Morphological Algorithms

Luc Vincent

Division of Applied Sciences, Harvard University
Pierce Hall, Cambridge, MA 02138, USA*

Published as Chapter 8 of "Mathematical Morphology in Image Processing", *E. Dougherty, Editor, Marcel-Dekker, New York, September 1992, Pages 255–288.*

1 Introduction

This chapter is concerned with the efficient implementation of "low-level" morphological transformations [25, 38]. The qualifier "low-level" used here means that we deal with the implementation of transformations which serve as elementary bricks when solving practical image analysis problems. This does not mean that these transformations are simple, or cannot be decomposed into simpler ones: on the contrary, some of the operations considered in this chapter (e.g., skeletons, watersheds, propagation functions) are complex, both to define and to compute! However, from a user's perspective, these transformations share the characteristics of being easily and intuitively understandable: for example, watersheds extract from a gray-level image the crest-lines which are located between the minima, top-hat transformations extract thin and light (or dark) regions, skeletons reduce binary shapes to their medial axes, etc.

Solving a moderately complex image analysis application by morphological methods often involves the concatenation of several tens or hundreds of low-level transformations [50]. This is the reason why each of these elementary bricks should be implemented as efficiently as possible. This task can be approached via various algorithmic techniques, the majority of which shall be described in this chapter. Each category of techniques is characterized by its advantages and drawbacks, and illustrated using transformations for which it is particularly suited. For more details, see [34, 46, 37].

The present section is first concerned with the notations that will be used throughout the chapter. A particular transformation, the *distance function,* is also recalled, since it is used as a "leitmotiv" to illustrate how the described families of algorithms work. The characteristics one expects morphological algorithms to be equipped with are then briefly discussed.

Section 2 is devoted to the most classical morphological algorithms, namely the *parallel* ones. As explained below, these algorithms turn out to be rather inefficient on conventional computers. A first step towards the implementation of fast morphological algorithms was made by introducing the *sequential* methods. They are presented and illustrated in Section 3.

*Present affiliation: Xerox Imaging Systems, 9 Centennial Drive, Peabody MA 01960, USA

Although sequential techniques serve very well for the computation of transformations such as distance function, granulometry function or geodesic reconstruction (see Section 3), they remain inefficient in many cases, since they involve numerous scannings of the entire image. To get rid of this problem, new scanning techniques have been introduced: the algorithms relying on them are such that, throughout the computation of a given morphological transform, only those pixels likely to be modified are taken into account. Such algorithms are based on contours and can be divided up into two families: the chains and loops propagation algorithms [34], which constitute the topic of Section 4 and the queue algorithms [46], discussed in Section 5. Both families have been recently introduced in the morphology world and constitute one of the best possible choices for implementing complex transformations on conventional computers. Not only do these methods lead to faster algorithms, they are also extremely flexible and usually produce more accurate results. We shall illustrate their use by the computation of such transformations as propagation functions, Euclidean distance functions, skeletons and watersheds. Lastly, the conclusion summarizes the qualities and drawbacks of these categories of algorithms and provides some guidelines as to what methods should be used for a given purpose.

1.1 Discrete images and grids, notations

In the following, we consider binary and grayscale images I as mappings from a rectangular domain $D_I \subset \mathbb{Z}^2$ into $\mathbb{Z}$. A binary image may take only values 0 and 1, and is often reduced to the set of its *feature pixels,* i.e., pixels with value 1. Many of the algorithms described below extend to n-dimensional spaces, but for the sake of simplicity, they will always be presented for 2-D images.

The underlying grid $G \subset \mathbb{Z}^2 \times \mathbb{Z}^2$ defines the neighborhood relations between pixels. G is usually a square grid (of 4- or 8-connectivity) or a hexagonal one (see Fig. 1). $N_G(p)$ denotes the set of the neighbors of a pixel $p \in \mathbb{Z}^2$ according to grid G:

$$N_G(p) = \{q \in \mathbb{Z}^2 \mid (p, q) \in G\}.$$

The discrete distance associated with G is denoted d_G: $d_G(p, q)$ is the minimal length of the paths of G connecting p to q.

In the present chapter, we mostly use the hexagonal grid (6-connectivity). Indeed, the discrete distance it induces, called *hexagonal distance* and denoted d_6, is more isotropic than the city-block distance d_4 or the chessboard distance d_8, respectively induced by the square grid in 4- and 8-connectivity [8]. More importantly: the hexagonal grid is a *triangulation* and thus satisfies the *digital Jordan property* [34, page 61] according to which every nondegenerate simple loop separates the digital plane $\mathbb{Z}^2$ into two different connected components. As illustrated by Fig. 2, this is not true for square grids and causes endless practical difficulties. For example, when dealing with square grids, consistency makes it often necessary to use 8-connectivity for the objects and 4-connectivity for the background (or vice-versa) [46]. For these reasons, morphologists often prefer the hexagonal grid. Its elementary vectors are denoted $\vec{u}_0$, $\vec{u}_1$,...,$\vec{u}_5$ and are illustrated by Fig. 3. Note however that all the algorithms described below extend to any kind of discrete grid.

The algorithms themselves are described in a pseudo-code which bears similarities to *C* and *Pascal.* It makes use of a certain number of keywords and symbols which are summarized in Table 1. Some shortcuts like:

Repeat until stability {...

or

For every pixel p' *in* $N_G(p)$ {...

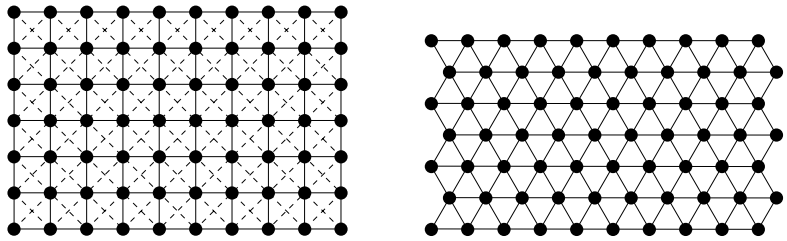


Figure 1: *Square (a) and hexagonal (b) digital grids. The former one can be considered either in 4- or 8-connectivity.*

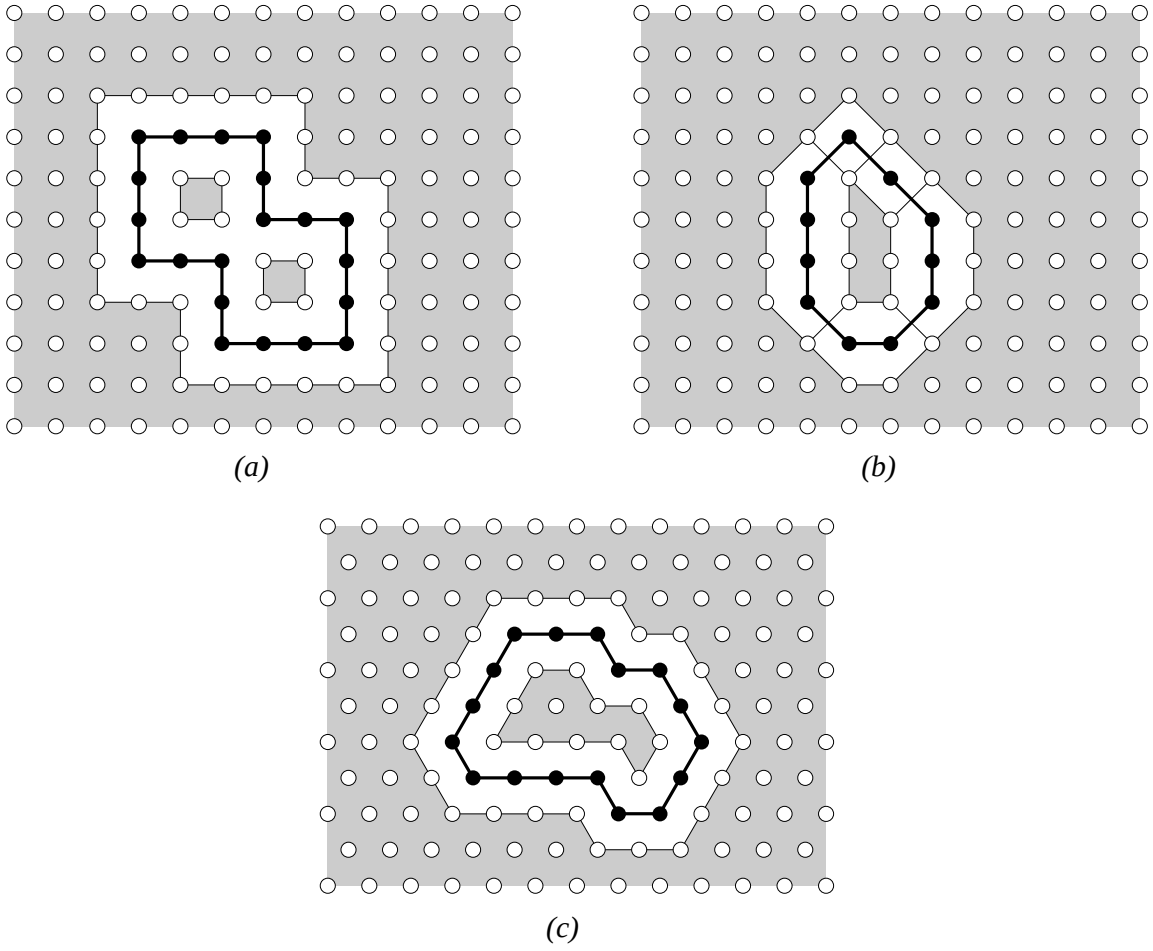


Figure 2: *Square grids of 4- or 8-connectivity do not satisfy the digital Jordan property, in contrast with the hexagonal grid (c). Indeed, the simple nondegenerate loop drawn in (a) (4-connectivity) separates the discrete plane into three connected components whereas that of (b) (8-connectivity) does not separate anything!*

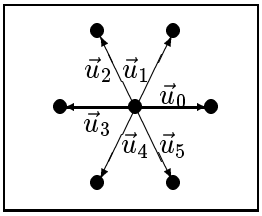


Figure 3: *The 6 elementary vectors of the hexagonal grid.*

$=, \neq, <, >, \leq, \geq$	Comparisons of values
$\leftarrow$	Assignment
{, }	Beginning and end of group of instructions
;	End of instruction
$/\star, \star/$	Beginning and end of comments
If, then, else	Logical tests
For... to; While; Repeat until	Classical loops
true, false	Logical values

Table 1: *Symbols and keywords of the pseudo-code used for algorithm descriptions.*

will also be used. Instructions specific to the type of image scanning used by the algorithm being discussed will be introduced as needed.

1.2 The distance function

Throughout the chapter, a particular transformation called the *distance function* [32, 8] is used to illustrate the four families of algorithms described. This transformation is indeed very typical and gives rise to several different implementations. The distance function $dist_X$ of a set $X \subset \mathbb{Z}^2$ associates with each pixel of X its distance to the background:

$$dist_X \left(\begin{array}{lcl} X & \longrightarrow & \mathbb{Z} \\ p & \longmapsto & \min\{d_G(p,q) \mid q \notin X\} \end{array} \right) \tag{1}$$

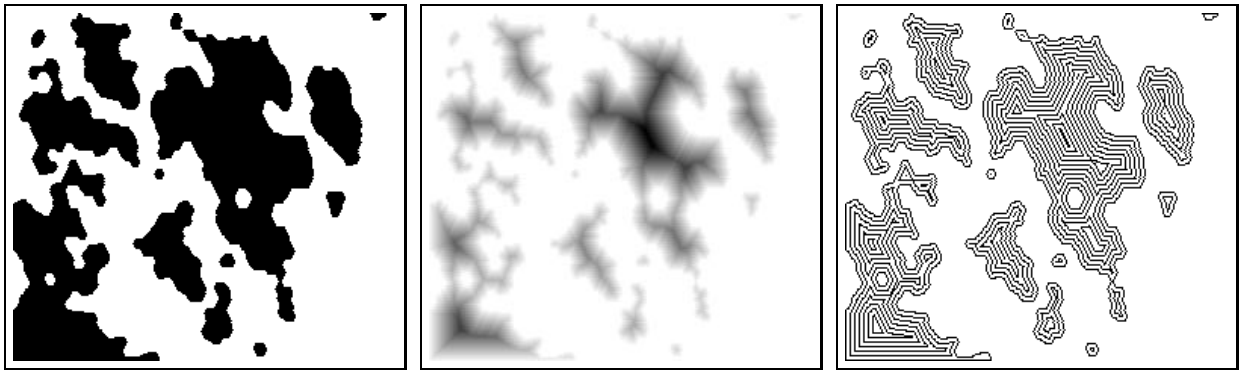
The distance function d_I of a binary image I is equivalent to that of its set of *feature pixels,* i.e., pixels with value 1. In addition, we put conventionally: $\forall p \in D_I, I(p) = 0 \Rightarrow dist_I(p) = 0$. An example of distance function is shown in Fig. 4.

1.3 Estimating the quality of a morphological algorithm

The performance of a morphological algorithm may be defined using three main criteria: speed, accuracy and flexibility.

Speed

This is a crucial issue in the field of image analysis. Indeed, on the one hand, an application program is often designed to be used routinely, either on a large amount of data (e.g., in medicine), or daily (e.g., in quality control). It is then unacceptable for the execution time to be larger than a specified upper bound. On the other hand, even during the solution of a given image analysis problem, many different possibilities have to be considered; for each of them, many transformations



(a) original image *(b) distance function* *(c) level lines of this function*

Figure 4: Example of hexagonal distance function.

have to be used, often repeatedly, with parameter adjustments, filter modifications, etc. It is therefore extremely important for the image analyst to have fast algorithms at his or her disposal: it considerably speeds up this development step and even enables to explore ideas which could not be considered otherwise. For example, until recently, the use of the watershed transformation [11, 3] was impossible in practice because of its prohibitive computation time. However, the appearance of the most recent specialized architectures (e.g., the *Quantimet 570* of *Leitz*) and algorithms [46, 52] (see Section 5) has moved it to one of the highest ranking of morphological segmentation tools.

Accuracy

An algorithm should of course give results that are as accurate as possible. In fact, most of the time, the result is expected to be totally exact. However, the definition of some transformations—like skeletons (see Section 5))—is sometimes not well adapted to the discrete framework. The algorithms for computing such transformations should then be designed to produce results "as close as possible" to the continous one. Morphological algorithms are also expected to avoid some of the aberrations associated with the use of discrete grids, like the "cone effect" (see Fig. 5.b). Lastly, one often tries to compute morphological transformations in an isotropic fashion. This involves resorting to discrete distances d closer to the Euclidean one than d_G (see Section 4).

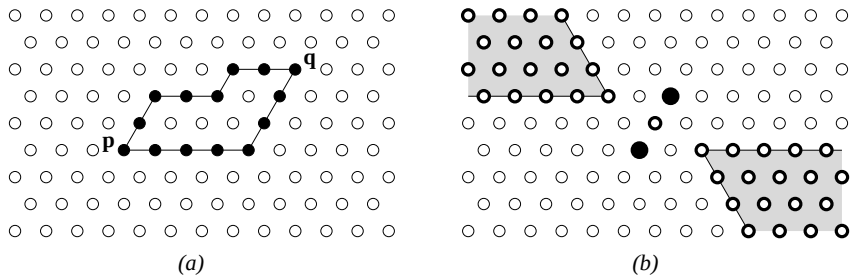


Figure 5: Some difficulties in discrete spaces: (a) there exist several paths of minimal length between two pixels; (b) the set of the pixels which are equidistant to two given connected components may be a thick area: here, according to the hexagonal distance, all the bold pixels (gray area) are equidistant to the two black ones!

Flexibility

By flexibility, we mean any of the following:

- the algorithm is adaptable to other grids,
- it works in both the Euclidean and geodesic [22] cases,
- it allows one to produce several transforms close to one another,
- it is adaptable to several metrics.

Flexible algorithms are very interesting in that they spare the energy of the programmer and reduce the implementation costs.

However, most algorithms cannot be extremely fast, accurate and flexible at the same time. Improving one of these characteristics is generally done to the detriment of the two remaining ones. For example, increasing the accuracy of an algorithm mostly requires additional tests and computations which affect its speed... On the other hand, morphological operations are not implemented in the same way on two different computers: on specialized architectures, complex transformations like skeletons and watersheds will be implemented using built-in thinning and thickening capabilities. However, such algorithms would be terribly slow on classical computers, where their execution times could be close to a couple of hours!

1.4 Image structures and how to access them

The most common data structure to represent and process binary and grayscale images is the two-dimensional array of pixels, all of which are either 1, 8 or 16 bits. This type of structure is very simple and enables a fast access to the neighbors of a given pixel, no matter what discrete grid is used. Several attempts have been made in morphology to manipulate images stored using different data structures, e.g., quadtrees and octrees [33, 4], interval coding [29], structures stemming from computational geometry, like polygons [30], etc. However, as explained in [46, page 22], none of these structures is really adapted to the implementation of morphological transformations.

This is the reason why, in the following, we only consider images stored as arrays of pixels. We always assume that the number of bits per pixel is sufficient and we do not account for edge effects. Indeed, as explained in [46, chapter 2], they are usually easy to cope with by giving to the pixels of the frame a particular value, usually 0, $-\infty$ or $+\infty$. Here, the images under study are considered to be defined in the entire space $\mathbb{Z}^2$ and to take value 0 outside of their definition domain, unless otherwise mentioned. Additionally, special data structures like loops or queues will be used to manipulate these arrays of pixels efficiently.

2 Parallel Algorithms

This category of algorithms is the most common and classical one in the field of morphology. A parallel algorithm typically works as follows: given an input image I, the pixels of I are scanned and the new value of the current pixel p is determined from that of the pixels in a given neighborhood $N(p)$ of p. In doing this, the following constraint is satisfied:

> *The new pixel values are written in an output image J different from I.*

J is then copied into I, and additional image scannings are performed until a given criterion is fulfilled, or until stability is reached.

Since I is different from J, its pixels can actually be scanned *in an arbitrary order.* In particular, one can imagine parallelizing the processing on some image parts, or even on all pixels, as is done by some specialized architectures. Hereafter, a "parallel scanning" is introduced by a sentence like:

For every pixel p of D_I, do $\{\ldots$

The parallel algorithm to determine the distance function of a binary image I in grid G is given below in a pseudo-code.

Algorithm: parallel distance function

- $\left\{ \begin{array}{ll} - & \texttt{input:} \quad I, \texttt{ binary image,} \\ - & \texttt{output:} \quad J, \texttt{ grayscale image defined on } D_I;\ J \neq I. \end{array} \right.$
- `Repeat until stability {`
  `For every pixel` $p \in D_I$ `{` `/* actual parallel scanning */`
  `If` $I(p) = 1$ `then` $J(p) \leftarrow \min\{I(q), q \in N_G(p)\} + 1$;
  `}` `Copy image` J `in` I;
  `}`

This algorithm is illustrated on Fig. 6. One is easily convinced that the number of scannings it requires is proportional to the largest computed distance. More generally, parallel algorithms usually require a large number of complete image scannings, sometimes several hundred! Therefore, although these algorithms are particularly suited to some architectures, they are definitely not adapted to conventional computers.

The basic parallel algorithms (dilations, erosions, distance function, etc) easily extend to any grid and to n-dimensional images [16], but here again, prohibitive execution times limit their practical interest. Moreover, the parallel computation of some more complex transformations like skeletons [9] and skeletons by influence zones (SKIZ) [21] is usually achived via iterations of parallel thinnings or thickenings. These operations involve *structuring elements* [38], i.e., pixel templates used as probes (see Golay's alphabet [17]). This is the reason why their adaptation to other grids requires cumbersome neighborhood analyses. This remark is even more true when it comes time to extend these algorithms to n-dimensional data! Furthermore, although some of them can bring Euclidean distances into play [54], in many cases, the very local way parallel algorithms work leads to approximative transforms (e.g., for skeletons).

3 Sequential Algorithms

In an attempt to reduce the number of scannings required for the computation of an image transform, sequential or recursive algorithms have been proposed [31]. They rely on the following two principles:

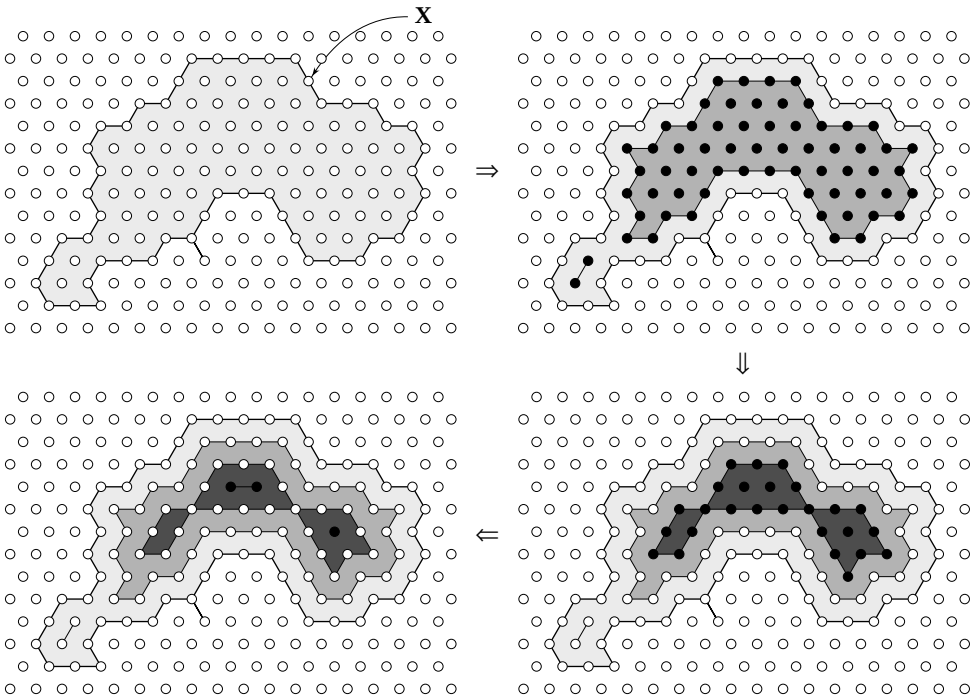


Figure 6: *Successive steps involved in the parallel computation of a distance function.*

- *the image pixels are scanned in a predefined order, generally raster (left to right and top to bottom) or anti-raster,*
- *the new value of the current pixel, determined from the values of the pixels in its neighborhood, is written directly* in the same image, *so that it is taken into account when determining the new values of the as yet unconsidered pixels.*

Note that here, unlike for parallel algorithms, the scanning order is essential. A number of transformations which can be obtained sequentially are described in [23]. In the following, a sequential scanning will be introduced by:

Scan D_I in raster order {
  Let p be the current pixel; ...

To compute a distance function sequentially, a raster scanning followed by an anti-raster one are sufficient [32]: out of the original binary image I, the raster scanning creates an intermediate gray-level image, whose highest values are located in the lower left part of the connected components of I. Each feature pixel p of I is assigned the length of (one of) the shortest path P between p and the background, with the following constraint: *every path element $\overline{xy}$ of P has either a strictly positive vertical component (i.e., $\overline{xy}$ is pointing upwards) or a zero vertical component and a positive left component.* This is why a second scanning, of anti-raster type, is necessary to get from I' an actual distance function. This algorithm is given below for the hexagonal distance:

Algorithm: sequential distance function

- input: I, binary image;
  /⋆ The distance function is computed directly in I ⋆/
- Scan D_I in raster order {
  Let p be the current pixel;
  If $I(p) \neq 0$ then $I(p) \leftarrow \min\{I(p+\vec{u}_1)+1, I(p+\vec{u}_2)+1, I(p+\vec{u}_3)+1\}$;
  }
- Scan D_I in anti-raster order {
  Let p be the current pixel;
  If $I(p) \neq 0$ then $I(p) \leftarrow \min\{I(p), I(p+\vec{u}_4)+1, I(p+\vec{u}_5)+1, I(p+\vec{u}_0)+1\}$;
  }

In all cases, the above algorithm only requires *two* image scannings. In comparison with the parallel one, its thus constitutes a clear improvement! This is even more true since this sequential distance function algorithm has been integrated into the chip PIMM1 described in [20]. In fact, sequential algorithms often constitute one of the best possible choices. Some of the operations the implementation of which they are best suited for are mentioned below. In addition, sequential algorithms such as the distance function one are easily extended to n-dimensional spaces [7] and can be adapted to better discrete distances [8, 10].

Granulometry function, grayscale dilations and erosions

Let I be a discrete binary image and let $(B_n)_{n\geq 0}$ be a family of structuring elements such that the B_n's are the homothetics of a given convex set B. The following equations hold:

$$\begin{array}{rcl} B_0 & = & \{\vec{o}\} \\ B_1 & = & B \\ \forall n \geq 1, B_{n+1} & = & B_n \oplus B, \end{array} \tag{2}$$

$\oplus$ denoting the Minkowski addition. Let also γ_C denote the morphological opening with respect to structuring element C [39]. The *granulometry function* of I with respect to (B_n), denoted here by $g(I)$, associates with each pixel $p \in D_I$ the smallest integer k such that $\gamma_{B_k}(p) = 0$. In other words:

$$g(I) \left(\begin{array}{rcl} D_I & \longrightarrow & \mathbb{Z} \\ p & \longmapsto & \min\{k \in \mathbb{Z}^+ \mid \gamma_{B_k}(p) = 0\} \end{array} \right. \tag{3}$$

Just like the distance function of a binary image I is the "pile" of its successive erosions, the granulometry function is nothing byt the pile of its successive openings with respect to the B_n's. This means that by thresholding $g(I)$ at value k, one simply gets the binary opening of I with respect to element B_{k-1}. The histogram of $g(I)$ provides the granulometric analysis of I. Thus, although this transformation is not very exciting from a theoretical point of view, its great interest comes from the fact that it is possible to obtain it very quickly by using sequential methods. For example, the granulometry function shown in Fig. 7 was obtained in three seconds on a Macintosh II. A very similar algorithm can be used to determine grayscale dilations and erosions by the B_n's.

Morphological shadowing

This is another case where sequential methods outperform all other techniques. The shadowing of a grayscale image is realized through dilation by a ray of the 3-D space $\mathbb{Z}^2 \times \mathbb{Z}$. This is often

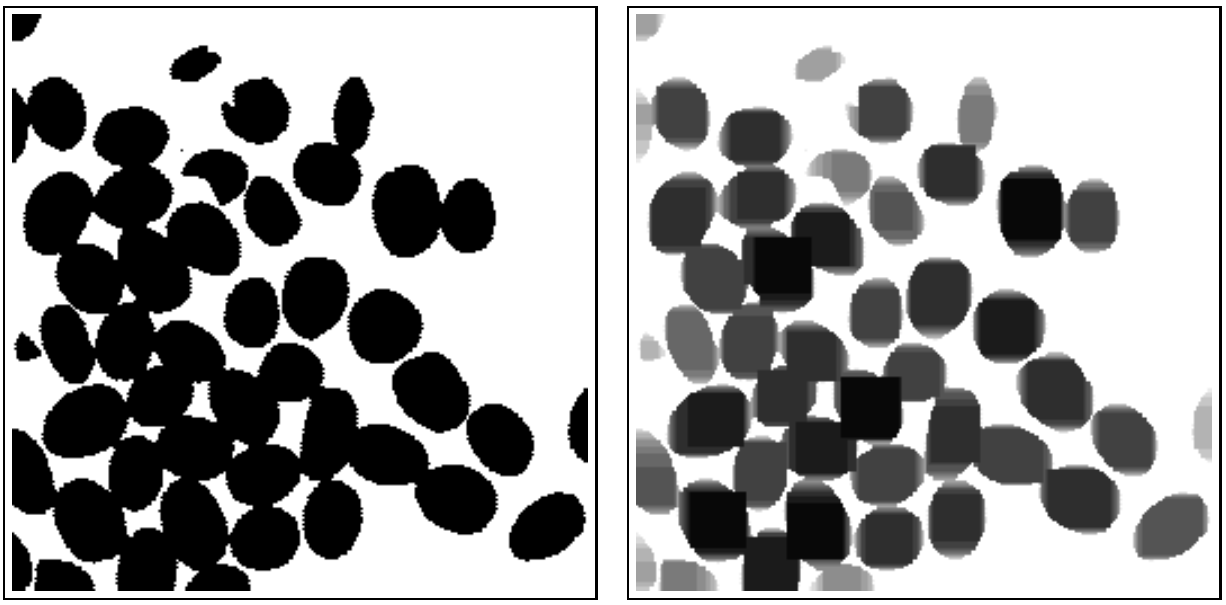


Figure 7: A binary image and its corresponding granulometry function with respect to a family of squares.

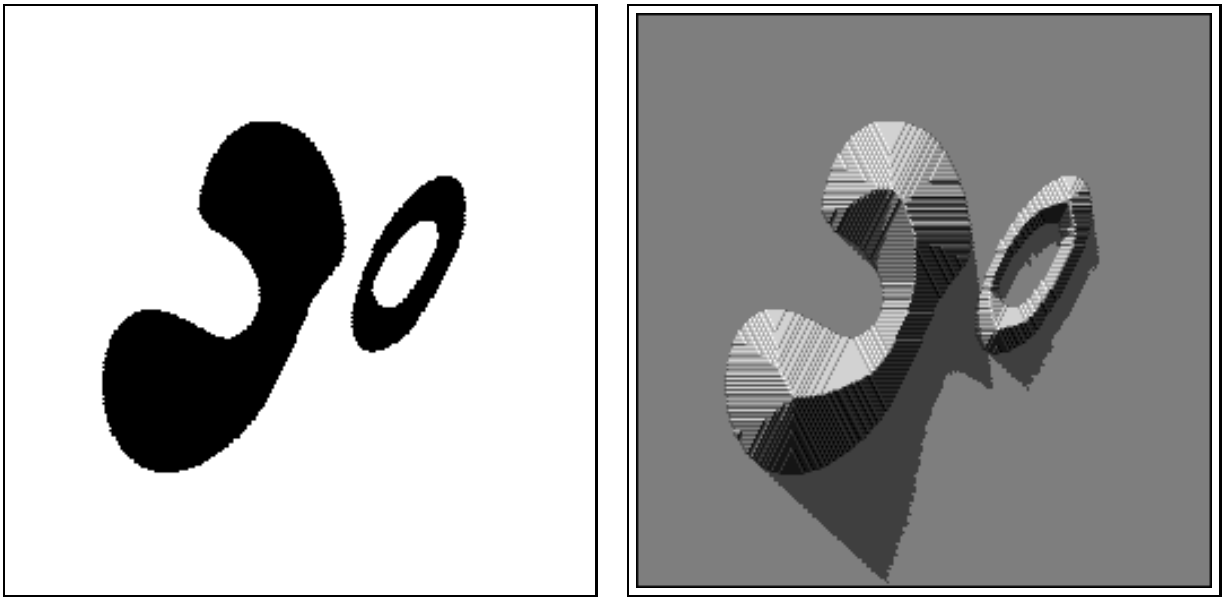
interesting for visualization purposes and can be efficiently implemented thanks to a recursive algorithm described in [45]. An example of morphological shadowing of a distance function is shown in Fig. 8. One can notice in this example that the crest-lines of the distance function have been highlighted, thereby leading to a family of methods for computing skeletons.

Geodesic reconstruction

In geodesic morphology [22], a transformation called *reconstruction* turns out to be of immense interest. Given two (binary or grayscale) images f and g such that $g \leq f$ (i.e., for every pixel p, $g(p) \leq f(p)$), the reconstruction $R_f(g)$ of f from g is obtained by dilating g geodesically under f until stability is reached. f is called the *mask* image whereas g is the *marker*. More precisely, denote by B the elementary ball of the grid being used. For example, B is hexagon H in 6-connectivity, 5-pixel square S_1 in 4-connectivity or 9-pixel square S_2 in 8-connectivity (see Fig. 9). Let δ_B stand for the dilation with respect to B and $\wedge$ be the pointwise minimum. The reconstruction of f from g is obtained by iterating the following operation until stability is reached:

$$g \leftarrow \delta_B(g) \wedge f. \tag{4}$$

In the binary case, reconstructing f from g allows us to extract those connected components of binary image f which contain at least a pixel of g [50]. This extends to the grayscale case in terms of peaks: as illustrated by Fig. 10, only those peaks of f that are marked by g are preserved through reconstruction. From Eq. (4), it is straightforward to derive a parallel algorithm for binary and grayscale reconstruction, but it is particularly inefficient on standard equipment. Here, it is much preferable to resort to a sequential algorithm [23, 46]: like the sequential distance function algorithm previously described, the present one works by propagating information downwards in a raster scanning and then upwards in an anti-raster scanning. Here however, as explained later, these raster and anti-raster scannings have to be iterated until stability is reached (see Fig. 12). This algorithm is described below for the hexagonal case. It works for both the grayscale and the binary case and usually only requires around 10 complete image scannings:



(a) original binary image *(b) distance function with artificial shadowing*

Figure 8: Shadowing of the distance function of a binary shape.

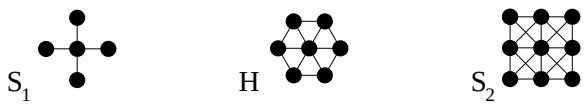


Figure 9: Elementary ball in 4-, 6- and 8-connectivity respectively.

Algorithm: sequential reconstruction

- $\left\{ \begin{array}{ll} - & \texttt{mask:} \quad I, \text{ binary or grayscale image,} \\ - & \texttt{marker:} \quad J, \text{ image defined on domain } D_I. \end{array} \right.$

  /⋆ The reconstruction is determined in marker-image J ⋆/

  /⋆ Note: we assume that $\forall p \in D_I,\ J(p) \leq I(p)$ ⋆/

- Repeat until stability is reached {
  Scan D_I in raster order {
  Let p be the current pixel;
  $J(p) \leftarrow (\max\{J(p), J(p+\vec{u}_1), J(p+\vec{u}_2), J(p+\vec{u}_3)\}) \wedge I(p)$;
  }
  Scan D_I in anti-raster order {
  Let p be the current pixel;
  $J(p) \leftarrow (\max\{J(p), J(p+\vec{u}_4), J(p+\vec{u}_5), J(p+\vec{u}_0)\}) \wedge I(p)$;
  }
  }

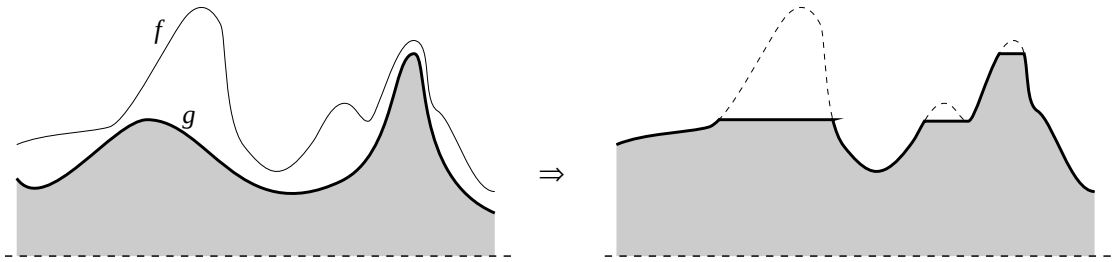


Figure 10: *Grayscale reconstruction of f from g.*

As mentioned earlier, reconstruction is a particularly powerful morphological tool. Its several binary applications (filtering, hole filling, etc) are rather well-know, but it is even more useful in the grayscale case [50, 46]. For example, to extract the maxima of an image I, it suffices to reconstruct I from $I-1$. By algebraic difference between I and the reconstructed function, one gets the desired maxima. Alternatively, the example of Fig. 11 illustrates the use of grayscale reconstruction for picture segmentation: Fig. 11.a is an image of blood vessels in the eye in which microaneurisms have to be detected. They are small compact light spots which are disconnected from the network of the (light) blood vessels. To extract them, the first step is to perform a series of openings of Fig. 11.a with respect to segments of different orientations. These segments are chosen to be longer than any possible aneurism, so that the aneurisms are removed by any such opening. On the other hand, since the blood vessels are elongated and light, there will be at least one orientation at which they are not completely removed by opening. After taking the supremum of these different openings, one gets Fig. 11.b, which is still an algebraic opening of Fig. 11.a [39]. It is used as marker to reconstruct the blood vessels entirely. Fig. 11.c is the result of the grayscale reconstruction of Fig. 11.a from Fig. 11.b. Since the aneurisms are disconnected from the blood vessels, they have not been reconstructed! Thus, by algebraic difference between Fig. 11.a and Fig. 11.c, followed by thresholding, the microaneurisms shown in Fig. 11.d are easily extracted.

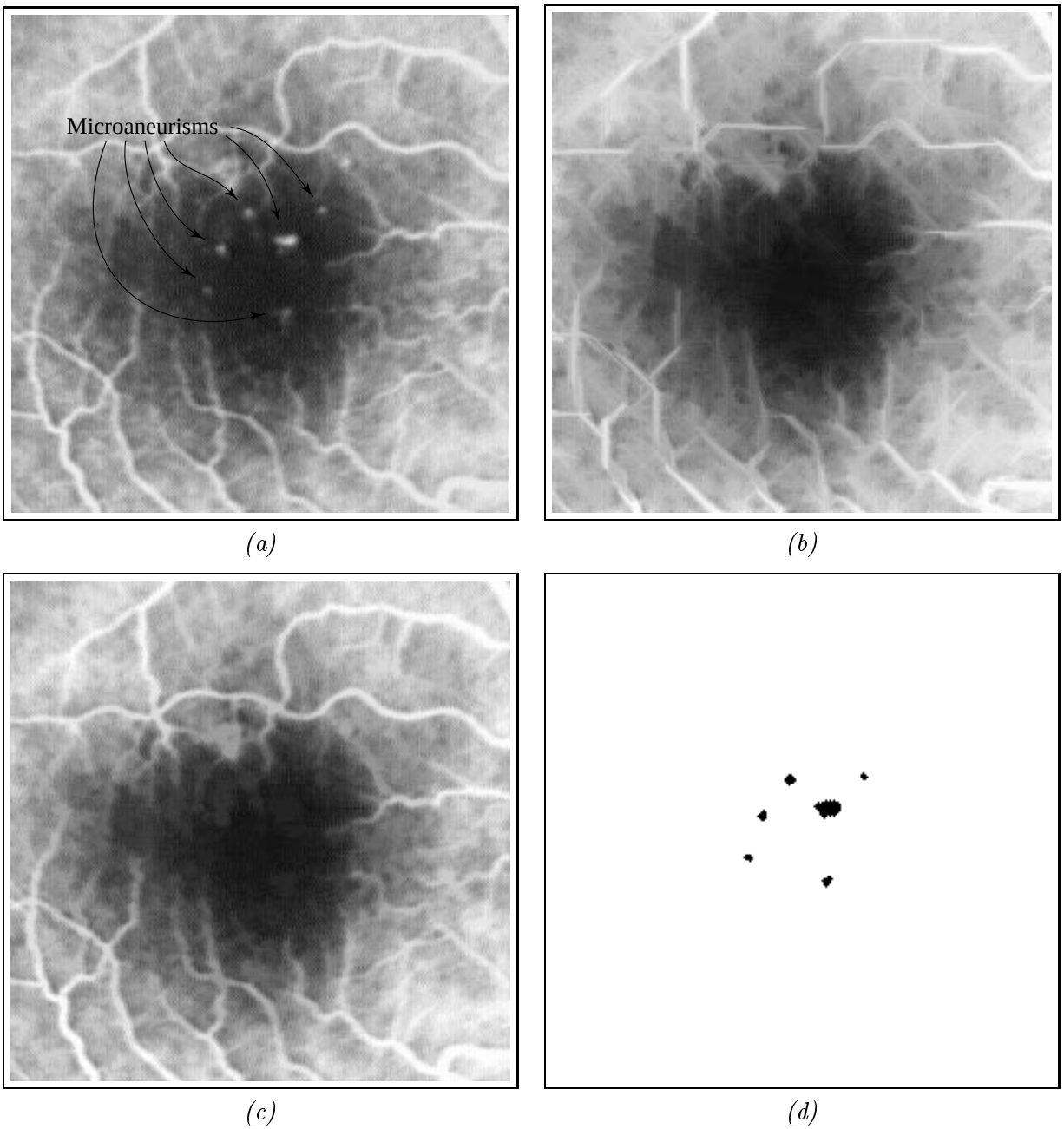


Figure 11: *Use of grayscale reconstruction for image segmentation: (a) original image blood vessels, (b) supremum of openings by segments, (c) reconstructed image, (d) microaneurisms obtained by substracting (c) from (a) and thresholding the result.*

The above reconstruction algorithm also underscores some typical drawbacks of sequential algorithms. For example, in the binary case, when the mask is a "rolled-up" particle, the number of image scannings required for its reconstruction may be very important, as illustrated by Fig. 12. However, only the values of a few pixels are actually modified after each scanning!

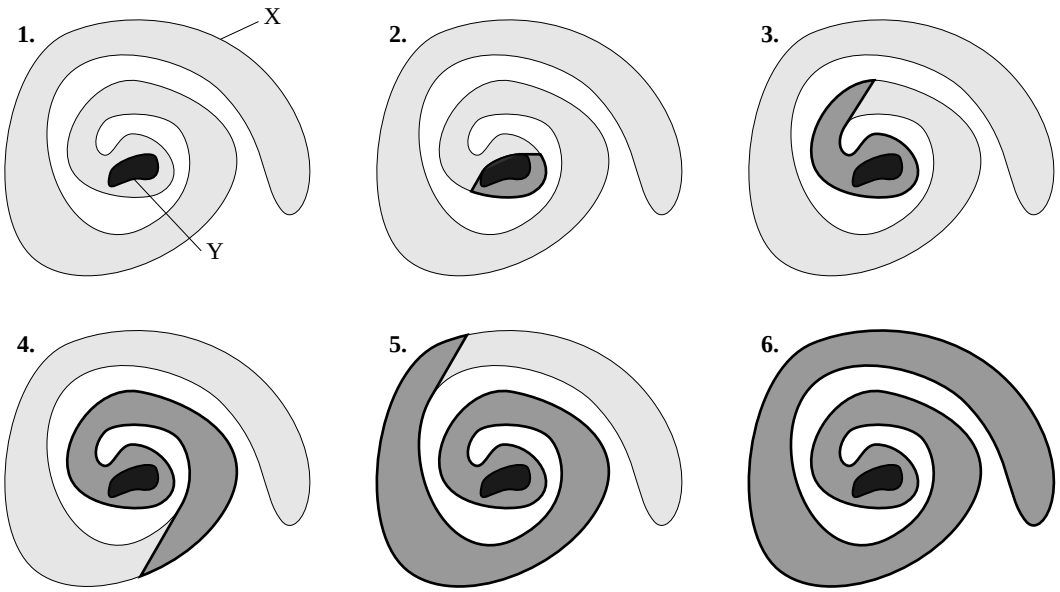


Figure 12: *The sequential computation of a binary reconstruction in a rolled up mask may involve several complete image scannings: here, the hatched zones represent the pixels which have been modified after each step.*

For this reason, a further step in the design of efficient morphological algorithms consists in *considering only the pixels whose value may be modified.* A first scanning is used to detect the pixels which are the process initiators and are typically located on the boundaries of the objects or regions of interest. Then, starting from these pixels, information is propagated only in the relevant image parts. The categories of algorithms described in the next two sections rely on these principles. They both require a *random access* to the image pixels as well as to the neighbors of a given pixel.

4 Loop and Chain Algorithms

These methods were proposed in 1988 by M. Schmitt [34] and are based on the following simple remark:

> *in a metric space (E, d), the boundary of the dilation $\delta(X)$ of a set X by an isotropic structuring element is a curve which is parallel to the boundary of X.*

This is illustrated by Fig. 13. Hence, if one is able to determine quickly the curves parallel to a given one, the calculation of isotropic dilations can be efficiently realized. This process can then be used to compute a large number of other morphological transformations which can be defined from isotropic dilations in an incremental fashion. Among others, distance functions, which are nothing but "piles" of erosions, are attainable this way.

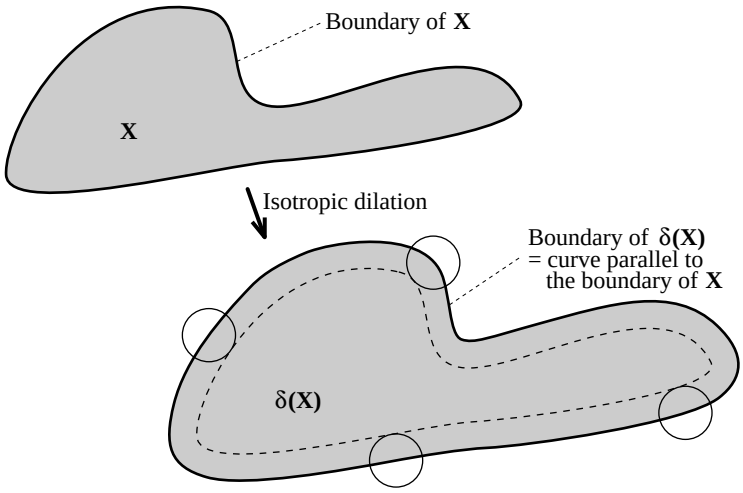


Figure 13: *The boundary of the dilated set $\delta(X)$ is parallel to the boundary of X.*

The first step of these algorithms therefore consists in a tracking of the contours of the image I under study and in their encoding as Freeman loops [12]. A loop L is a data structure made of:

1. an origin pixel Or_L,
2. a length $l(L)$,
3. a list of $l(L)$ integers of the segment $[0, 5]$, coding the elementary vectors $\vec{u}_0$, $\vec{u}_1$, ..., $\vec{u}_5$ of the hexagonal grid.

The two extremities of a loop coincide. An example of a loop and of its encoding is shown in Fig. 14.

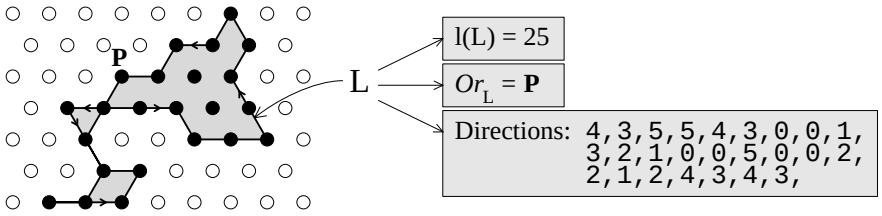


Figure 14: *Example of a loop and of its encoding.*

Given a loop L coding the boundary of a set $X \subset \mathbb{Z}^2$, the dilated loop δL—coding the boundary of $\delta(X)$—is determined by means of *rewriting rules.* These rules allow one to derive from two successive contour elements of L a certain number (between 0 and 5) of contour elements of δL. The dilated loop is thus obtained from L in linear time with respect to $l(L)$. In the hexagonal case, there are exactly six rewriting rules (up to the six rotations), which are illustrated by Fig. 15. Rule number 4 may seem useless, but is in fact essential as soon as two successive dilations have been performed [34].

Now, once the dilated loops are determined, they must be written in the image and the corresponding pixels have to be given the appropriate value. For example in the case of a binary

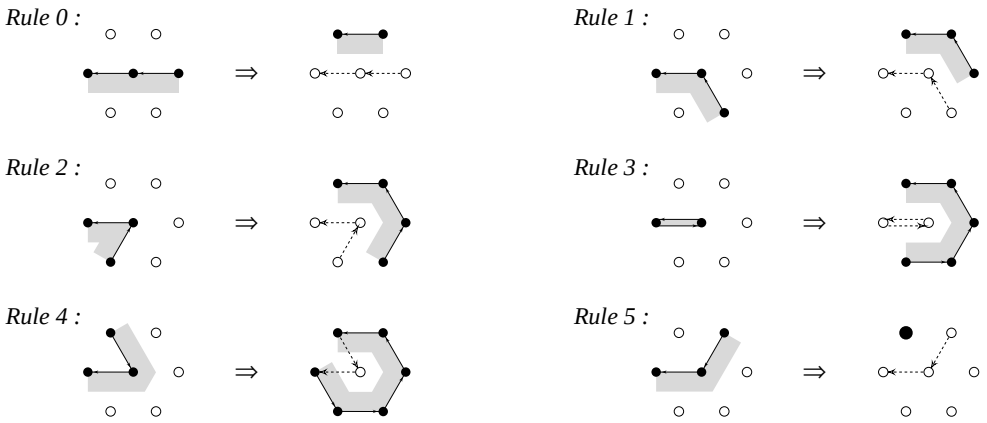


Figure 15: *Rewriting rules allowing to determine δL from L.*

dilation, the pixels corresponding to the dilated loops have to be assigned value 1. In fact, while the loops are written in the original image, one can detect if some of them intersect. For example, it may well happen that two loops coming from two different connected components intersect after a dilation step, as illustrated by Fig. 16. In such cases, the overlapping parts become useless and can be cut. The remaining loop parts are called chains: they are nothing but loops whose extremities do not coincide, and are manipulated exactly as loops by using the rewriting rules of Fig. 15. They can well be cut again at further steps. An example of a chain dilation is shown in Fig. 17. The succession of operations described in this paragraph is referred to as *adjustment.* More precisely, during this adjustment step, one only keeps the chain or loop parts which are located in a given mask (set of pixels having a certain value) and gives them the appropriate value.

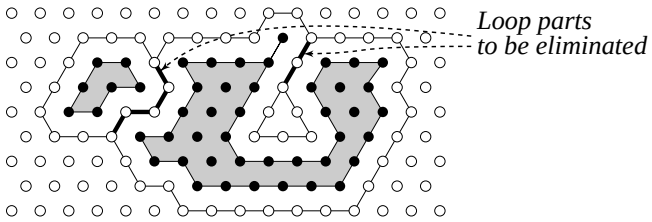


Figure 16: *After dilation, certain loop parts must be eliminated: this is the adjustment step, in which* chains *are created. Here, the original (two) loops are located around the gray areas, and the parts of the dilated loops to be cut are drawn in a bold stroke.*

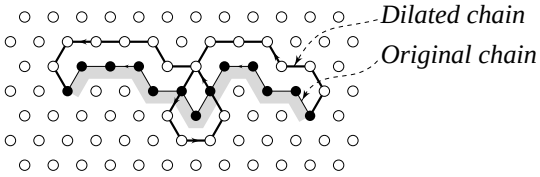


Figure 17: *Dilation of a chain. Note that a loop has been created here, which will be eliminated during the adjustment step.*

As an illustration, let us consider again the case of the hexagonal distance function. To determine it, we iterate dilations and adjustments of the chains until stability is reached. After each step, the value given to pixels in the "adjusted" chains is incremented by 1.

Algorithm: distance function by chains and loops

- input: I, binary image; /⋆ distance directly computed in I ⋆/
- Track the contours of I^C (complement image) and encode them as loops;
- $dist \leftarrow 2$; /⋆ variable containing the current distance ⋆/
- Repeat until there remain chains or loops {
  Dilate the chains and the loops;
  Adjust them in the mask $\{p \in D_I \mid I(p) = 1\}$, giving the corresponding pixels value $dist$;
  $dist \leftarrow dist + 1$;
  }

Like almost all the algorithms relying on this chain propagation principle, the above one only requires two image scannings: one for the contour tracking step plus one scanning *of the feature pixels only* in the propagation step (in fact here, to avoid an additional scanning, the algorithm is designed to yield $\mathrm{dist}_I + 1$).

Chains and loops algorithms are thus extremely fast. Moreover, after the initial contour tracking is achieved, loops and chains may well be propagated *inside a given mask:* for this reason, the present methods are particularly suited to the computation of binary geodesic transformations [34]. In this framework, they are the fastest techniques available.

Propagation function

The previous remark is particularly true for the *propagation function* of a simply connected set, where chain propagation methods provide the only known efficient algorithm [35, 24]. Recall that the propagation function p_X associates with each pixel of a connected set X its geodesic distance to the farthest pixel of X:

$$p_X \left(\begin{array}{ccc} X & \longrightarrow & \mathbb{Z}^+ \\ p & \longmapsto & \sup\{d_X(p,q) \mid q \in X\}. \end{array} \right. \tag{5}$$

The algorithm for computing p_X is detailed in [34] for the hexagonal case. It basically works through the determination of a supremum of geodesic distance functions, each of these being obtained via chain propagations. This transformation is illustrated by Fig. 18 in the case of a 4-connected square grid. It has a very large number of practical applications, ranging from the extraction of extremities and geodesic centers [34] to the determination of anti-skeletons [36].

Euclidean distance function and applications

Chains and loops algorithms are flexible in that they can also be adapted to better distances: for example, the dodecagonal one can be obtained by modifying the rewriting rules of Fig. 15 [34, pages 86–89]. But it is even possible to adapt these algorithms to actual Euclidean distance [48]: the idea is to modify the chain and loop structure as well as the rewriting rules of Fig. 15 in such a

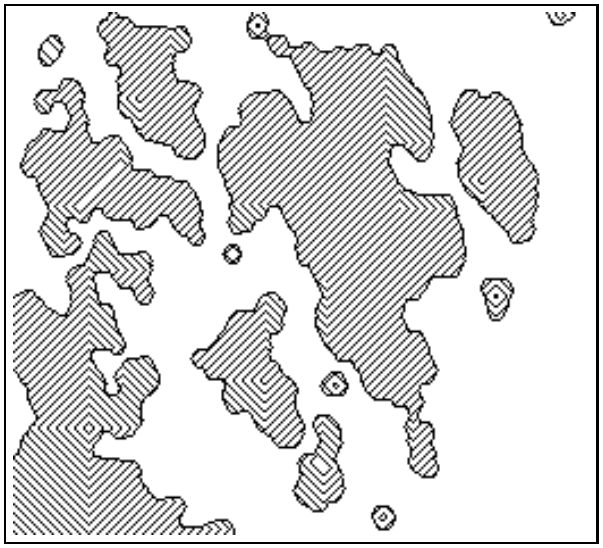


Figure 18: *Level lines of the 4-connected propagation function of a binary image.*

way that Euclidean distances are conveyed in the image by theses structures. Previous algorithms were of sequential type and only yielded more isotropic distances [8] or approximations of Euclidean distance functions [10]. An example of exact Euclidean distance function is shown in Fig. 19.

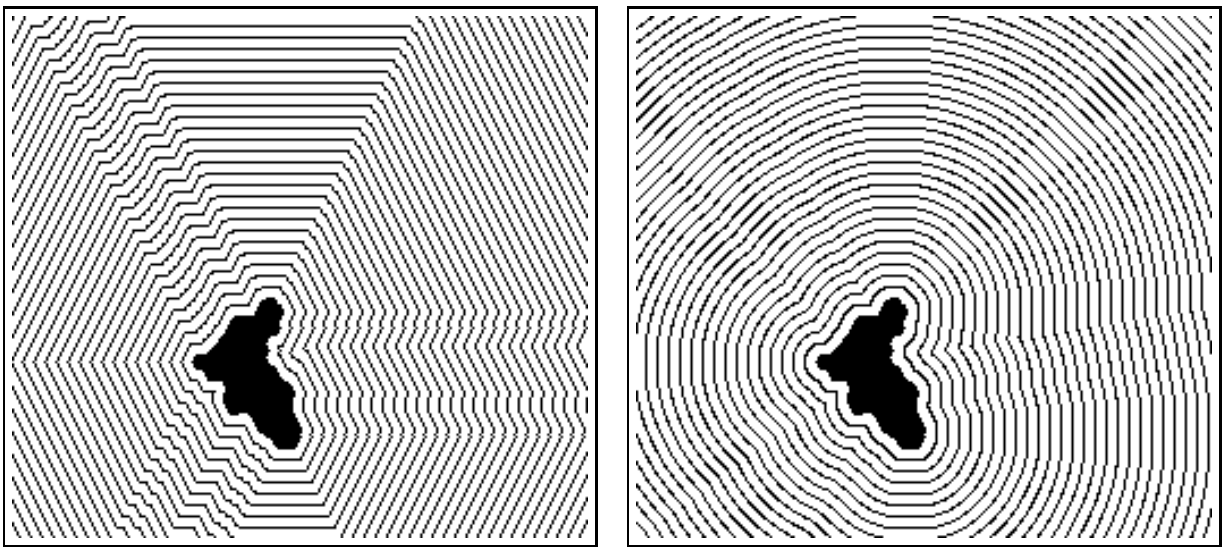


Figure 19: *Comparison between hexagonal and Euclidean distance function.*

The same technique extend to the determination of Euclidean skeletons [27] and skeletons by influence zones [21]. Additionally, Delaunay triangulations [30, 6], Gabriel graphs [15] and relative neighborhood graphs [40] can be derived from these methods, and obtained in arbitrary binary pictures [46, 48]. Examples can be found in the chapter of the present book entitled "Graph Morphology in Image Analysis" [18].

Morphological transformations with arbitrary structuring elements

Other extensions of the present loop based methods include efficient algorithms for computing binary dilations, erosions, openings and closings with structuring elements of arbitrary size and shape [49]. Here, chains and loops are no longer propagated in the image. Instead, the involved structuring element is encoded appropriately and propagated along the loops representing the set to be dilated or eroded. Combining an erosion and a dilation step allows us to determine openings and closings equally well, as illustrated by Fig. 20.

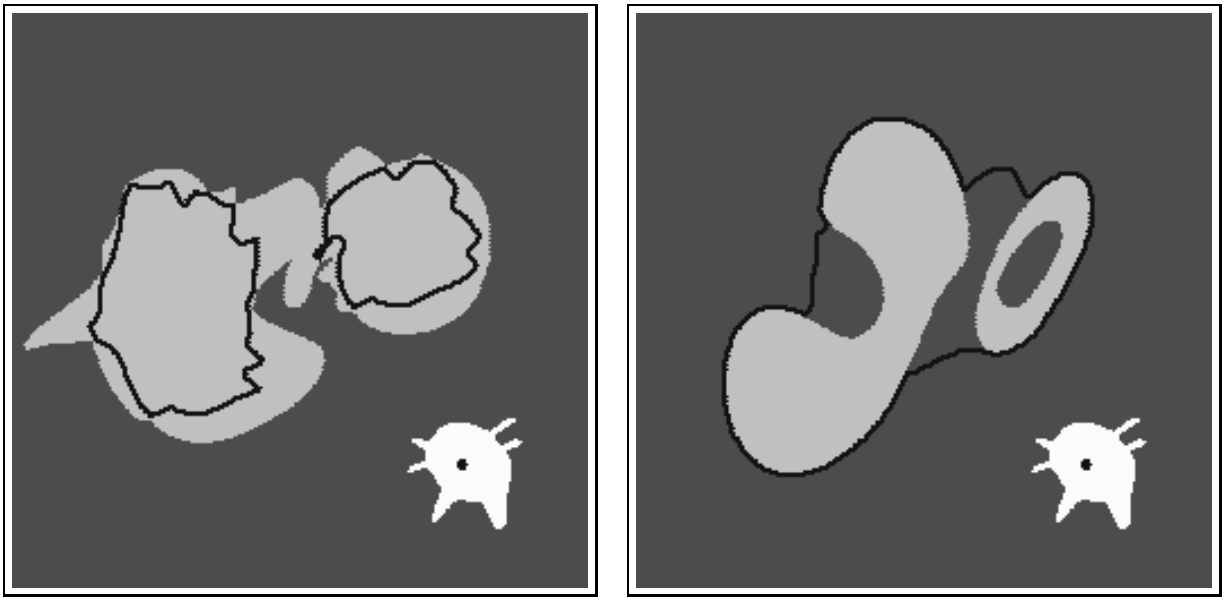


Figure 20: Binary opening and closing by an arbitrary (and weird!) structuring element.

To summarize, chains and loops are particularly efficient for the computation of binary morphological transformations and give rise to the interesting extensions described above. Unfortunately, they are not easy to adapt from one grid to another. Furthermore, they do not extend to multidimensional spaces. In these respects, the algorithms based on queues of pixels which are discussed below are much more general.

5 Algorithms Based on Queues of Pixels

In this section, we again satisfy the principle according to which only the "interesting" image pixels are considered at each step. The image under study is regarded as a graph whose vertices are the pixels, and whose edges are provided by the discrete grid G. Then, instead of loops and chains, we make use of a *queue of pixels* to perform breadth-first scannings of this graph. This idea has already proved to be particularly interesting in image analysis and morphology [41, 46].

A queue is a First-In-First-Out (FIFO) data structure, which means that the pixels which are first put into it are those which can first be extracted. In other words, each new pixel included in the queue is put on one side whereas a pixel being removed is taken from the other side (see Fig. 21). In practice a queue is simply a large enough array of pointers to pixel, on which three operations may be performed:

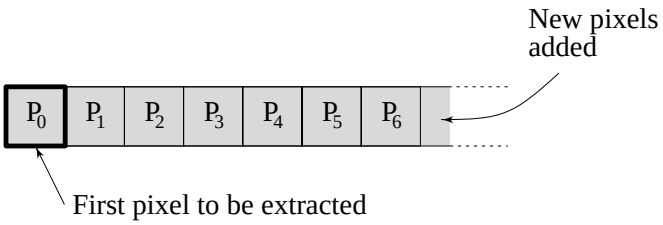


Figure 21: *How a queue of pixels works.*

- *fifo_add(p):* *puts the (pointer to) pixel p into the queue.*
- *fifo_first():* *returns the (pointer to) pixel which is at the beginning of the queue, and removes it.*
- *fifo_empty():* *returns* true *if the queue is empty and* false *otherwise.*

The implementation of our distance function using this queue and the above operations is accomplished as follows:

Algorithm: distance function using a queue of pixels

- `input:` I`, binary image;`
  `/*` `The distance function is computed in` I `directly` `*/`
- `For every pixel` $p \in D_I$`, do {`
  `/*` `detection of the pixels to be initially put on the queue` `*/`
  `If` $I(p) = 1$ `and` $\exists p' \in N_G(p), I(p') = 0$ `{`
  fifo_add(p);
  $I(p) \leftarrow 2$;
  }
  }
- `While` *fifo_empty*$() =$ *false* `{`
  $p \leftarrow$ *fifo_first*$()$;
  `For every` $p' \in N_G(p)$ `{`
  `If` $I(p') = 1$ `{`
  $I(p') \leftarrow I(p) + 1$;
  fifo_add(p');
  }
  }
  }

Here again, this algorithm actually yields $dist_I + 1$, a trick which avoids an additional image scanning.

Like the methods described in the previous section, queue based algorithms are extremely efficient, in both the non-geodesic and the geodesic cases. The simplicity of the above distance function procedure is also interesting, and this characteristic is shared by most FIFO algorithms.

They are thus more suited than the chain propagation ones to the development of procedures for computing complex transformations like grayscale reconstruction (for an efficient alternative to the sequential algorithm described in Section 3, see [46, chapter 6]), skeletons and watersheds. The latter two are briefly described and illustrated below.

Moreover, contrary to the chain propagation algorithms, the present ones are extremely easy to adapt from one grid to another, since it suffices to modify the function that generates the neighbors of a given pixel. Similarly, their extension to n-dimensional images and even to graphs is straightforward. They have been used to implement a large number of morphological transformations on graphs [43], which have already been used in physical applications [51, 19] and are expected to be of great interest for complex picture segmentation tasks [44]. Several ideas and algorithms about mathematical morphology on graphs can be found in [42] or in chapter "Graph Morphology in Image Analysis", in the present book [18].

Skeletons

The skeleton transformation is widely used in morphological image processing. It was introduced by Blum in 1961 as the *medial axis transformation* [5]. The definition he proposed is based on the concept of grassfire: assuming a grassfire starting from the boundary of a set $X \subset \mathbb{Z}^2$ is propagating within it a uniform speed, the skeleton $S(X)$ of X is the set of the pixels where different firefronts meet. This is illustrated by Fig. 22. A more formal definition of the skeleton was then proposed by Calabi [9], based on the notion of maximal ball: the skeleton of X is defined as the locus its maximal balls for the used metrics. One can show that this skeleton can be obtained as the set of local maxima of the distance function of X [46, 47].

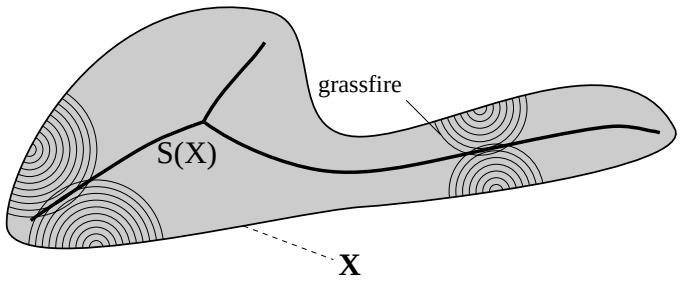


Figure 22: The skeleton can be viewed as the locus of the pixels where two firefronts stemming from the objects boundaries meet.

Unfortunately, a well-known result is that the skeleton by maximal balls, sometimes called the *true skeleton,* is not necessarily connected even though the original set is! However, to be useful in practice, skeletonization needs to be a homotopic transformation [38]: roughly speaking, it needs to preserve the number of connected components and the number of holes in the original set, as well as the inclusion relationships between these components and holes. The practical problem with the implementation of skeletons consists therefore in extracting an object of unit-width which would be as close as possible to the skeleton by maximal balls, while preserving the homotopy of the original set. The first class of methods proposed in the literature are part of what we referred to in this chapter as parallel algorithms (see Section 2). They consist in removing successive peels of the set by means of homotopic thinnings until stability is reached [38]. For example, with hexagonal connectivity one generally performs homotopic thinnings with respect to structuring elements L

shown in Fig. 23 [17]. The main drawbacks of these techniques are their inefficiency and the fact that the resulting objects, though homotopic and of unit width, have not much in common with skeletons by maximal balls!

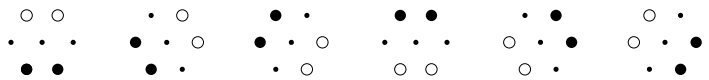


Figure 23: *The two-phase structuring element L and its 6 rotations.*

Sequential algorithms based on the crest-lines of the distance function have also been proposed [26, 28]. They are more efficient than the previous ones (they work in a fixed number of images scannings), produce accurate results and can even be extended to Euclidan distances [27]. However, as explained in [47], these algorithms require cumbersome neighborhood analyses and their flexibility is rather poor. Other methods proposed in the literature include computational geometry based algorithms [30] as well as contour-based techniques [1, 53]. They are among the most efficient methods, but are very complex, have little flexibility and only allow the determination of one given type of skeleton (see [46]).

Based on the above remarks, the skeleton algorithm detailed in [47] makes use of homotopic peelings, crest points and contours: more precisely, starting from the boundaries of X, successive peelings are realized until stability (i.e., one-pixel thickness) is reached. These peelings—or grassfire propagation process—are efficiently implemented via a queue of pixels. At every step, the current pixel p may be removed (i.e., given value 0) if and only if one of the following conditions is fulfilled:

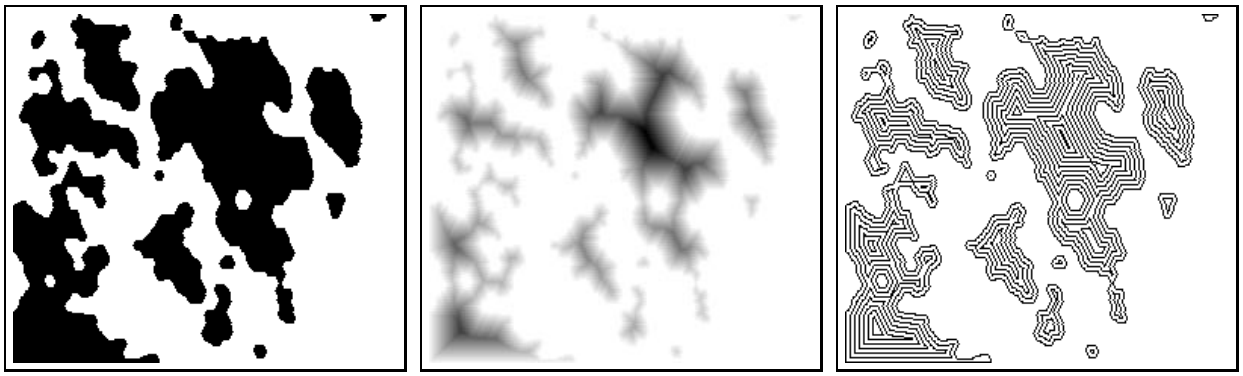
1. p does not belong to the skeleton by maximal balls. In other words, p is not a crest-point (local maximum) of the distance function.

2. Removing p does not modify the homotopy locally.

The first condition assures the accuracy of the result in that it will be a superset of the skeleton by maximal balls. The second one means that the resulting object is homotopic; the local homotopy checkings are realized via specially designed look-up tables [47]. In this skeletonization process, the pixels belonging to the skeleton by maximal balls play the role of *anchor points:* the firefronts stemming from the boundaries of X tend to anchor themselves on these particular points. An example of skeletonization based on these principles is shown in Fig. 24.

Like every FIFO algorithm, the present one is particularly efficient since only the feature pixels are considered during the fire propagation step. For example, the skeletonization process illustrated by Fig. 24 takes less than one second on a *Sun Sparc Station 1.* The resulting skeletons are also more accurate than with most other methods: they are skeletons by maximal balls to which connecting arcs of unit thickness have been added for homotopy preservation.

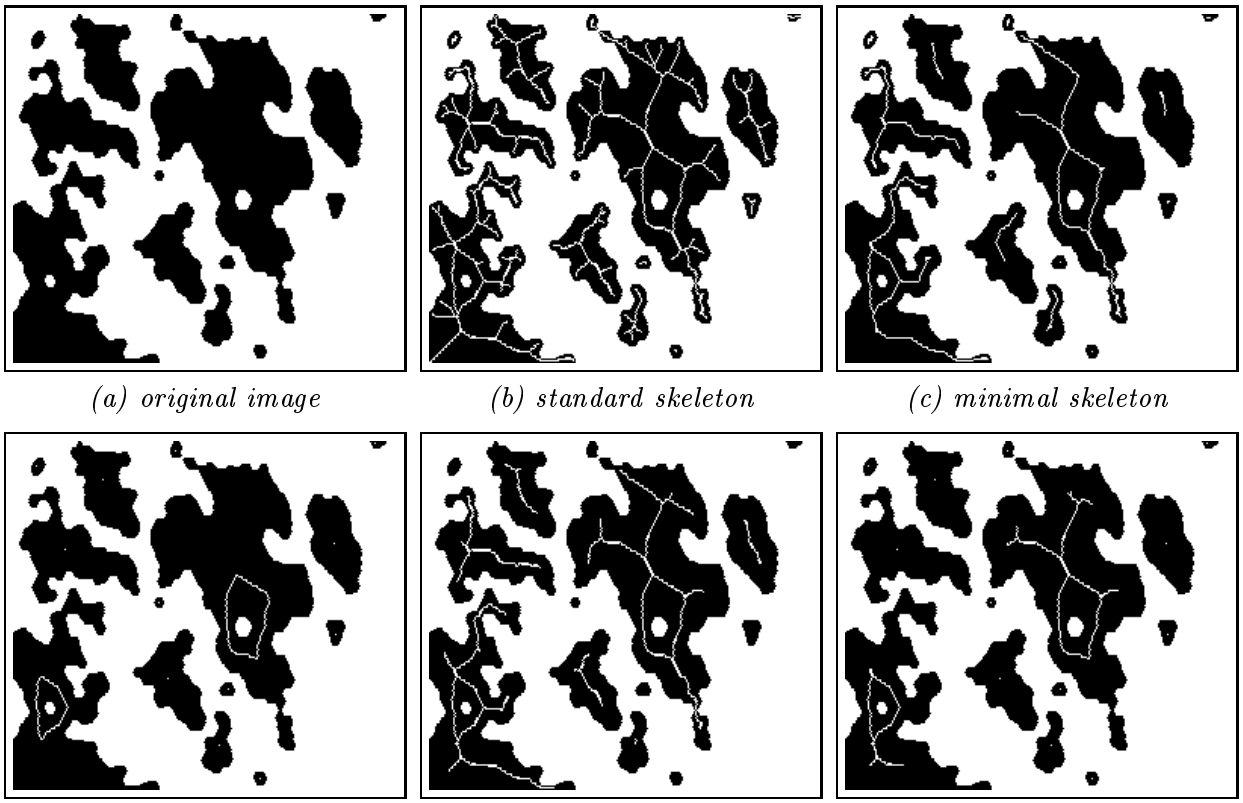
The algorithm works in the Euclidean case as well as in the geodesic one. Most interestingly, it allows us to calculate a whole range of different skeleton-like transformations whose computation is hardly possible otherwise: this is simply achieved by using different sets of anchor points. For example, by taking as anchor points the *regional* maxima of the distance function instead of its local maxima, one gets an object referred to in the literature as the *minimal skeleton.* Similarly, an empty set of anchor points results in a "homotopic marking" of the set. Using as anchor points only the local maxima of elevation greater than n yields a smoother skeleton called *skeleton of order n* [47]. Any of these skeletons can then be post-processed via prunings, themselves realized via FIFO algorithms. Fig. 25 shows a sample of these possibilities. To summarize, the present queue-based skeleton algorithm is particularly efficient, accurate and flexible.



(a) original image *(b) distance function* *(c) level lines of this function*

(d) skeleton by maximal balls *(e) propagation of the firefront* *(f) final skeleton*

Figure 24: Construction of the standard skeleton.



(a) original image *(b) standard skeleton* *(c) minimal skeleton*

(d) homotopic marking *(e) pruned skeleton (16 iterations)* *(f) skeleton of order 16*

Figure 25: Various kinds of skeletons which may be efficiently determined using algorithms based on queues of pixels [47]. This example was produced using the hexagonal grid.

Watersheds

In the last decade, increasing attention has been put on the watershed trasformation as a tool for image segmentation [11, 3, 2, 52]. It is defined for grayscale images via the notion of a *catchment basin:* let us regard the image under study as a topographic relief (where the gray-level of a pixel stands for its altitude) on which it is raining on. A drop of water falling at a point p flows down along a steepest slope path until it is trapped in a minimum m of the relief. The set $C(m)$ of the pixels such that a drop falling on them eventually reaches m is called catchment basin associated with the minimum m. The set of the boundaries of the different catchment basins of an image constitute its *watersheds.* These notions are demonstrated in Fig. 26.

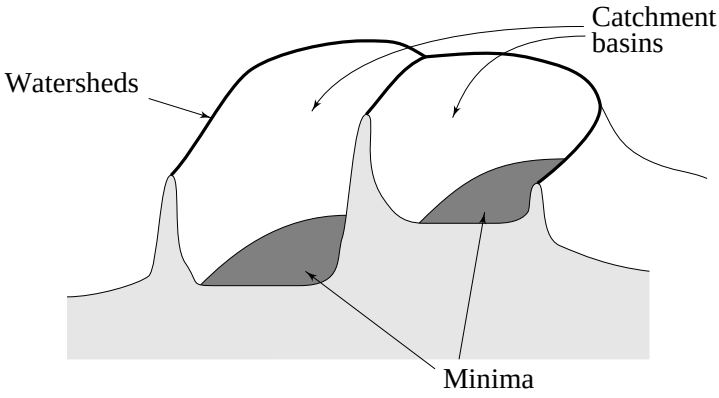


Figure 26: *Minima, catchment basins and watersheds.*

Here again, numerous techniques have been proposed to determine watersheds in digital pictures. The major ones are reviewed in [46, 52]. One of the most interesting algorithms, originally proposed by Beucher, consists of "inverting" the watershed definition. Consider that the minima of the image—regarded here as a 3-D surface—have been pierced and that this image is slowly immersed into a lake. The water progressively floods the different catchment basins, and at some point, water originating from two different minima will merge, thereby connecting the corresponding catchment basins. We prevent this by erecting dams at every place where connection would otherwise occur. Once the surface is totally immersed, the set of dams thus built corresponds to the watersheds of the initial image.

This immersion and dam erection process can now be simulated by an algorithm. The most efficient algorithm described in the literature makes use of FIFO breadth-first scanning techniques for the actual flooding of the catchment basins [52]. A labelling of the catchment basins is also used, which automatically prevents the connection of two different basins. It has been shown that the results provided by this technique are more accurate that those of any other method. For example, the cone effect of Fig. 5.b is always avoided. Furthermore, just like almost all other FIFO algorithm, the present one extends to any grid and any dimension in a straightforward manner.

This algorithm dramatically reduces the computation times required for extracting watersheds. On conventional computers, previous approaches typically needed up to a couple of hours! The present one takes around 5 to 10 seconds on a Macintosh II, for a 256×256 image, thus opening the door to powerful segmentation methods in inexpensive software-based systems.

Let us conclude this section by an example of application of the present algorithm. We consider here Fig. 27.a: it is part of a series of successive images of the same scene, and the problem is to recover the motion of the camera. One of the approaches taken to solve this motion estimation problem consists of decomposing the images into regions and to match these regions over successive

time frames [13]. Performing this decomposition by means of watershed techniques turns out to provide meaningful regions, which are then easily matched from one image to the next one [14]. Following the methodology described in [50], the watershed tool is applied to the morphological gradient of the original image (Fig. 27.c). In fact, to avoid oversegmentation, the watersheds of the gradient are constrained by a *marker image.* The markers are connected components of pixels belonging to each of the regions to be extracted, and are obtained in this case from the intensity "domes" and "basins" of the original image (see Fig. 27.b). The result of this constrained watershed transformation is the highest watershed lines of the gradient which are located between two markers (Fig. 27.d). The entire segmentation process takes approximately 5 seconds on a *Sun Sparc Station 2.*

6 Conclusion and Summary

Table 2 summarizes the qualities and drawbacks of the families of algorithms which have been briefly reviewed in this chapter. Now, in practice, what algorithm should be chosen to implement a given transformation in a given environment? Of course, there is no absolute answer to this question. However, the following guidelines can be proposed:

- Parallel algorithms should generally be avoided, unless running on specialized architectures.
- Sequential algorithms constitute one of the best choices to implement grayscale reconstructions, grayscale dilations and erosions with some structuring elements, and granulometry functions.
- Chain and loop methods should be chosen whenever binary transformations are concerned. They are indeed the fastest in this case, particularly for geodesic operations (reconstruction, geodesic distance function, hole filling, labelling, etc), and provide the only known efficient propagation function algorithm.
- FIFO algorithms will be preferred in all other cases, and in particular in the following ones: geodesic transformations in square grids, n-dimensional or graph morphology, complex transformation like skeletons, SKIZ and watersheds.

Family of algorithms	speed	accuracy (if appropriate)	adaptation to other grids	development ease	hardware implementation
parallel	x	xx	xx	xxx	xxxx
sequential	xx	xx	xx	xx	xxx
loops and chains	xxxx	xxx	x	xxx	x
queues	xxx	xxx	xxxx	xxxx	x

Table 2: Respective qualities of the various families of algorithms.

Clearly, the last two families are going to become increasingly important in the future. Regardless of accuracy and flexibility considerations, chain or queue algorithms are often faster on conventional computers than parallel algorithms on specialized hardwares! Now, a few years after the introduction of the parallel morphological algorithms, the first specialized hardwares were built on these principles. Between the publication of the sequential distance function algorithm (1968) and its first hardware implementation (1989), more than twenty years have elapsed. Let us hope

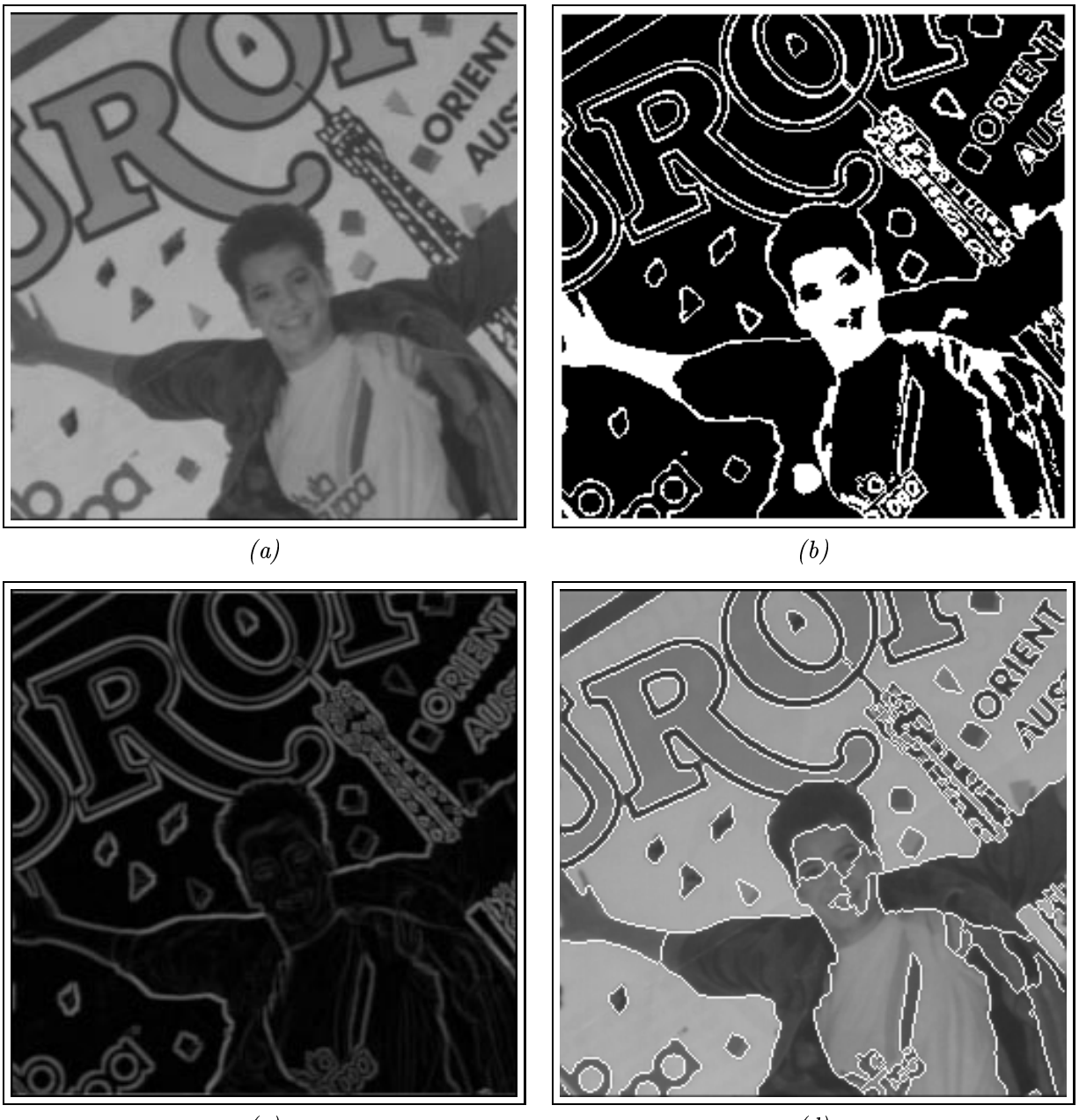


Figure 27: Example of a watershed segmentation: (a) original image, (b) marker image, (c) gradient image, (d) final segmentation obtained via watersheds of the gradient controlled by the markers.

that we will not have to wait twenty more years to see the first hardware realizing queue based morphological operations. Indeed, this would probably allow us to compute complex morphological transformations in just several hundredths of a second.

Acknowledgements

This work was supported in part by *Dassault Electronique* and the *National Science Foundation* under Grant MIPS–86–58150, with matching funds from *DEC* and *Xerox.* Most of the examples have been produced using the software DIP Station^{TM} on a Macintosh. The author would like to thank Dr. Peter Winthrop Hallinan for his help in reviewing this chapter.

References

[1] C. Arcelli, L. Cordella, and S. Levialdi. From local maxima to connected skeletons. *IEEE Trans. Pattern Anal. Machine Intell.*, 2:134–143, 1981.

[2] S. Beucher. *Segmentation d'Images et Morphologie Mathématique.* PhD thesis, Ecole des Mines, Paris, June 1990.

[3] S. Beucher and C. Lantuéjoul. Use of watersheds in contour detection. In *International Workshop on Image Processing, Real-Time Edge and Motion Detection/Estimation*, Rennes, France, 1979.

[4] I. Bloch. Morphologie mathématique et représentation d'images par arbres. Technical report, Ecole des Mines, CMM, Paris, 1987.

[5] H. Blum. An associative machine for dealing with the visual field and some of its biological implications. In E. Bernard and M. Kare, editors, *Biological Prototypes and Synthetic Systems*, pages 244–260. Plenum Press, New York, 1962. Proc. second Annual Bionics Symposium, Cornell University, 1961.

[6] J.-D. Boissonat. Shape reconstruction from planar cross-sections. Technical report, INRIA, Le Chesnay, France, 1986.

[7] G. Borgefors. Distance transformations in arbitrary dimensions. *Comp. Vis., Graphics and Image Processing*, 27:321–345, 1984.

[8] G. Borgefors. Distance transformations in digital images. *Comp. Vis., Graphics and Image Processing*, 34:334–371, 1986.

[9] L. Calabi and W. Harnett. Shape recognition, prairie fires, convex deficiencies and skeletons. Technical Report 1, Parke Math. Lab. Inc., One River Road, Carlisle MA, 1966.

[10] P. Danielsson. Euclidean distance mapping. *Comp. Graphics and Image Processing*, 14:227–248, 1980.

[11] H. Digabel and C. Lantuéjoul. Iterative algorithms. In J.-L. Chermant, editor, *2nd European Symposium on Quantitative Analysis of Microstructures in Material Science, Biology and Medicine*, pages 85–99, Stuttgart FRG, 1978. Riederer Verlag.

[12] H. Freeman. On the encoding of arbitrary geometric configurations. *IEEE Transactions on Computers*, C10:260–268, 1961.

[13] C.-S. Fuh and P. Maragos. Region-based optical flow estimation. In *IEEE Int. Computer Vision and Pattern Recog. Conference*, pages 130–135, San Diego CA, 1989.

[14] C.-S. Fuh, P. Maragos, and L. Vincent. Region-based approaches to visual motion correspondence. Technical report, HRL, Harvard University, Cambridge, 1991. Submitted to PAMI.

[15] K. Gabriel and R. Sokal. A new statistical approach to geographic variations analysis. *Systematic Zoology*, 18:259–278, 1969.

[16] S. Gesbert, V. Howard, D. Jeulin, and F. Meyer. The use of basic morphological operations for 3-d biological image analysis. *Transactions of the Royal Microscopical Society*, 1, 1991.

[17] M. Golay. Hexagonal pattern transforms. *IEEE Trans. on Computers*, 18(8), 1969.

[18] H. Heijmans and L. Vincent. Graph morphology in image analysis. In E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, pages 171–203. Marcel-Dekker, Sept. 1992.

[19] D. Jeulin, L. Vincent, and G. Serpe. Propagation algorithms on graphs for physical applications. *JVCIR*, 3(2):161–181, June 1992.

[20] J.-C. Klein and R. Peyrard. PIMM1, an image processing ASIC based on mathematical morphology. In *IEEE's ASIC Seminar and Exhibit*, pages 25–28, Rochester NY, 1989.

[21] C. Lantuéjoul. Issues of digital image processing. In R. M. Haralick and J.-C. Simon, editors, *Skeletonization in Quantitative Metallography*. Sijthoff and Noordhoff, Groningen, The Netherlands, 1980.

[22] C. Lantuéjoul and F. Maisonneuve. Geodesic methods in quantitative image analysis. *Pattern Recognition*, 17(2):177–187, 1984.

[23] B. Laÿ. Recursive algorithms in mathematical morphology. In *Acta Stereologica Vol. 6/III*, pages 691–696, Caen, France, Sept. 1987. 7th International Congress For Stereology.

[24] F. Maisonneuve and M. Schmitt. An efficient algorithm to compute the hexagonal and dodecagonal propagation function. In *5th European Congress For Stereology*, pages 515–520, Freiburg im Breisgau FRG, Sept. 1989. Acta Stereologica. Vol. 8/2.

[25] G. Matheron. *Random Sets and Integral Geometry*. John Wiley and Sons, New York, 1975.

[26] F. Meyer. Skeletons and perceptual graphs. *Signal Processing*, 16(4):335–363, 1989.

[27] F. Meyer. Digital Euclidean skeletons. In *SPIE Vol. 1360, Visual Communications and Image Processing*, pages 251–262, Lausanne, Switzerland, Oct. 1990.

[28] A. Montanvert. *Contribution au Traitement de Formes Discrètes. Squelettes et Codage par Graphe de la Ligne Médiane*. PhD thesis, Université Scientifique, Technologique et Médicale de Grenoble, France, 1987.

[29] L. Piper and J.-Y. Tang. Erosion and dilation of binary images by arbitrary structuring elements using interval coding. *Pattern Recognition Letters*, pages 201–209, Apr. 1989.

[30] F. Preparata and M. Shamos. *Computational Geometry: an Introduction*. Springer Verlag, 1985.

[31] A. Rosenfeld and J. Pfaltz. Sequential operations in digital picture processing. *J. Assoc. Comp. Mach.*, 13(4):471–494, 1966.

[32] A. Rosenfeld and J. Pfaltz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[33] H. Samet. The quadtree and other related hierarchical data structures. *ACM Computing Surveys*, 16(2):87–260, 1984.

[34] M. Schmitt. *Des Algorithmes Morphologiques à l'Intelligence Artificielle*. PhD thesis, Ecole des Mines, Paris, Feb. 1989.

[35] M. Schmitt. Geodesic arcs in non-euclidean metrics: Application to the propagation function. *Revue d'Intelligence Artificielle*, 3 (2):43–76, 1989.

[36] M. Schmitt. The anti-skeleton: some theoretical properties and application. In *SPIE Vol. 1360, Visual Communications and Image Processing*, Lausanne, Switzerland, Oct. 1990.

[37] M. Schmitt and L. Vincent. *Morphological Image Analysis: a Practical and Algorithmic Handbook*. Cambridge University Press, (to appear in 1997).

[38] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[39] J. Serra, editor. *Image Analysis and Mathematical Morphology, Volume 2: Theoretical Advances*. Academic Press, London, 1988.

[40] G. T. Toussaint. The Relative Neighborhood Graph of a finite planar set. *Pattern Recognition*, 12:1324–1347, 1980.

[41] L. van Vliet and B. J. Verwer. A contour processing method for fast binary neighbourhood operations. *Pattern Recognition Letters*, 7:27–36, Jan. 1988.

[42] L. Vincent. Mathematical morphology on graphs. In *SPIE Vol. 1001, Visual Communications and Image Processing*, pages 95–105, Cambridge, MA, Nov. 1988.

[43] L. Vincent. Graphs and mathematical morphology. *Signal Processing*, 16:365–388, Apr. 1989.

[44] L. Vincent. Mathematical morphology for graphs applied to image description and segmentation. In *Electronic Imaging West 89*, pages Vol. 1, pp. 313–318, Pasadena, CA, 1989.

[45] L. Vincent. Morphological shading and shadowing algorithm. In Hermès, editor, *PIXIM 89, Computer Graphics in Paris*, pages 109–124, Paris, 1989.

[46] L. Vincent. *Algorithmes Morphologiques à Base de Files d'Attente et de Lacets: Extension aux Graphes*. PhD thesis, Ecole des Mines, Paris, May 1990.

[47] L. Vincent. Efficient computation of various types of skeletons. In *SPIE Vol. 1445, Medical Imaging V*, pages 297–311, San Jose, CA, 1991.

[48] L. Vincent. Exact euclidean distance function by chain propagations. In *IEEE Int. Computer Vision and Pattern Recog. Conference*, pages 520–525, Maui, HI, June 1991.

[49] L. Vincent. Morphological transformations of binary images with arbitrary structuring elements. *Signal Processing*, 22(1):3–23, Jan. 1991.

[50] L. Vincent and S. Beucher. The morphological approach to segmentation: an introduction. Technical report, Ecole des Mines, CMM, Paris, 1989.

[51] L. Vincent and D. Jeulin. Minimal paths and crack propagation simulations. In *5th European Congress For Stereology*, pages 487–494, Freiburg im Breisgau FRG, Sept. 1989. Acta Stereologica. Vol. 8/2.

[52] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Machine Intell.*, 13(6):583–598, June 1991.

[53] Y. Xia. Skeletonization via the realization of the fire front's propagation and extinction in digital binary shapes. *IEEE Trans. Pattern Anal. Machine Intell.*, 11(10):1076–1086, 1989.

[54] H. Yamada. Complete euclidean distance transformation by parallel operations. In *7th International Conference on Pattern Recognition*, pages 69–71, Montreal, 1984.