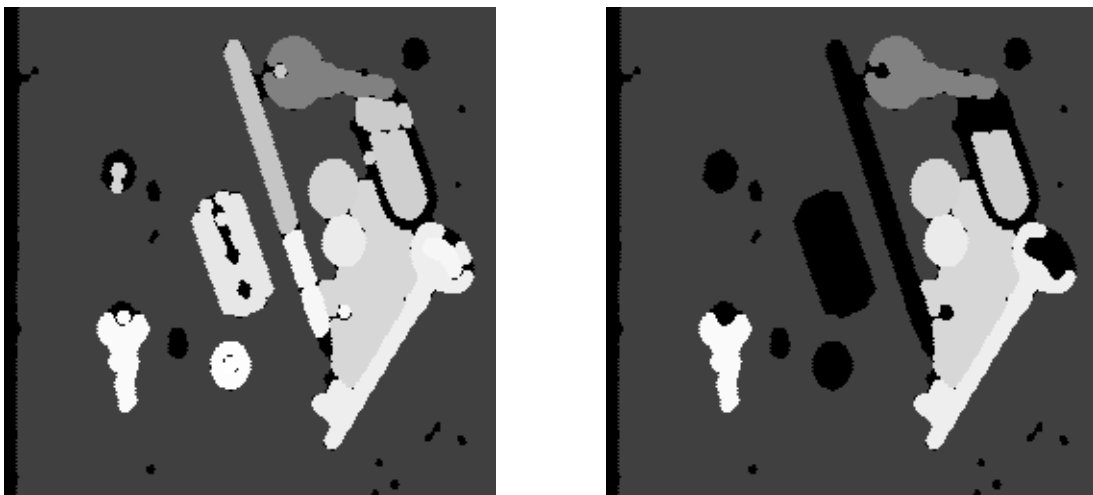

Fig. 17*: Opening by reconstruction of size 8 (right image) of the cells of the left partition image.*

Finally, by combining the cells erosion and the cells reconstruction, it is very simple to define a reconstruction opening operator which applies simultaneously on each cell of a partition.

This operator can be found in the appendix (it is named ***cellsOpenByBuild***) and Fig.17 illustrates it.

4. Operations on the partition graph

In the previous part, we considered the partition image as a patchwork of cells. The same operation is performed on every cell of the partition independently of the adjacent cells. Everything happens as if each cell was extracted, was processed and reinserted again in the partition.
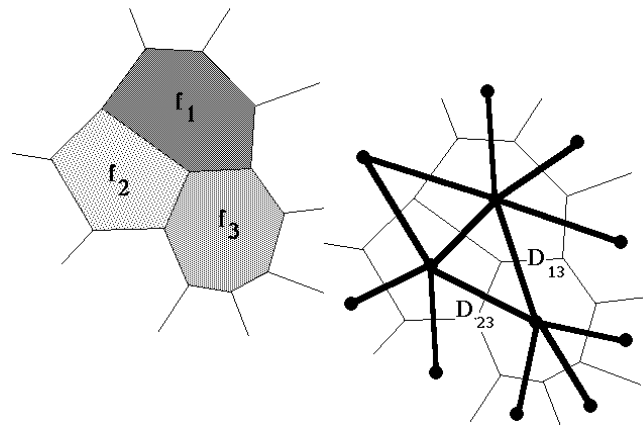

Fig. 18*: Representation of a partition as a valued graph.*

In this section, on the contrary, a partition image will be considered as a representation of a valued Delaunay graph (Fig. 18). Each cell of the partition corresponds to a valued vertex of the graph, the grey value of the cell being equal to the value of the vertex. Vertices corresponding to adjacent cells are connected by an edge in the graph.

4.1. Erosion and dilation of the partition graph

Elementary morphological operators can be defined on this graph ([8], [9]).

Let G(V,E) the Delaunay graph corresponding to a partition P. V is the set of vertices (that is the cells of the partition) and E the set of edges connecting the adjacent cells. The elementary dilation $\delta(G)$ of G is a new valued graph G'(V, E) with the same structure (same vertices and edges) but where the new value v_i' of each vertex i is given by:

$$v_i' = \sup_{k \in N_i}(v_k)$$

N_i being the neighborhood of i composed of i and of all its neighboring vertices (Fig. 19).

Similarly, we can define the eroded graph $G'' = \varepsilon(G)$ as the graph where the new value v_i'' of each vertex i is given by:

$$v_i'' = \inf_{k \in N_i}(v_k)$$

Dilations and erosions of larger sizes can be defined by iteration:

$$\delta^n(G) = \underbrace{\delta \circ .. \circ \delta(G)}_{n}$$

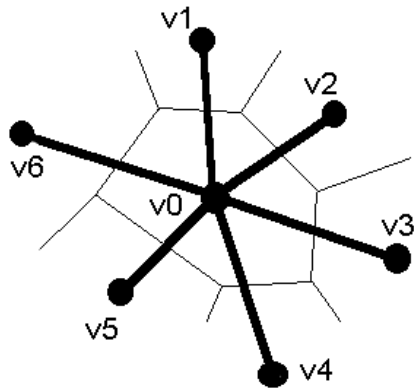$$\varepsilon^n(G) = \underbrace{\varepsilon \circ .. \circ \varepsilon(G)}_{n}$$

14

Fig. 19: *Neighborhood of a vertex v_0 in a graph which is considered for the definition of the dilation and the erosion.*

New partition images can be associated to the new graphs. If P is the partition corresponding to G and f, the corresponding image, we have:

$$f(x) = v_i \text{ if } x \in Y_i$$

Y_i being a cell of the partition P.

A new partition P' corresponds to the graph $G' = \delta(G)$. This partition is represented by a function f' defined by:

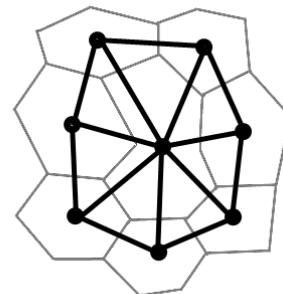$$f'(x) = v'_i = \sup_{k \in N_i}(v_k) \text{ if } x \in Y_i$$
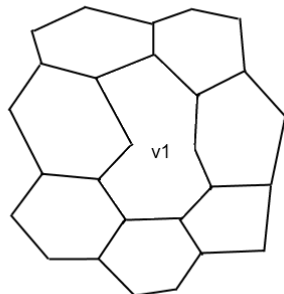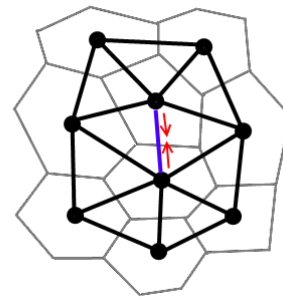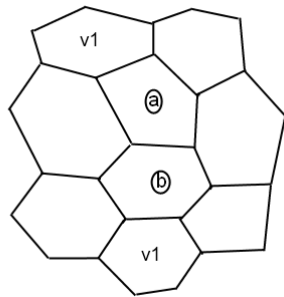

Fig. 20: *Each of the two adjacent cells (a) and (b) is adjacent to a maximal cell with the same value v_1 (upper left). The corresponding graph is shown in the upper right. After dilation of the graph, the cells (a) and (b) have merged (lower left) and the edge joining these two cells (in blue) has been contracted (lower right).*

However, the cells Y_i' of the partition P' are not identical to the cells Y_i of the initial partition P. Indeed, two or more adjacent cells in the partition P (with different values) may have the same value after dilation or erosion of the graph. So, these initial cells of the partition P will be merged into a single cell in P' as they were adjacent and they now have the same value. Therefore the partition P' is the representation of a new graph G^* which is different from the graph $G' = \delta(G)$. This new graph is obtained by merging some adjacent vertices and by connecting the resulting vertex to the remaining ones (Fig. 20). This operation is called edge contraction.

4.2. Defining graph operators on partitions

Performing morphological transforms on graphs is an interesting approach (albeit not really new as any operator applied on a digital image is in fact realized on a neighborhood graph). However, these transformations are seldom used (except in hierarchical segmentations). This is surely due to the fact that the step consisting in generating a graph from a partition image followed by the reverse step to build the resulting partition image is a tedious and time-consuming task, especially if we want to go back and forth between the image and the graph representations. Indeed, using a graph representation allows to deeply reduce the amount of data to be processed, but this also prevents to see the result of the operations, which is a very annoying drawback.
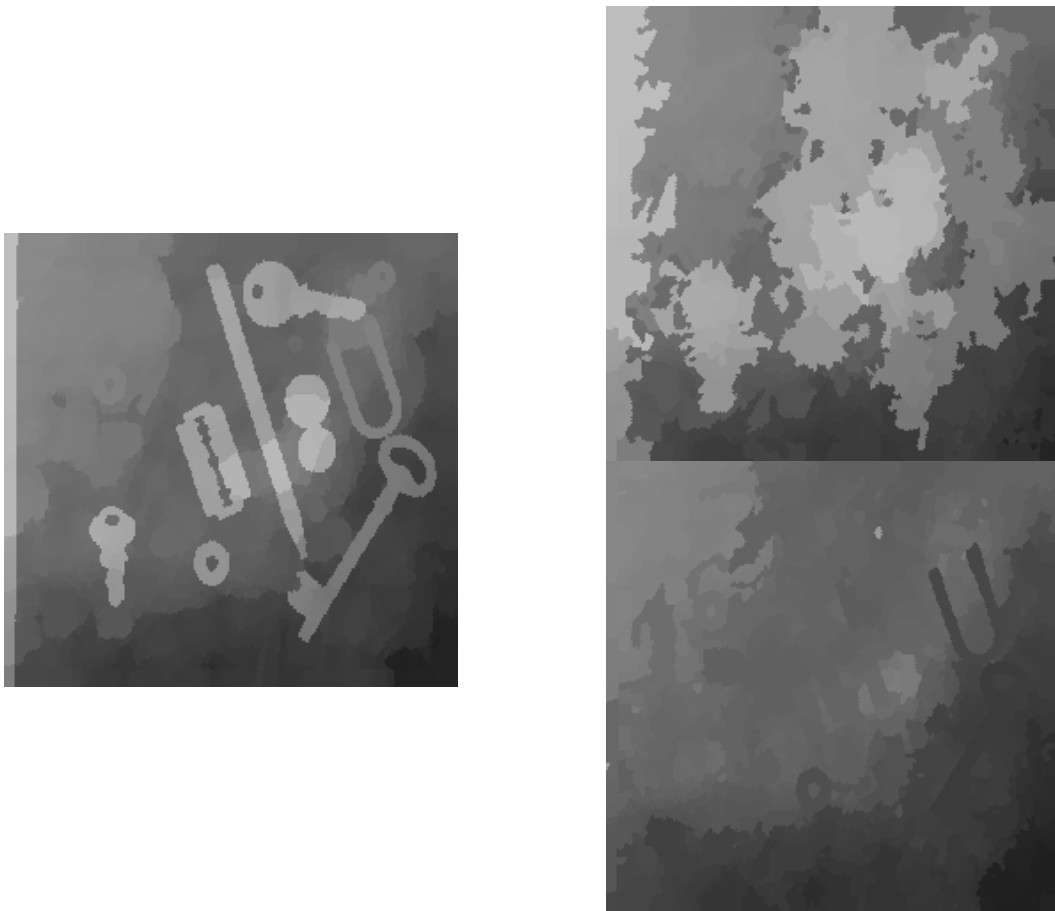

Fig. 21: Graph dilation (upper right) and erosion (lower right) performed directly on the partition image (left).

Fortunately, it is possible to define graph operators by acting directly on the partition image without the need to explicitly build the corresponding graph. Let P be the partition image

associated to the graph G. We can obtain directly the partition P' associated to the graph $G' = \delta(G)$ by the following operation:

- Dilate the function f representing the partition P.
- Perform a reconstruction of the partition P with $\delta(f)$ as marker:

$$P \rightarrow f \rightarrow R_P(f, \delta(f)) = f' \rightarrow P'$$

The dilation of the partition image f allows to mark each cell with the supremum value of all its neighbors. This partial marking is then propagated into the entire cell by means of the reconstruction.
In the same way, the partition P'' associated to the graph $G'' = \varepsilon(G)$ is obtained by a simple inversion of the function f at start followed by another inversion of the result (Fig. 21).

Iterating these operations, that is performing a size n dilation or erosion on the graph G, requires that the initial partition image f be kept in order to use it at each step of the operation. The algorithm is therefore the following (for dilation):

- Let f' = f (initialisation)
- Perform the operation $R_P(f, \delta(f')) = f'$ n times to get a size n dilation (in f').

If, on the contrary, we simply iterate the transformation on the partition produced by the previous step, the edge contraction associated to the operation induces a faster propagation of the extremal cells values (maxima in case of dilation, minima for the erosion), as shown at Fig. 22.
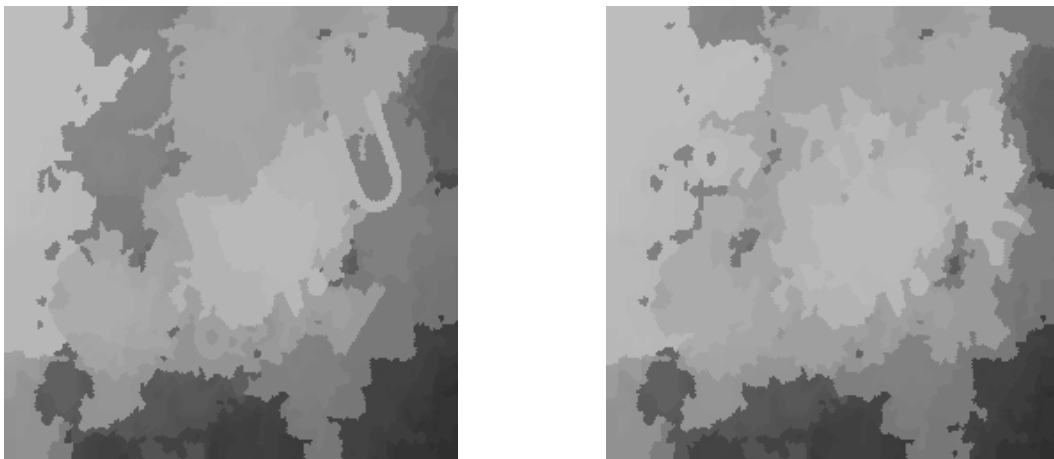
Fig. 22: *Difference between a classical size 2 graph dilation (left) and an iteration of two dilations performed on the partition (right).*

These two basic operators can be found in the appendix. They are named **partitionErode** and **partitionDilate**.

5. Conclusion

The operators which have been presented here have been originally defined and designed in order to be able to apply transformations and to collect information (size, shape,

topology, etc.) on the cells of a partition image as if each cell was processed individually. However, all the cells are transformed at the same time, which increases dramatically the processing speed by avoiding the excruciatingly slow individual analysis approach.

To achieve this, we showed that the definition of two new basic transformations applied on doublets of points ($\varepsilon_P^{\mathsf{L}}$ and η_P^{L}) allows to define on the cells of a partition P a whole range of anti-extensive operators: erosion, opening, Hit-or-Miss transform, thinnings.

Geodesic reconstruction algorithms applied to the cells of a partition have also been designed. These algorithms widely use the recursive directional reconstruction implementation which exists in the Mamba Image library. Although this implementation is slower than approaches based on hierachical queues, it is currently the only one which allows to cope with propagation issues which appear when the thickness and area of the cells are very small. However, this reconstruction remains very fast.

Remind also that these operators can be applied on any digital image, thanks to the extended definition of a partition image which was introduced and used for their design.

So, these operators are unquestionably useful when applied on partition images produced by many morphological transformations: watershed transform (especially the one which does not generate boundaries between the different catchment basins) [2], hierarchical segmentations (waterfalls, P algorithms, etc.)[6], [7], residual operators (in particular ultimate openings and granulometric functions) [4]. These operators bring efficient and fast tools for cells filtering, extraction and measurement. It would also be possible to use them to address some more complex issues: comparison and mixing of partitions, watermarking algorithms, markers extraction and tracking, features extraction...

Finally, we have revisited the basic morphological transformations on graphs. As, in most cases in image analysis, graphs are built from partitions, it is interesting to be able to perform graph operations while staying in the partition image representation of the graph. Indeed graph dilation and erosion can be obtained directly on the partition associated to the graph by means of the cell reconstruction defined in the first part of this document.

6. References

[1] Beucher N.: MAMBA documentation (MambaApi Reference Manual). Web documents, 2009-2011.
(available at http://www.mamba-image.org).
[2] Beucher N., Beucher S.: Hierarchical Queues: general description and implementation in MAMBA Image library. http://www.mamba-image.org / CMM-Armines-Mines ParisTech publication, april 2011.
(available at http://cmm.ensmp.fr/~beucher/publi/HQ_algo_desc.pdf).
[3] Beucher S.: Segmentation d'images et morphologie mathématique. Doctorate thesis, Paris School of Mines, june 1990, pp. 23-24.
(available at http://cmm.ensmp.fr/~beucher/publi/SB_these.pdf).
[4] Beucher S.: Numerical residues. Image and Vision Computing 25, ISMM05 Special Issue, pages 405-415, 2007.
(available at http://cmm.ensmp.fr/~beucher/publi/SB_NumRes_IVC_Rev1.pdf).
[5] Beucher S.: About a problem of definition of the geodesic erosion. CMM/Mines ParisTech publication, september 2011.
(available at http://cmm.ensmp.fr/~beucher/publi/GeodesicErode_eng.pdf).
[6] Beucher S.: Towards a unification of waterfalls, standard and P algorithms. CMM/Mines ParisTech publication, January 2012.

(available at http://cmm.ensmp.fr/~beucher/publi/Unified_Segmentation.pdf).

[7] Beucher S., Marcotegui B.: P algorithm, a dramatic enhancement of the waterfall transformation..CMM/Mines Paristech publication, september 2009.

(available at http://cmm.ensmp.fr/~beucher/publi/P-Algorithm_SB_BM.pdf).

[8] Burger F., Gandillot X., Treillard P.: Transformations morphologique sur un graphe, application à l'étude des lymphocytes. Internship report, Paris School of Mines / Ecole Polytechnique, 1982.

[9] Vincent L.: Graphs and Mathematical Morphology. Signal Processing, Vol. 16, n°4, april 1989, pp. 365-388.

7. Appendix

The various partition operators presented above have been implemented with the Mamba Image library. The Python source code is given below.

These operators have been gathered in a Pyhon module named ***partitions.py***. This module is available at http://cmm.ensmp.fr/~beucher/Mamba/partitions.py.

```python
"""
This module contains operators acting on partitions. Two types of operators are
defined: operators acting on each cell of the partition independently, or
operators considering each cell of the partition as a node of a weighted graph.
In the first case, each cell of the partition is considered as a binary set and
the defined operation is applied on each cell to produce a new transformation
(which is not necessarily a partition). In the second case, the partition is
considered as a graph and morphological operators are defined on this graph.
These operators provide morphological transformations on graphs, without the
need to explicitly define this graph structure from the partition.
"""

# Contributor: Serge BEUCHER

import mamba
import mambaComposed as mC

def cellsErode(imIn, imOut, n=1, se=mC.DEFAULT_SE, edge=mamba.FILLED):
    """
    Simultaneous erosion of size 'n' (default 1) of all the cells of the partition
    image 'imIn' with 'se' structuring element. The resulting partition is put in
    'imOut'.
    'edge' is set to FILLED by default.
    This operation works on 8-bit and 32-bit partitions.
    """

    imWrk1 = mamba.imageMb(imIn)
    imWrk2 = mamba.imageMb(imIn, 1)
    mC.dilate(imIn, imWrk1, n=n, se=se)
```

```python
    mC.erode(imIn, imOut, n=n, se=se, edge=edge)
    mamba.generateSupMask(imOut, imWrk1, imWrk2, False)
    mamba.convertByMask(imWrk2, imWrk1, 0, mamba.computeMaxRange(imIn)[1])
    mamba.logic(imOut, imWrk1, imOut, "inf")


def cellsOpen(imIn, imOut, n=1, se=mC.DEFAULT_SE, edge=mamba.FILLED):
    """
    Simultaneous opening of size 'n' (default 1) of all the cells of the partition
    image 'imIn' with 'se' structuring element. The resulting partition is put in
    'imOut'.
    'edge' is set to FILLED by default.
    This operation works on 8-bit and 32-bit partitions.
    """

    imWrk = mamba.imageMb(imIn)
    cellsErode(imIn, imWrk, n, se=se, edge=edge)
    mC.dilate(imWrk, imOut, n, se=se.transpose())


def cellsComputeDistance(imIn, imOut, grid=mamba.DEFAULT_GRID, edge=mamba.EMPTY):
    """
    Computation of the distance function for each cell of the partition image 'imIn'.
    The result is put in the 32-bit image 'imOut'.
    This operator works on hexagonal or square 'grid' and 'edge' is set to EMPTY by
    default.
    """

    imWrk1 = mamba.imageMb(imIn)
    imWrk2 = mamba.imageMb(imIn, 1)
    se = mC.structuringElement(mamba.getDirections(grid), grid)
    cellsErode(imIn, imWrk1, 1, se=se, edge=edge)
    mamba.threshold(imWrk1, imWrk2, 1, 255)
    mamba.computeDistance(imWrk2, imOut, grid=grid, edge=edge)
    mamba.addConst(imOut, 1, imOut)


def equalNeighbor(imIn, imOut, nb, grid=mamba.DEFAULT_GRID, edge=mamba.FILLED):
    """
    This operator compares the value of each pixel of image 'imIn' with the value
    of its neighbor in direction given by 'nb'. If the values are equal, the pixel
    is unchanged. Otherwise, it takes value 0.
    This operator works on hexagonal or square 'grid' and 'edge' is set to FILLED
    by default.
    This operator works for 8-bit and 32-bit images.
    """

    imWrk1 = mamba.imageMb(imIn)
    imWrk2 = mamba.imageMb(imIn, 1)
    mamba.copy(imIn, imWrk1)
    mamba.copy(imIn, imOut)
    mamba.supNeighbor(imIn, imWrk1, nb, 1, grid=grid, edge=edge)
    mamba.infNeighbor(imOut, imOut, nb, 1, grid=grid, edge=edge)
    mamba.generateSupMask(imOut, imWrk1, imWrk2, False)
    mamba.convertByMask(imWrk2, imWrk1, 0, mamba.computeMaxRange(imIn)[1])
    mamba.logic(imOut, imWrk1, imOut, "inf")


def nonEqualNeighbor(imIn, imOut, nb, grid=mamba.DEFAULT_GRID, edge=mamba.FILLED):
    """
```

This operator compares the value of each pixel of image 'imIn' with the value
of its neighbor in direction given by 'nb'. If the values are different, the pixel
is unchanged. Otherwise, it takes value 0.
This operator works on hexagonal or square 'grid' and 'edge' is set to FILLED
by default.
This operator works for 8-bit and 32-bit images.
```
"""

    imWrk1 = mamba.imageMb(imIn)
    imWrk2 = mamba.imageMb(imIn)
    imWrk3 = mamba.imageMb(imIn, 1)
    mamba.copy(imIn, imWrk1)
    mamba.copy(imIn, imWrk2)
    mamba.supNeighbor(imWrk1, imWrk1, nb, 1, grid=grid, edge=edge)
    mamba.infNeighbor(imWrk2, imWrk2, nb, 1, grid=grid, edge=edge)
    mamba.generateSupMask(imWrk1, imWrk2, imWrk3, True)
    mamba.convertByMask(imWrk3, imWrk1, 0, mamba.computeMaxRange(imIn)[1])
    mamba.logic(imIn, imWrk1, imOut, "inf")


def cellsHMT(imIn, imOut, dse, edge=mamba.EMPTY):
    """
    A Hit-Or-Miss transform is performed on each cell of the partition 'imIn'. 'dse'
    is a double structuring element (see thinthick.py module). The result is put
    in 'imOut'. 'edge' is set to EMPTY by default.
    """

    imWrk1 = mamba.imageMb(imIn)
    imWrk2 = mamba.imageMb(imIn)
    cse0 = dse.getStructuringElement(0)
    cse1 = dse.getStructuringElement(1)
    mamba.copy(imIn, imOut)
    mamba.copy(imIn, imWrk1)
    for dir in cse1.getDirections(withoutZero=True):
        equalNeighbor(imWrk1, imWrk2, dir, grid=cse1.getGrid(), edge=edge)
        mamba.logic(imOut, imWrk2, imOut, "inf")
    for dir in cse0.getDirections(withoutZero=True):
        nonEqualNeighbor(imWrk1, imWrk2, dir, grid=cse0.getGrid(), edge=edge)
        mamba.logic(imOut, imWrk2, imOut, "inf")


def cellsThin(imIn, imOut, dse, edge=mamba.EMPTY):
    """
    A simple thinning transform is performed on each cell of the partition 'imIn'.
    'dse' is a double structuring element (see thinthick.py module). The result
    is put in 'imOut'. 'edge' is set to EMPTY by default.
    """

    imWrk = mamba.imageMb(imIn)
    cellsHMT(imIn, imWrk, dse, edge=edge)
    mamba.sub(imIn, imWrk, imOut)


def cellsFullThin(imIn, imOut, dse, edge=mamba.EMPTY):
    """
    A full thinning transform is performed on each cell of the partition 'imIn'
    until idempotence. 'dse' is a double structuring element. The result
    is put in 'imOut'. 'edge' is set to EMPTY by default.
    """
```

```
    imWrk = mamba.imageMb(imIn)
    mamba.copy(imIn, imOut)
    v1 = mamba.computeVolume(imOut)
    v2 = 0
    while v1 != v2:
        v2 = v1
        for i in range(mamba.gridNeighbors(dse.getGrid())):
            cellsThin(imOut, imOut, dse, edge=edge)
            dse = dse.rotate()
        v1 = mamba.computeVolume(imOut)


def cellsBuild(imIn, imInOut, grid=mamba.DEFAULT_GRID):
    """
    Geodesic reconstruction of the cells of the partition image 'imIn' which are
    marked by the image 'imInOut'. The marked cells take the value of their
    corresponding marker. The result is stored in 'imInOut'.
    The images can be 8-bit or 32-bit images.
    'grid' can be set to HEXAGONAL or SQUARE.
    """

    imWrk1 = mamba.imageMb(imIn)
    imWrk2 = mamba.imageMb(imIn)
    imWrk3 = mamba.imageMb(imIn, 1)
    vol = 0
    prec_vol = -1
    dirs = mamba.getDirections(grid)[1:]
    while (prec_vol!=vol):
        prec_vol = vol
        for d in dirs:
            mamba.copy(imIn, imWrk1)
            mamba.copy(imIn, imWrk2)
            mamba.supNeighbor(imWrk1, imWrk1, d, 1, grid=grid)
            mamba.infNeighbor(imWrk2, imWrk2, d, 1, grid=grid)
            mamba.generateSupMask(imWrk2, imWrk1, imWrk3, False)
            mamba.convertByMask(imWrk3, imWrk1, 0, mamba.computeMaxRange(imIn)[1])
            mC.linearDilate(imInOut, imWrk2, d, 1, grid=grid)
            mamba.logic(imWrk2, imWrk1, imWrk2, "inf")
            mamba.buildNeighbor(imWrk1, imWrk2, d, grid=grid)
            mamba.logic(imWrk2, imInOut, imInOut, "sup")
        vol = mamba.computeVolume(imInOut)


def cellsExtract(imIn, imMarkers, imOut, grid=mamba.DEFAULT_GRID):
    """
    Geodesic reconstruction and extraction of the cells of the partition image
    'imIn' which are marked by the binary marker image 'imMarkers'. The marked
    cells keep their initial value. The result is stored in 'imOut'.
    The images can be 8-bit or 32-bit images.
    'grid' can be set to HEXAGONAL or SQUARE.
    """

    imWrk1 = mamba.imageMb(imIn)
    imWrk2 = mamba.imageMb(imIn)
    mamba.copy(imIn, imWrk1)
    mamba.convertByMask(imMarkers, imWrk2, 0, mamba.computeMaxRange(imIn)[1])
    mamba.logic(imIn, imWrk2,imOut, "inf")
    cellsBuild(imWrk1, imOut, grid=grid)
```

```python
def cellsOpenByBuild(imIn, imOut, n=1, se=mC.DEFAULT_SE):
    """
    Opening by reconstruction of size 'n' (default 1) of the partition image
    'imIn'. 'se' defines the structuring element. The result is put in 'imOut'.
    The images can be 8-bit or 32-bit images.
    """

    imWrk = mamba.imageMb(imIn)
    mamba.copy(imIn, imWrk)
    cellsErode(imIn, imOut, n=n, se=se)
    cellsBuild(imWrk, imOut, grid=se.getGrid())


def partitionErode(imIn, imOut, n=1, grid=mamba.DEFAULT_GRID):
    """
    Graph erosion of the corresponding partition image 'imIn'. The size is given
    by 'n'. The corresponding partition image of the resulting eroded graph is
    put in 'imOut'.
    'grid' can be set to HEXAGONAL or SQUARE.
    """

    imWrk = mamba.imageMb(imIn)
    mamba.negate(imIn, imWrk)
    mamba.copy(imWrk, imOut)
    se = mC.structuringElement(mamba.getDirections(grid), grid)
    for i in range(n):
        mC.dilate(imOut, imOut, se=se)
        cellsBuild(imWrk, imOut, grid=grid)
    mamba.negate(imOut, imOut)


def partitionDilate(imIn, imOut, n=1, grid=mamba.DEFAULT_GRID):
    """
    Graph dilation of the corresponding partition image 'imIn'. The size is given
    by 'n'. The corresponding partition image of the resulting dilated graph is
    put in 'imOut'.
    'grid' can be set to HEXAGONAL or SQUARE.
    """

    imWrk = mamba.imageMb(imIn)
    mamba.copy(imIn, imOut)
    mamba.copy(imIn, imWrk)
    se = mC.structuringElement(mamba.getDirections(grid), grid)
    for i in range(n):
        mC.dilate(imOut, imOut, se=se)
        cellsBuild(imWrk, imOut, grid=grid)
```