Chapter 12

Algorithms for Mathematical Morphology

12.1. Introduction

In this chapter, we deal with the very important problem of implementing the various image analysis operators, filters and methods seen in previous chapters.

In general, researchers like to present a novel operator through a mathematical description. However, this may not always be a simple task to translate this description into computer code. Yet, if such an operator is of any interest at all, we should expect to be able to use it in practice. To get there, we need to go beyond pure mathematics into the realm of programming. We have to express the mathematical operator in an applicable algorithm.

Note that although a mathematical description is useful for understanding, an implementation often deviates from the mathematical description. This usually explains why we often see more than one implementation, with varying characteristics, for most proposed operators. Also, the evolution of computer architectures and programming techniques imply that new implementations have sometimes been proposed decades after the initial definition of some operators.

Rather than attempting to build a comprehensive database of all algorithms and data structures that have ever been used for implementing morphological operators, an undertaking that would be enormous and not very interesting to read, we concentrate on the purer algorithmic aspects of mathematical morphology in this chapter. We

Chapter written by Thierry GÉRAUD, Hugues TALBOT and Marc VAN DROOGENBROECK.

therefore ignore some specific implementation aspects dealing with both software and hardware.

Since the Babylonians and more recently since Lovelace [STU 87], an algorithm has been formally defined as a series operations sequenced to solve a problem by a calculation. morphology, filters or operators usually operate on sets or functions and are defined in formal mathematical terms.

An algorithm is therefore the expression of an efficient solution leading to the same result as the mathematical operator applied to input data. This translation process in a mathematical algorithm aims to facilitate the implementation of an operator on a computer as a program regardless of the chosen programming language. Consequently, the algorithmic description should be expressed in general and abstract terms in order to allow implementations in any environment (platform, language, toolbox, library, etc.).

Computer scientists are familiar with the formalization of the concept of an algorithm and computation on a real computer with the Turing machine [TUR 36]. This formalization enables any correct algorithm to be implemented, although not in a tractable form. Rather than describing algorithms with this formalism, we use a more intuitive notation that, in particular, relies on non-trivial *data structures*.

This chapter is organized as follows. In section 12.2, we first discuss the translation process of data structures and mathematical morphology definitions in computational terms. In section 12.3, we deal with different aspects related to algorithms in the scope of mathematical morphology. In particular, we propose a taxonomy, discuss possible tradeoffs, and present algorithmic classes. These aspects are put into perspective for the particular case of the morphological reconstruction operator in section 12.4. Finally, historical perspectives and bibliographic notes are presented in section 12.5.

12.2. Translation of definitions and algorithms

12.2.1. *Data structures*

Before discussing an algorithm, we have to describe the data to be processed and how they materialize once they are no longer pure mathematical objects.

An image f is a function from a space $\mathcal{E}$ to a space $\mathcal{V}$. Since infinite spaces cannot be stored or handled appropriately, these two spaces are always sampled (or discretized) to provide $E \subset \mathcal{E}$ and $V \subset \mathcal{V}$:

$$f : \left\{ \begin{array}{ccc} E & \longrightarrow & V \\ p & \longmapsto & f(p). \end{array} \right.$$

For convenience, a discrete topology with the notion of neighborhood is often associated with E. Let us denote an element of E by p (which stands for 'point') and a neighboring point of p by n. In the terms of graph theory, a subset of E is a mesh with nodes (the points p) and a collection of edges that connect points to their neighbors. The most common situation is given by a regular sampling of a subpart of E. The resulting mesh is then regular and points belong to a grid. Such a classical topology for a 2D image defined on a squared grid is depicted in Figure 12.1.

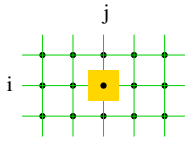

Figure 12.1. *Illustration of locations defined by a discrete squared grid and a pixel*

A point p of such an image is easily described by two integer indices (i, j). An image f, which links a point to a value, can be represented in memory by a 2D array of values: $f(p)$ is then equivalent to $f[i, j]$. This common representation of an image has the advantage that point values can be stored and modified independently. In the following, the allocation procedure for a given image f of a value $v \in V$ at point $p \in E$ is denoted in the abstract terms $f(p) := v$ although, from a practical point of view, the underlying mechanism is described by $f[i, j] := v$.

From the implementation perspective, f is not a mathematical function but a variable (or a memory) that represents a function at a given time t during the execution process of an algorithm. Formally, an algorithm generates a series of f_t functions, and each allocation changes a function f_t to the next function f_{t+1}. In computational terms, the f variable hides the existence of all the successive functions; it is similar to a function that evolves over time. The algorithm begins to deviate from mathematics.

12.2.2. *Shape and size of the function domain*

In the field of computer science, the representation of an image by an array assumes that the underlying function f is defined on a finite domain. In practice, the function domain is generally a rectangular subset of $\mathbb{Z}^2$ or of $\mathbb{N}^2$ for 2D images. A process that iterates on all points of such an image then accesses all the array elements with two loops is shown on the left-hand side of the following pseudocode.

```
for i := 1 to n_raws
    for j := 1 to n_columns
        ... // use f[i,j]
```

```
for_all p ∈ E
    ... // use f(p)
```

However, the abstract right-hand side code is to be preferred as it relates less to a specific implementation and more to the mathematics of the algorithm. In addition, it does not exhibit any restrictive underlying assumption about f: the function domain can have a non-rectangular shape and its domain does not necessarily have to be 2D. An abstract expression of an algorithm is suitable to hide the details of the representation of the data and to focus on the functionalities of the algorithm. On the other hand, we also have to provide all the details about the underlying data structure because performance and complexity issues are closely related to the basic operations on the data. If browsing all the points of a set of N points has a $O(N)$ complexity, the complexity of a random access to the value of a point p has a complexity dependent on the data structure in use.

A program accesses values of an image whose data are organized in memory as an array in constant time: the reading and writing of image values at a point p has a complexity of $O(1)$, regardless of the access order. The representation of an image as an array is practical and widely used. In addition, this representation is compact as it has a negligible overhead to describe the structure itself compared to the data associated with the image.

It might not always be the optimal solution, however. Consider for example the case of an image that describes the contour of an object. Coding contours of an object by a 2D array guarantees a constant-time access to any point of a discrete grid to determine if it belongs to the contour. However, this coding is inappropriate with regards to two other aspects. First of all, the required memory size expands from the size of the contour to the size to the smallest rectangle that encloses the object. Next, the access to any contour point requires searching in an array: we have to search for the first contour point by scanning the image line by line and then in the neighborhood to find the next contour point. For algorithms that operate on the contour of an object directly, for example a morphological dilation, it might be advisable to use a more appropriate data structure such as a list of contour points.

The conclusion is that any data structure used by an algorithm impacts on the complexity of the algorithm.

12.2.3. *Structure of a set of points*

The classical representation of a function as an array is flexible enough to be able to define a set of points. Indeed, if the destination space of f is the set of Booleans ($V = \mathbb{B}$), f can be interpreted as the characteristic function of a set of points: formally, $F = \{\, p \in E \,|\, f(p) = true \,\}$. f is then a binary image. The set $F \subset E$ encodes an object while its complementary $E \setminus F$ denotes the image background. To scan all the points of F, it is then sufficient to look for points p in the domain E satisfying $f(p) = true$. In the following, we make no notational difference between a set ($F \subseteq E$) and its characteristic function ($F : E \rightarrow \mathbb{B}$).

12.2.4. *Notation abbreviations*

For convenience, we use the notation abbreviations listed in Table 12.1 to describe algorithms.

	Extensive notation	Abbreviation
Scans all the points of a set $F \subseteq E$	**for_all** $p \in E$, **if** $F(p) = true, \ldots$	**for_all** $p \in F$, $\ldots$
Allocates a constant c to all the points of an image f	**for_all** $p \in E$, $f(p) := c$	$f := c$ □
Copies the content of an image f_1 to an image f_2	**for_all** $p \in E$, $f_2(p) := f_1(p)$	$f_2 := f_1$

Table 12.1. *List of notations and abbreviations used in this chapter*

As explained before, with this kind of abstract formulation it is possible to bypass some practical difficulties. For example, there is no need for a 2D image to be rectangular (its domain may be arbitrarily shaped). One problem occurs when dealing with the neighbors of points located on the border of the image. Again, we define the following abstract formulation to operate on all the neighboring points of a point p in turn, regardless of the shape of the neighborhood:

for_all $n \in \mathcal{N}(p)$
$\ldots$

With this notation, we can focus on the description of the algorithm and ignore some less-important implementation details.

Finally, we use the symbol $\triangleright$ to denote the conventional video order to scan all the points of an image (from the upper left corner to the bottom right corner, line by line). A reverse video order (from the bottom right corner to the upper left corner, line by line) is denoted by $\triangleleft$.

12.2.5. *From a definition to an implementation*

As written previously, if morphological operators are often described in mathematical terms, their translation in algorithmic terms is not always straightforward. Experience also shows that, in the event that such a translation is possible, its efficiency is often questionable (sometimes, it is even the worst possible implementation). This should not come as a surprise as the purpose of a mathematical definition is to ensure the correctness of an algorithm rather than to provide hints for an appropriate algorithm.

To discuss the suitability of algorithms, we consider a binary dilation. There are several equivalent ways to define the dilation of a set X by a set B:

$$X \oplus B = \bigcup_{b \in B} X_b \tag{12.1}$$

$$= \{ x + b \in E \mid x \in X, b \in B \} \tag{12.2}$$

$$= \{ p \in E \mid \exists b \in B, p - b \in X \}. \tag{12.3}$$

Definition (12.1) leads to a so-called trivial algorithm given as algorithm (1) in Figure 12.2 (from line 1 to line 16). In computational terms, the main procedure is SETDILATION, which uses a function TRANSLATE. In algorithmic terms, it is easy to see that the method is derived directly from the definition, which justifies the soundness of the algorithm.

For a processor, access to memory accounts for a significant computational cost in reading mode but even more in writing mode. For simplicity, let us count only the number of memory allocations (that is, in the writing mode) to $true$. We can see that the trivial algorithm requires $|B| \times |X| \times 2$ allocations, where $|.|$ denotes the cardinality of a set.

A second version, derived from equation (12.2), reduces the number of allocations by half. It is detailed in Figure 12.2 by the DILDIRECT procedure (from line 32 to 44). The best approach, however, is given by the algorithm derived from relation (12.3). It improves significantly on the previous algorithms as the number of allocations is reduced to $|X \oplus B|$. Note that the multiplication of the sizes of X and B has been replaced by a summation on their sizes. The corresponding algorithm is detailed in Figure 12.2 by the procedure named DILREVERSE (from line 45 to line 60). This is the classical implementation of a set dilation, which can be found in several image processing software packages.

Even with such a simple morphological operator, it appears that there is a major difference between a concise mathematical definition and the transcription of it in an algorithm.

The case of the dilation is representative of some of the issues raised during the transcription of a definition to an algorithm. Next we consider a more complex algorithm to highlight algorithmic strategies for a single operator.

Most algorithms in mathematical morphology show a pseudo-polynomial complexity. For example, the trivial algorithm of dilation has a complexity of $\mathcal{O}(N \times |B|)$ where N and $|B|$ denote the number of points of the image and of the structuring element, respectively.

```
1  // algorithm (1)
2
3  SETDILATION(X : Image of 𝔹,
4                 B : Set of Point)
5    → Image of 𝔹
6  begin
7    data X_b, U : Image of 𝔹
8    // initialization to the empty set
9    U := false □
10   for_all b ∈ B
11     // computes X_b
12     X_b := TRANSLATE(X, b)
13     // updates U
14     U := UNION(U, X_b)
15   return U
16 end
17
18 TRANSLATE(X : Image of 𝔹,
19               b : Point)
20   → Image of 𝔹
21 begin
22   data O : Image of 𝔹
23   // initialization to the empty set
24   O := false □
25   // computes the set
26   for_all p ∈ X
27     if p + b ∈ E
28       O(p + b) := true
29   return O
30 end
31
32 // algorithm (2)
33 DILDIRECT(X : Image of 𝔹,
34               B : Set of Point)
35   → Image of 𝔹
36 begin
37   data O : Image of 𝔹
38   O := false □ // initialization
39   for_all p ∈ E
40     for_all b ∈ B
41       if X(p) = true and p + b ∈ E
42         O(p + b) := true
43   return O
44 end
45 // algorithm (3)
46 DILREVERSE(X : Image of 𝔹,
47                B : Set of Point)
48   → Image of 𝔹
49 begin
50   data O : Image of 𝔹
51   for_all p ∈ E
52     for_all b ∈ B
53       if p − b ∈ E and X(p − b) = true
54         // existence of a point b
55         O(p) := true
56         goto next
57     O(p) := false // there is no candidate b
58     label next
59   return O
60 end
```

Figure 12.2. *Dilation of a set X by B*

12.3. Taxonomy of algorithms

For different reasons, drawing up a taxonomy of algorithms used in mathematical morphology is a difficult task. First of all, several pages would not be sufficient to cite all existing algorithms; scientists have been continuing to propose new algorithms for operators that have been known for several decades! Also, we need a descriptive and complete set of criteria to propose a taxonomy that encompasses a large collection of algorithms. Finally, it has to be noted that there is no universal algorithm valid for any morphological operator. Algorithmic strategies are as diverse as the operators themselves with specific characteristics, tradeoffs, underlying data structures, etc.

12.3.1. *Criteria for a taxonomy*

Taxonomy criteria as applicable to mathematical morphology algorithms are numerous. As an illustration, a non-exhaustive list of criteria runs as follows:

– type of auxiliary or intermediate data structure (file, tree, etc.);

– order or strategy to browse points in the image;

– complexity of the algorithm;

– required memory size;

– algorithmic properties;

– operating conditions (and thus limitations) of an algorithm;

– concerned classes of operators or filters;

– universality of the algorithm;

– purpose of the algorithm; and

– processed data types.

Application constraints lead to additional possible taxonomy criteria: domain of applicability, data range and precision, application objectives, etc.

It would take too long to analyze algorithms with respect to all these criteria. Note, however, that some criteria lead to a classification of algorithms and that other criteria only allow them to be distinguished. Table 12.2 lists some criteria for both cases.

Table 12.3 elaborates on criteria useful to discriminate between algorithms.

12.3.2. *Tradeoffs*

An image processing chain is a particular case of the transcription of a scientific calculus by a processing unit. When part of this chain relates to a morphological operator, it is constrained by a general framework. In particular, a typical constraint is the need to find an appropriate balance between three antagonistic notions, described as follows:

– Expected computational time or speed: some applications need to be run in real time; others are less severe on the processing delay. The absolute execution time is only one part of the discussion related to the implementation of an algorithm. It is also important to analyze the variability of the running time. A hardware implementation will favor a constant execution time, even at the cost of an increased execution time.

– Storage space: resources proper to the algorithm in terms of disk space or memory usage are generally limited. Likewise, the amount of data handled by an application can be bounded. Therefore available storage capacity plays a crucial role in the choice of an algorithm and adequate auxiliary data structures.

Criterion: universality of the algorithm	
restricted	**large**
Decomposition of the structuring element (to speed up computations of dilations and erosions);	Dijkstra's algorithm (for the computation of a distance function); Viterbi's algorithm (for the pruning filter as described in Chapter 7); Prim's or Kruskal's algorithm (for a segmentation process based on the computation of the minimum spanning tree as described in Chapter 9).
Criterion: algorithmic properties	
parallel	**sequential**
Classical dilation or erosion algorithm by a structuring element (version (3) as described in section 12.2.5); detection of simple points	chamfer distance (see section 2.4.3); alternate sequential filters (see section 1.2.7)
Criterion: data range and precision	
small quantization step (of the values)	**large quantization step**
distribution sort or radix sort; use of tree representations (see section 7.2)	fast sorting, heapsort
Criterion: browsing order	
propagation of a front	**all points on turn**
distance based dilation; most algorithms to compute the watershed (see section 1.5 and Chapter 3)	trivial dilation; hit-or-miss transform (see section 1.1.4)

Table 12.2. *Criteria for a taxonomy of algorithms, with examples*

Criterion: auxiliary structures
arrays, files, priority queues, trees, graphs, etc.

Criterion: purpose
data simplification, resulting transform, computations/estimations of the data, partitioning, etc.

Criterion: processed data type
pixels, textures, objects, regions, contours, etc.

Table 12.3. *Discriminative criteria for morphological algorithms*

– The results of a computation have a given level of precision: they could be exact or approximated, which then requires elaboration on the expected level of precision. Many practical situations do not intrinsically require an exact calculation, or at least not for all points.

Practitioners are motivated by application requirements when they implement a morphological operator but they are constrained by the tradeoff triangle comprising three antagonistic notions, as illustrated on Figure 12.3. If execution speed is favored, then precision is lowered or computational resources are increased. Note however that modern architectures are capable of performing many morphological operations in real time so that questions about precision become meaningless.

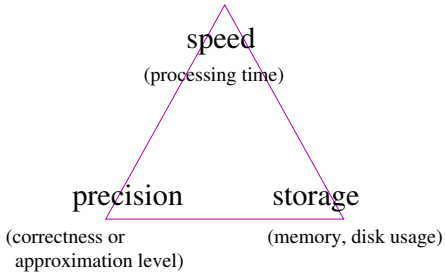

Figure 12.3. *Tradeoff triangle*

12.3.3. *Classes of algorithms and canvases*

Any algorithm running on images usually relies on the following features:

– one or several ways of browsing pixels;

– auxiliary data structures (images and/or any other kind of structures, classical or not);

– a processing rationale composed of several steps, mostly 'initialization, loops and finalization'; and

– a definition of neighborhood, crucial in mathematical morphology to inspect data around each pixel.

Let us consider the large set of algorithms dedicated to morphological operators. We observe that it can be split into several different groups of algorithms sharing the same algorithmic scheme, i.e. the same sequence of operations, the same use of control structures (such as loops) and the same auxiliary structures. This leads us to algorithm classes.

In the following we use the term *canvas* to describe a class of algorithms. Such a description looks like an algorithm template with some blank parts corresponding to the variability of algorithms belonging to this class. This canvas, or algorithmic scheme, is comparable to using a pattern to make clothes where the choice of fabric, color and ornaments remains to be defined.

```
1  POINTWISE(f : Image,
2                h : Function)
3     → o : Image
4
5  begin
6     for_all p ∈ E
7        o(p) := h(f(p))
8  end
```

```
1  SLIDING_WINDOW(f : Image,
2                 w : Window,
3                 h : Function)
4     → o : Image
5  begin
6     for_all p ∈ E
7        o(p) = h( { f(q) | q ∈ w(p) } )
8  end
```

Figure 12.4. *Left: pointwise canvas (left) and right: sliding window canvas*

Figure 12.4 depicts a couple of canvases that are very common in image processing. The left is a pointwise operator on image values; the value at the point p in the output image o is obtained by applying a function h to the value of p in the input image f : $o(p) := h(\, f(p)\,)$. With $h = C$ (complementation function), this canvas becomes the complementation operator; with $h = lum$ (luminance function), it is then a conversion of a color image into a grayscale image.

The canvas on the right-hand side of Figure 12.4 computes $o(p)$ from the set of values belonging to a window w centered at p in the input image. It is therefore the canvas of convolutions $\psi(f) = f * g$ when $h(\{f(q) \mid q \in w(p)\}) = \sum_q g(p-q)f(q)$, and of dilations $\psi(f) = \delta_w(f)$ when $h = \vee$ (supremum).

Algorithmic canvases are interesting for many reasons:

– Firstly, they are general. Each canvas is not specifically dedicated to a single particular operator. On the contrary, a canvas has the ability to be adapted to handle several operators; to that aim, we just have to define its variable parts (h in the previous examples).

– Secondly, they are intrinsically abstract. Their expression does not introduce any constraint that would restrict their use to a limited set of input data. For instance, having a double loop over the coordinates of points in the canvases of Figure 12.4 would implicitly state that they are only applicable on 2D images, which is clearly limitative.

– Lastly, they are helpful for educational purpose. Each canvas translates a meta-algorithm; understanding it from an algorithmic point of view allows us to comprehend any operator that can be derived from this canvas.

12.4. Geodesic reconstruction example

The different classes of algorithms presented in the following did not appear at the same time in the literature. They belong to a historical perspective, which is discussed in section 12.5.

To illustrate these classes, we present different algorithms that map the same morphological operator: the geodesic reconstruction by dilation of a function (e.g. a grayscale image) [VIN 93b].

This example (Figure 12.5), taking the form of an exercise of style, nicely illustrates different algorithmic schemes used by many other operators in mathematical morphology.

12.4.1. *The mathematical version: parallel algorithm*

As a first step, we can start by implementing the geodesic reconstruction by dilation from its mathematical definition:

$$\mathcal{R}_g^{\delta}(f) = \lim_{n \to \infty} \delta_g^n(f) = \delta_g^{\infty}(f) \tag{12.4}$$

where

$$\begin{array}{lcl} \delta_g^1(f) & = & \delta(f) \bigwedge g, \\ \delta_g^{n+1}(f) & = & \delta(\, \delta_g^n(f)\,) \bigwedge g. \end{array} \tag{12.5}$$

This reconstruction definition is close to that given in section 1.2.2; more precisely, it is its generalization to functions. Here δ is the geodesic dilation: $\delta(f) = f \oplus B$ where $B = \mathcal{N}(0) \cup \{0\}$ where $\mathcal{N}$ is the considered neighborhood and 0 is the spatial origin.

An implicit but compulsory assumption for this operator to be valid is that $f \leq g$. In other words, the marker function f to be dilated shall be 'under' the mask function g. As in the case of sets, the reconstruction of functions aims to reconstruct some details of g from a simplified image f.

The definition of this reconstruction is itself an algorithm: it is the result of iterations repeated until convergence of a geodesic dilation followed by a pointwise condition.

Such a recursion is implemented with a loop and the algorithm terminates when the result is stable (when no modification has been noted during the last iteration). The convergence of this algorithm is mathematically ensured, yet it is very slow in practice. Indeed, consecutive passes reconsider parts of the image where local convergence has been reached during previous steps. This algorithm is illustrated by the routine RD_PARALLEL in Figure 12.5.

```
RD_PARALLEL(f : Image,
            g : Image)
  → o : Image
begin

  data
    o' : Image
    stability : 𝔹

  // initialization
  o := f

  // iterations
  repeat
    o' := o // swap

    // dilation\ldots
    for_all p ∈ E
      o(p) := max{ o'(q) |
                   q ∈ 𝒩(p) ∪ {p} }


    //\ldotsunder condition
    for_all p ∈ E
      o(p) := min{ o(p), g(p) }

    stability := (o = o')
  until stability
  return o

end
```

```
RD_SEQUENTIAL(f : Image,
              g : Image)
  → o : Image
begin

  data
    o' : Image
    stability : 𝔹

  // initialization
  o := f

  // iterations
  repeat
    o' := o // memorization

    // first pass (forward)
    for_all p ∈ E ▷
      o(p) := min{ max{ o(q) |
                  q ∈ 𝒩⁻(p) ∪ {p} }, g(p) }

    // second pass (backward)
    for_all p ∈ E ◁
      o(p) := min{ max{ o(q) |
                  q ∈ 𝒩⁺(p) ∪ {p} }, g(p) }

    stability := (o = o')
  until stability
  return o

end
```

Figure 12.5. *Reconstruction canvases (part 1/2): parallel algorithm (left) and sequential algorithm (right), described in sections 12.4.1 and 12.4.2, respectively*

12.4.1.1. *Similar algorithms*

The 'repeat modifications until stability' type of algorithm does not belong exclusively to the area of mathematical morphology; it is also used, for instance, to compute diffusions.

The complexity of one-pass is pseudo-polynomial with respect to the number N of image points and to the connectivity M. In the worst case (as for a Peano curve image), this algorithm has a complexity of $\mathcal{O}(M \times N^2)$. Practically, no one should be using this algorithm except perhaps on some strictly parallel computing architectures.

A point in favor of this algorithm is that it is easily parallelizable. We observe that the computations performed at every point within both loops only depend on the image obtained at the end of the previous iteration. Those loops can therefore be split into several independent tasks, each task running on a single part of an image partition. From this point of view, this algorithm contrasts with the alternative algorithms presented. With the present parallel version, there is no dependence between the computation at point p and the computation of its neighbors during the same iteration.

12.4.2. *Sequential algorithm*

Obtaining an acceptable complexity for the reconstruction can be achieved by noting that this filter can be expressed in a sequential way. However, we will see that this is not sufficient.

In the parallel version (lines 18 and 20 of Figure 12.5), each dilation is performed independently from those of previous iterations. The current image o is obtained by dilation of the image o' resulting from the previous iteration.

In the sequential version, the auxiliary image o' is no longer used during the dilation step; each dilation is performed in-place (see lines 51 to 57 of Figure 12.5). At a given point p, the local dilation value is computed from the neighboring values of p in o and immediately written in the work image o. Consequently, a modification of o appearing at point p can propagate to other points during the same pass.

In order to ease the comparison between the parallel and the sequential algorithms, we keep a copy o' of o to test the stability condition. Note that we can get rid of o' if we count the number of modifications in o during the forward-backward pair of passes to test stability.

Despite the propagation mechanism, the complexity of the sequential algorithm is not improved with respect to the parallel algorithm! Again, in the particular unfavorable case of an image representing a Peano curve, the reconstruction requires a number of passes proportional to the number of pixels. However, for convex objects, a forward step and a backward step are sufficient to obtain the result. In practice, natural images have a lot of such locally convex parts; these parts are processed efficiently. This explains why we the sequential algorithm usually outperforms the parallel version.

12.4.2.1. *Similar algorithms*

The class corresponding to this sequential algorithm is large. In particular, it includes discrete distance map computation, such as chamfer distances, and the pseudo-Euclidean distance of Danielsson [DAN 80].

```
65  RD_QUEUE_BASED(f : Image,
66                 g : Image)
67    → o : Image
68  begin
69    data
70      q : Queue of Point
71      M : Image
72
73    // initialization
74    M := REGIONAL_MAXIMA(f)
75    for_all p ∈ M
76      for_all n ∈ N(p)
77        if n ∉ M
78          q.PUSH(p)
79    o := f
80
81
82
83
84
85
86
87    // propagation
88    while not q.EMPTY()
89      p := q.FIRST()
90      for_all n ∈ N(p)
91        if o(n) < o(p) and o(n) ≠ g(n)
92          o(n) := min{o(p), g(n)}
93          q.PUSH(n)
94
95    return o
96  end
```

```
98  RD_HYBRID(f : Image,
99            g : Image)
100   → o : Image
101 begin
102   data
103     q : Queue of Point
104
105   // initialization
106   o := f
107
108   // two−pass sequence\ldots
109   for_all p ∈ E ▷
110     o(p) := min{ max{ o(q) |
111                 q ∈ N⁻(p) ∪ {p} }, g(p) }
112   for_all p ∈ E ◁
113     o(p) := min{ max{ o(q) |
114                 q ∈ N⁺(p) ∪ {p} }, g(p) }
115     //\ldotswith enqueuing
116     for_all n ∈ N⁺(p)
117       if o(n) < o(p) and o(n) < g(n)
118         q.PUSH(p)
119
120   // propagation
121   while not q.EMPTY()
122     p := q.FIRST()
123     for_all n ∈ N(p)
124       if o(n) < o(p) and o(n) ≠ g(n)
125         o(n) := min{o(p), g(n)}
126         q.PUSH(n)
127
128   return o
129 end
```

Figure 12.6. *Reconstruction canvases (part 2/2): queue-based algorithm (left) and hybrid algorithm (right) are described in sections 12.4.4 and 12.4.3, respectively*

12.4.3. *Queue-based algorithm*

In a queue-based algorithm version of the reconstruction (Figure 12.6), another data structure is used: a First-In-First-Out (FIFO) queue structure. The general idea is to dilate through a front that propagates into the whole image while remaining under the condition imposed by g (line 92).

The two main advantages of this approach are its very simple formulation and the one-pass dilation. Contrary to the previous versions, the need to scan the image pixels

several times is avoided, thus meaning that useless operations (repeatedly accessing stable regions) are avoided and that complexity is significantly lowered.

Most queue-based algorithms rely on the same scheme:

– the queue is initialized with some particular points;

– while the queue is not empty do:

1) remove the point at the front of the queue;
2) perform a given computation at that point; and
3) add some neighbors of that point at the back of the queue.

This scheme is in technical terms a breadth-first traversal of the neighborhood graph of the image starting from the points pushed in the queue during the initialization step. Other types of browsing are possible; for instance, we get a depth-first traversal if we swap steps 1 and 3.

In the case of the reconstruction, the initialization starts from the detection of regional maxima and the points of their external contour are pushed in the queue. Those maxima are then propagated by the queue. The operation managed by the queue is not really a 'common' dilation, in the sense that the propagation does not perform a dilation with a structuring element or a geodesic element. Only maximal values of f are dilated. However, it is an algebraic dilation (see Chapter 2). Since the dilation is only effective under the condition imposed by the mask g, we obtain the expected reconstruction by dilation as defined previously [VIN 93b].

The core of the algorithm, given in the left column of Figure 12.6, lies in the propagation process (lines 87 to 93). We can easily note that every point p is only inspected once, which contrasts with the previously given algorithms. This part of the algorithm therefore has a linear complexity.

There is a price to pay for this reduced complexity:, the initialization step requires the computation of the regional maxima of f (line 74), which can be as costly as the propagation step. Such an operation is equivalent to a connected component labeling, the complexity of which is quasi-linear due to the union-find algorithm [TAR 75].

More generally, using a random-access structure such as a queue or a stack is efficient if the initialization stage extracts adequate information. We can relate that issue to the notion of redundancy of information. In a dilation process, relevant information is the localization and the value of local maxima. Algorithms can either ignore such information (those are content-blind algorithms), or rely on this information and propagate it. Amongst the fastest (and also the most complex) algorithmic approaches, we have those that detect relevant information *and* propagate it while scanning image points.

To understand how detection and propagation can be merged into a single step, we can refer to the algorithm proposed in [VAN 05] for the morphological opening. The queue-based reconstruction given in this section does not feature such an elaborate scheme; in the following, we see that it is possible to obtain a more efficient reconstruction than the current version.

12.4.3.1. *Remarks*

12.4.3.1.1. Similar algorithms

Algorithms similar to the queue-based reconstruction include:

– for the breadth-first traversal: distance functions [RAG 93], SKIZ, skeletonization [VIN 91a] and ordered dilations [ZAM 80]; and

– for the depth-first traversal: the component tree computation given in [SAL 92].

12.4.3.1.2. Stacks instead of queues

Using a stack (either a simple Last-In-First-Out structure or a pushdown automaton) instead of a queue can be relevant when combined with the storage of some information related to data. It should be avoided, however, when the behavior of the algorithm is recursive. Indeed, in such cases, computation calls are stacked so that they can be executed later. Even if the algorithm is theoretically correct, it may become inefficient when the size of the stack grows too much (note that its size may be as large as the image).

12.4.3.1.3. Priority queue

A simple queue ensures only one property of the contained elements: they are ordered as they were pushed in the queue. This ordering usually corresponds to a particular spatial ordering of the image domain. It is sometimes useful to obtain a more general property of the element ordering in the queue. For instance, we may want points to be sorted first by their values and then by their introduction order in the queue.

To that aim, a particular structure was proposed by Meyer in the context of a watershed transform algorithm: the hierarchical queue [MEY 91]. This is an array of queues of size as large as the number of values; it is therefore efficient if and only if the values are encoded with a small number of bits (typically less than 12). Note, however, that it is a particular case of the more general and very common *priority queue* structure, that can be implemented using a heap or a self-balancing binary search tree.

12.4.3.1.4. Complexity

Some data structures are more appropriate than others depending on the algorithms, on the nature of input images and on the operations that rely on those

structures. The most classical data structures that we can find in morphology have been studied by Breen and Monro [BRE 94]; furthermore they also emphasized the distance existing between theory and practical results, related to the use of those structures.

In particular, Fibonacci heaps [FRE 87] (although theoretically efficient) are rather slow when involved in effective algorithms. When an efficient priority queue is required, maintaining a stable sort (i.e. with the insertion order of elements always preserved in the queue) regardless of the data type used for priorities, *splay-queues* are often an appropriate choice [SLE 85].

12.4.4. *Hybrid algorithm*

In the hybrid algorithm, given in the right-hand column of Figure 12.6, we first note a sequential part, that is limited to a couple of passes. During these passes, the reconstruction is performed in convex regions of the input image. In the second part of the algorithm, we have the propagation of the queue-based algorithm. This last part completes the reconstruction until convergence is achieved.

The advantage of this method over the previous one is manifold. The computation of regional maxima is avoided since the queue is initialized with the frontier obtained after the sequential passes. Furthermore, in the case of non-pathological images, most of the reconstruction is actually achieved during this sequential part and the final propagation, much more costly, is only performed on a small part of the image domain.

Note that hybrid algorithms, where an actual synergy exists between the different approaches that they combine, are rather rare in the literature.

12.4.4.1. *Remarks*

As before, the worst case scenario for this algorithm is again images representing fractal patterns. In these cases, the queue size remains small but many loops are required for the algorithm to converge.

A method of obtaining better performances from this algorithm is to use a queue implemented by a circular array (more compact in memory and faster when inserting and suppressing elements).

This hybrid version highlights the important relationship between the algorithm itself and the data structures it relies on. Authors usually study the complexity of the algorithms they propose with respect to the number of manipulations of image data, that is the number of input/output (reading/writing) performed on points. Unfortunately many authors forget to take into account the effective cost of the auxiliary structures involved in those algorithms. The simplest form of such structures is a data buffer in memory, where even a writing operation is not negligible (dependent

```
MAKE_SET(p : Point)
begin // creates singleton { p }
  parent(p) := p
end

IS_ROOT(p : Point) → 𝔹
begin // tests if p is root
  return parent(p) = p
end

FIND_ROOT(p : Point) → Point
begin // finds the root of p
  if IS_ROOT(p)
    return p
  else
    parent(p) := FIND_ROOT(parent(p))
    return parent(p)
end

DO_UNION(n : Point, p : Point)
begin // merges two trees
  r := FIND_ROOT(n)
  if r ≠ p
    if g(r) = g(p) or g(p) ≥ o(r)
      parent(r) := p
      o(p) := max(o(r), o(p))
    else
      o(p) := MAX
end

RD_UNION_FIND(f : Image,
              g : Image)
 → o : Image
begin
  data
    parent : Image of Point
    S : Array of Point

  // initialization
  o := f
  S := SORT(g) // w.r.t. ▷ and g(p) ↓

  // first pass
  for_all p ∈ S
    MAKE_SET(p)
    for_all n ∈ 𝒩(p) if DEJA_VU(n)
      DO_UNION(n, p)

  // second pass
  for_all p ∈ S^-1
    if is_root(p)
      begin
        if o(p) = MAX, o(p) := g(p)
      end
    else
      o(p) := o(parent(p))

  return o
end
```

Figure 12.7. *Reconstruction by dilation with union-find, described in section 12.4.5*

upon the buffer size and the amount of RAM available). On the other hand, being able to precisely characterize the cost of an algorithm including its auxiliary structures is a tricky task, since it is also highly dependent on machine hardware.

12.4.5. *Algorithm based on union-find*

The union-find algorithm (Figure 12.7) is an identification of the equivalence classes of a graph. This algorithm is relatively complex; we therefore only present a rough guide here. It is composed of three steps: an initialization, a union stage and a labeling stage (find).

12.4.5.1. *Rough sketch of the algorithm*

The cornerstone of this algorithm lies in a change of representation for images: we move from the notion of pixels, with no connectivity information except purely local, to the structure of a tree, where a node can represent a large connected component of an image. In this tree, the root node maps the whole image domain whereas leaves relate to local components.

During the initialization, the points of image g are sorted by decreasing value of gray levels and stored in the array S (line 169).

During the union stage, points are scanned and stored in S. The current point can be isolated in the sense that its neighbors have not yet been inspected; it therefore belongs to a regional maxima. This point then forms a singleton set. If not isolated, it is connected to a regional maximum of g; it is then merged with the corresponding tree and becomes its new root. A key property to understand this algorithm is to realize that this regional maximum of g is related to a regional maximum of f. During the process of browsing S, a forest of trees is created that progressively spans the image domain.

In the final phase (find), we handle the points where $g > f$ differently. Those in the connected component containing the local regional maximum of g receive the maximum value of f, and the value of points where $g = f$ are kept unchanged.

Eventually f has been dilated under the constraint of g and the result is the expected reconstruction.

12.4.5.2. *Details*

During the union stage, the output image o is used as auxiliary data to store the state of all components/trees: either $max(f)$ when dilating is effective ($f < g$ locally) or MAX when the constraint applies. o only takes its final values during the last step of the algorithm.

For every flat zone where it is guaranteed that we will obtain $o = g$, no tree is computed and the flat zone points are all singletons.

The DEJA_VU function can be evaluated on the fly so this auxiliary structure can be saved.

If g is an image with low-quantized values (for instance an 8-bit image), we can sort points in linear time due to a radix sort (a distributed sort based on histogram computation).

12.4.5.3. *Complexity*

In this chapter, a very particular version of the union-find based algorithm is presented (Figure 12.7). It relies on a path compression technique, embedded in the FIND_ROOT routine (line 145), so that the number of recursive calls to this routine is reduced. It is not however sufficient to obtain the best complexity of the union-find algorithm. To minimize the number of recursive calls, it is also necessary to keep all trees as balanced as possible. For that, we have to add to the version presented here the 'union-by-rank' technique. We have intentionally omitted this technique to make this algorithm more readable.

Actually, the union-find based reconstruction is quasi-linear in the case of g being a low-quantized image [TAR 75]; otherwise, for floating data for instance, the complexity is $\mathcal{O}(N \log(N))$ due to the sorting step.

12.4.5.4. *Comparison with previous versions*

Although the union-find algorithm is more efficient in theory than the other presented versions, in practice it is not always faster than the hybrid algorithm. However it is emblematic of modern implementations of connected operators: algebraic attribute openings and closings, levelings, watershed transforms [BRE 96, GÉR 05, JON 99, MEI 02]. The representation of image contents as a tree forest allows for an exceptionally rich theoretical description of connected operators [FAL 04, NAJ 06]. See also Chapters 7 and 9.

With the parallel and sequential approaches, the limiting factor with respect to complexity was the number of passes to perform to reach convergence. With the queue-based and the hybrid approaches, the risk came from the queue structure becoming too huge. In the case of the union-find algorithm, the bottleneck is located in the tree root search.

12.4.6. *Algorithm comparison*

In order to compare the five algorithms described here, we shall reuse some of the criteria presented in section 12.3.1. Table 12.4 below illustrates the large diversity of algorithms available to translate a single operator and the difficulty in using the criteria effectively. Indeed, some of these algorithms are not monolithic: they correspond to several criteria at once. The hybrid algorithm, for instance, is only half-sequential: it uses both video passes on the image and a propagation front. Regarding the union-find algorithm, we could classify it as sequential as between the initialization (the sorting pass) and the two other passes, every image pixel is considered exactly three times. However, the order in which they are considered are not the usual video and anti-video sequences.

Algorithm name	Algorithm class	Pixel order	Data structures
Parallel	Parallel	Video passes	None extra
Sequential	Sequential	Video passes	None extra
Queue-based	Queue-based	Front	Standard queue
Hybrid	1/2 sequential	2 Passes + front	Standard queue
Union-find	Pseudo-sequential	3 Passes	Array and tree

Table 12.4. *Characteristics of the various reconstruction algorithms presented in the text*

To compare the performance of these five algorithms for the geodesic reconstruction by dilation (Table 12.5), we took for g the standard `lena` 512×512 pixels gray-level image and for f the pixelwise maximum of g and g rotated 90 degrees clockwise. The neighborhood relation is the 4-connectivity. As we seek a comparison between algorithms, we did not perform any compiler-level optimization or used particular techniques such as pointer arithmetic (although they could have resulted in significantly improved running times). The algorithm performance order would have remained the same, however.

Algorithm	Running time (sec)
Parallel	25.28
Sequential	3.18
Queue-based	0.65
Hybrid	0.34
Union-find	0.34

Table 12.5. *Relative performance of the reconstruction algorithms presented in the text*

Table 12.5 illustrates, with performance ranging within a rough factor of 100 to 1, that translating a mathematical operator into actual code can in practice result in widely different results. Producing a 'good' algorithm with properties in accordance with what the practitioner expects is indeed an art and a science in itself.

12.5. Historical perspectives and bibliography notes

The history of various algorithms and their links to mathematical morphology as well as other related disciplines would require a book by itself. We provide in this section a few (hopefully) useful notes.

12.5.1. *Before and around morphology*

Like all scientific endeavors, mathematical morphology was not born and neither did it evolve in a scientific vacuum. It represents a step towards a better understanding of spatial representations pertaining to physical or virtual objects. Before Matheron and Serra named their discipline in 1962 [MAT 02], image analysis already existed and many algorithms had already been developed in comparable disciplines. Morphology also continued to evolve in a moving context. It is perhaps useful to specify some algorithmic markers in this creative broth.

12.5.1.1. *Graph-based algorithms*

Images are most often represented on regular graphs. As such it is not really surprising that so many mathematical morphology algorithms derive from classical graph algorithms. We may mention Dijkstra's minimal paths [DIJ 59], the minimum spanning tree problem [JOS 56, PRI 57], and the classical union-find algorithm [TAR 75]. A good source in many important algorithms is the book by Cormen *et al.* [COR 09].

It is more than likely that not all the classical or recent literature on algorithms on graphs has been fully exploited in the context of mathematical morphology. It is probably a very good source of future results.

12.5.1.2. *Discrete geometry and discrete topology algorithms*

Discrete geometry is an active field of research very closely linked to mathematical morphology. The goal of discrete geometry is to redefine and algorithmically exploit the objects and operators of classical geometry, in a purely discrete framework. For instance lines, planes, intersections, vectors and so on have been partially redefined in such a way [REV 91]. Properties of these new objects, although obviously related to the classical objects, are markedly different and usually much more amenable to their use in an algorithmic setting. A recent book on the topic is [GOO 04].

Among the most useful algorithms in discrete geometry, also used in mathematical morphology, we can mention distance transforms [BOR 84, ROS 66, SHI 92] that are very useful by themselves, but can also be used to implement binary erosions and dilations [RAG 92].

Discrete topology is a discipline that seeks to define topological operators in discrete spaces such as images, but also on arbitrary graphs, on discrete manifolds such as triangulated surfaces or on complexes [BER 07a, KON 89]. The link with morphology is very strong especially in the areas of thinning operators [KON 95] and skeletonization algorithms [MAN 02]. The watershed transform can also be seen as a topological operator [COU 05]. The *topological watershed*, of gray-level image I, for instance, is the smallest image on the lattice of numerical functions with the same

topology as I. The topological watershed operator is also the most efficient watershed algorithm (with quasi-linear complexity).

12.5.1.3. *Continuous-domain algorithms*

The continuum is not representable exactly on computers; however, some mathematical objects exists that are intrinsically continuous (such as partial derivative equations), but that can be discretized with known properties (for instance, up to second order accuracy using finite differences). These can be used to solve some image analysis problems. This approach leads to some interesting algorithms.

Taking as a starting point segmentation algorithms such as active contours [KAS 98], it is possible to find links with skeletonization [LEY 92] as well as generalizations of the watershed transform, for instance including some curvature constraints [NGU 03].

Fast marching algorithms [SET 96a] are in essence equivalent to a flexible algorithm for computing the geodesic Euclidean distance transform [SOI 91]. This applies to scalar or tensorial metrics [SET 01]. These algorithms make it possible to propose, in some contexts, a mathematical morphology formulation in the continuous domain [SAP 93]. It is important to note that the original Sethian algorithm is only first-order accurate. A fast method to compute the exact geodesic Euclidean distance transform is an open problem at the time of writing.

These methods have been used in morphology for instance in connected filtering [MEY 00b], by replacing the dilation operator by a continuous propagation. Formulation of the watershed transform in the continuous domain have been proposed by several authors [MAR 96a, NAJ 93] and can be solved using fast marching methods.

The principal benefit derived from a continuous formulation is to abstract away the notion of pixel. Up to an approximation, it is possible to define a dilation of arbitrary, and not only integer, radius. It is also possible to propose morphological operators on arbitrary manifolds, for instance on triangulated surfaces, although discrete formulations have also been proposed in this context.

We will explore other links with the continuous domain through algorithms inspired by linear theories.

12.5.1.4. *Discrete and continuous optimization for segmentation*

Active-contour [KAS 98] or level-set [OSH 88, MER 94, SET 96b] types of algorithms are comparable in their approach to segmentation. The idea is to propose a gradient-descent optimization procedure, under some constraints. Common constraints include the necessity of closed contours, the inclusion and/or exclusion of

certain zones and topology preservation. These algorithms work in 2D or 3D and are fairly flexible with respect to the kind of cost function they can optimize. For instance, it is possible to affect costs to region content, motion analysis, regularity, etc. However, the more complex formulations most often cannot be optimized globally.

More recently, the image analysis and computer vision communities have found a renewed interest in simpler formulation, which can be optimized globally, for instance using graph cuts [BOY 04], continuous maximum flows [APP 06] or random walks [GRA 06]. Indeed these formulations are less flexible, but are more reliable and less sensitive to noise. There are some strong links between these techniques and the watershed transform [ALL 07, COU 09a].

12.5.1.5. *Linear analysis algorithms*

Here linear analysis means the domain of operators linked to linear integral transforms, such as the Fourier, Radon and wavelet transforms. These were historically adapted to images from signal processing. In this domain, the basic structure is the classical group with addition as base operator. For signals and some types of images (X-ray or tomography) this makes perfect sense as superposition of signals is a reasonable hypothesis. For some types of problems this is also a perfectly suitable structure. For instance, many sources of noise such as sampling noise are approximated by Gaussian additive white noise, for which there exists an optimal deconvolution in the least-square sense.

Mathematical morphology is not linear, and the basic structure is the complete lattice with infimum and supremum as operators. However there are some links between the two approaches. This is of course true at the level of applications, but some tools and approaches are also similar. As an example we can cite several works on multiresolution [HEI 00] and scale space [JAC 96, VAC 01b].

To complete this section, we present a curiosity: it is possible to define the dilation from a convolution operator

$$\delta_B[I] = (I \star B) > 0, \tag{12.6}$$

where I is a binary image and B an arbitrary structuring element. Using the FFT, this algorithm can be implemented in constant time with respect to B; this is the only known implementation with this characteristic.

12.5.2. *History of mathematical morphology algorithmic developments*

From the very beginning, the development of mathematical morphology as both a practical and theoretical area was linked to software and hardware developments. The *texture analyzer* was the first machine implementing morphological operators, and was

developed at the École des Mines in Paris [MAT 02]. Following this, many advances in this field were the result of a constant synergy between applications, theory and algorithmic and hardware developments.

In the early days, dedicated architectures for image processing were a necessity due to the relatively weak computing power of general-purpose architectures. On dedicated hardware, access to data could (most often) only be realized in an ordered sequential manner using video passes on the image. This also drove the development of corresponding algorithmic techniques.

12.5.2.1. *Parallel algorithms*

The use of video passes on the images and the limited memory of early architectures (typically only three lines could be loaded in main memory at any given time) is a limitation that is still found today, for instance in embedded architectures such as mobile phones. These limitations, amongst others, force the use of *parallel* algorithms. Here this term means a type of processing such that the result on any arbitrary pixel is independent of the result on other pixels. This implies order-independence. It also implies that it is relatively easy to implement such algorithms on massively parallel architectures, but the actual architecture the algorithm runs on is not so important.

An illustration of this type of algorithm is given in Figure 12.5 on the left-hand side, and is also described in section 12.4.1. In hardware terms, these algorithms are well-suited to SIMD *single instruction multiple data* and to the limit case of the so-called *artificial retina*, where each pixel is equipped with its own little processor [MAN 00]. Among hardware developments that were known to use parallel morphological algorithms are the Morpho-Pericolor [BIL 92] and the Cambridge Instrument Quantimet 570 [KLE 90] as well as the ASIC (application-specific integrated circuit) PIMM1 [KLE 89].

The first algorithms implementing the watershed transform, the skeletonization and morphological filters were described and implemented in a parallel fashion. See for instance [BEU 79a, MON 68].

12.5.2.2. *Sequential algorithms*

Some (but not all) algorithms can be expressed in a *sequential* manner, which here designates an implementation that uses the current result to derive the next one, most often adopting a particular pixel scanning order. This is illustrated in Figure 12.5 on the right-hand side and is described in section 12.4.2.

Sequential algorithms, sometimes also incorrectly called *recursive* algorithms are often more efficient than parallel algorithms at least on most general-purpose computers. This is the case because they can make use of the local redundancy in many natural images. A typical sequential algorithm is the classical distance transform

of Rosenfeld *et al.* [ROS 66], which computes the distance transform in two passes over the image: one in the video scanning order and the second in the anti-video order. At the hardware level, few sequential algorithms have been implemented, but Lemonnier [LEM 96], among others, has proposed a sequential watershed transform algorithm.

12.5.2.3. *Breadth-first algorithms*

As general-purpose computers became more powerful, the idea of exploring pixels from the border of objects without necessarily following a scanning order imposed by the hardware (in particular the memory wiring) became more popular. This is achieved using a suitable data structure. Among this family of algorithms, we mention those using boundary paths [SCH 89], queues [VIN 90] and priority queues [MEY 90a]. A classical algorithm belonging to that class is the watershed transform from flooding [MEY 90b, VIN 91c]. This kind of algorithm is not well suited to hardware implementations, mostly because the underlying data structure imposes that memory bandwidth be the limiting factor (and not computation speed).

12.5.2.4. *Graph-inspired algorithms*

Breadth-first algorithms are a classical approach in graph-based problems. The idea of continuing in this direction and adapting other classes of graph algorithms to image data was therefore natural. Among graph-inspired morphological algorithm, we can cite the Image Foresting Transform (IFT) [FAL 04], which is used in segmentation and classification.

More recently, the idea of considering an image truly as a graph and also to assign values to edges took hold. This makes it possible to define a discrete gradient in a natural way: simply by the numerical difference between two vertices linked by an edge [COU 07d]. This had already been proposed earlier by the graph-cut community [BOY 01, BOY 04]. This notion defines a border between regions as a series of edge cuts and not a path of vertices, which solves numerous topological problems. It also paved the way for a unifying framework encompassing many segmentation methods [COU 09a].

12.5.2.5. *Topological algorithms*

Beyond the essential notion of simple point, which is the starting point for many efficient topology-preserving algorithms, many works have considered the essential notion of image topology. An important notion is the component tree (section 12.4.5.1 and Chapter 7). The component tree, through its efficient representation of regions and catchment basins, can be used in many interesting algorithms involving for instance hierarchical segmentation, levelings and other filters [NAJ 06].

The staple of morphological algorithms, the watershed transform, can also be seen as a topological transform [BER 05, COU 05]. For details on the topological watershed, see Chapter 3.

12.5.2.6. *Morphological filtering algorithms*

Morphological filtering algorithms form an interesting class by themselves. As a starting point, useful literature on morphological filtering includes the two books by Serra [SER 82, SER 88b], an article on the theory of morphological filtering by Serra and Vincent [SER 92c] and the articles by Heijmans and Ronse [HEI 90, RON 91]. A more introductory article by Heijmans is [HEI 96]. None of these articles discuss algorithmic aspects, which are nonetheless essential. The following is an incomplete but illustrative list of some problems studied in mathematical morphology.

12.5.2.6.1. Fast erosions and dilations

The topic of fast implementations of basic morphological operators has been studied by many authors. In spite of this, many libraries of mathematical morphology software (including well-known and expensive ones) compute a min or max filter on a window using $O(MN)$ comparisons, with M the number of pixels in the image and N the number of pixels in the window. It is often possible to decompose structuring elements (SE) into more readily computable subparts [XU 91]. Among the most commonly used SE are the regular convex polygons in 2D. These can easily be decomposed into operations using line segments.

A significant achievement by the community has been to propose increasingly efficient algorithms to compute the basic morphological operators in arbitrary 1D segment windows, including arbitrary orientation [BRE 93, GIL 02, HER 92, VAN 05]. As a result, the computation of all four basic morphological operators with convex regular polygonal windows can be achieved in constant time with respect to N in 2D. Note that at the time of writing, an equivalent result in 3D or more is still an open problem, except for some particular cases.

Regarding arbitrary structuring elements in n dimensions, there exists an algorithm with complexity $O(\sqrt[n]{N^{n-1}}M)$ [VAN 96]. A faster algorithm for 2D but extendable to more has been proposed [URB 08]. Several algorithms have been proposed in the binary case [JI 89, VIN 91b], with complexity asymptotically linear with respect to M. The FFT-based algorithm mentioned in section 12.5.1.5 has complexity $M \log M$.

12.5.2.6.2. Algebraic openings and thinnings

Filtering in mathematical morphology tends to rely more on openings and closings than erosions and dilations. It is common to define a notion of opening or closing that is not directly related to that of structuring element, but is rather based on the concepts of *attribute* and connectivity [CRE 93b, CRE 97b, HEI 99]. These ideas are close to the notion of reconstruction seen in this chapter, and were also presented in Chapter 1.

Due to connected and attribute filtering, many very effective operators were proposed in the last decade. Historically, from the algorithmic point of view, the first implementation of a connected filter is due to Vincent [VIN 92, VIN 93a, VIN 94]

with the area filter. The general notion was extended to attributes [BRE 96] that are not necessarily increasing, leading to operators that were no longer openings or closings but *algebraic thinnings* (using very similar principles). An efficient implementation was proposed in [MEI 02], followed by a generalization using the component tree and the union-find in [GÉR 05]. More recently, the notion of connectivity was extended to *hyper-connectivity* [WIL 06] to account for overlaps.

Path connectivity is also both a topological and a connectivity notion. By adding constraints to acceptable paths, from straight line segments [SOI 01] to more flexible paths [HEI 05, TAL 07], it is possible to enable the filtering of notoriously difficult thin objects in various applications [VAL 09a, VAL 09b].

12.5.2.6.3. Spatially variant filtering

More recently, efficient operators using filtering by non-translation-invariant (or *spatially variant*) filters have been proposed. From the theoretical point of view, these operators have been known since Serra [SER 82], but were recently given more theoretical treatment [CHA 94, BOU 08a, BOU 08b]. This kind of filtering method is adaptive in the sense that a different structuring element is used at each point, depending on the local content of the image (for instance depending on the orientation, perspective or texture) [LER 06b, VER 08]. These filters can be effective in the context of inverse filtering for thin feature extrapolation [TAN 09a, TAN 09b].

12.5.2.6.4. Extension to n dimensions

Mathematical morphology is, from the theoretical point of view, largely dimensional-agnostic, meaning most operators can be defined irrespective of the dimension of the underlying space [GES 90, GRA 93]. However, there are some practical difficulties as dimension increases. For instance, from the geometrical and topological point of view, while the hexagonal grid is a useful vertice arrangement in 2D that is naturally relatively isotropic and self-dual with respect to connectivity, no such arrangement exists in 3D and little is known of higher dimensions.

From the direction sampling point of view, which is often used in orientation-based filtering, it is possible to sample the 2D plane directionally in such a way that is both regular and of arbitrary resolution (say every degree or more or less). This is impossible in 3D and more: a result known since Platon and Euclid [HEA 56]. 3D (and more) filtering requires more resources of course but, thanks to recent advances in sensors, instruments and computers, it has become increasingly common and important. Application fields include medical imaging, materials science, biological and bio-molecular imaging.

12.6. Conclusions

In this chapter, we have sought to express the distance that exists between the mathematical formulation of an operator and its actual algorithmic translation. From a simple but representative example, we have also shown that in general there does not exist a single best way to express the implementation of an operator but several (for which characteristics can be markedly different).

Algorithmic research, whether dedicated to mathematical morphology or not, remains a wide open field. As time progresses, increasingly sophisticated operators are being proposed, with correspondingly demanding computational loads together with the ever-increasing size and complexity of the data itself. This can only mean that it will become increasingly important to devote sufficient time and resources to the development of efficient implementations of image analysis operators, whether this implementation be in hardware or software.

We can also anticipate a few fresh challenges on the algorithmic frontier.

First, external and internal observers (for instance users but also article reviewers) have noticed increasing difficulties in reproducing methods and results presented in the scientific community. From a scientific article, the way leading to an actual piece of working code can be very long. To the understanding of the proposed operators and algorithms can be added the difficult and painful programming and debugging tasks. As a direct consequence, a loss of information capital and of knowledge is observed. Many, if not most, solutions proposed in the literature are simply abandoned or ignored; few articles propose a comparison with a significant number of solutions. In our opinion, the morphological community should make the effort to endow the public at large with a working library of computer code implementing its efforts. This could take the form of a mutual, open platform for code repository.

A second challenge concerns the implementation of algorithms. Algorithms are by their nature abstract. Indeed, in this chapter we have kept an abstract presentation as much as possible. This reflects the fact that they might work just as well on a 1D signal as on 3D volume data, irrespective of sampling, grid and topology issues, unless specified. Unfortunately, the actual translation of algorithms to code is almost inevitably accompanied by a loss of generality: such a library of code will only work on 2D, gray-level, square grid images. Another will be devoted to satellite images, yet another to 3D medical volumes, and so on. Most libraries do not accept arbitrary-shaped structuring elements, for instance. Even a ‘simple’ dilation as given by algorithm (3) in Figure 12.2 becomes, once implemented, a dilation restricted to a limited number of cases. We note, however, that generic solutions exist, allowing users to apply algorithms to vastly different datasets without necessarily sacrificing efficiency [LEV 09].

A third challenge concerns community effort. It is increasingly understood that to be acceptable during and after publication, to be reviewed effectively and to be cited, an algorithm description should be accompanied by an implementation, freely accessible to the researcher or individual user. Indeed, re-implementation efforts are usually simply duplicated work. In other communities such as computer vision, discrete geometry or computational geometry, active repositories of code exist. This is not yet the case with mathematical morphology, although various attempts have been made. It is the personal belief of the authors of this chapter that this has hindered the adoption of many effective algorithms in the larger community of researchers and users of image analysis. Of course, making code freely available does not always sit nicely with intellectual property demands of funding agencies and institutions. This is yet another challenge for which solutions have been proposed, such as dual licensing.

Finally, a last challenge is in the evolution of computer architectures. We are now in the era of generalized multiprocessors and cheaply available massively parallel co-processors and clusters. This means that the tools of image analysis and, in particular, algorithms, must yet again adapt themselves to this changing environment.