# Computationally efficient, one-pass algorithm for morphological filters

Petr Dokládal, Eva Dokladalova

# Computationally Efficient, One-Pass Algorithm for Morphological Filters

Petr Dokládal[a,*], Eva Dokládalová[b]

[a] *Center of Mathematical Morphology, Department Mathematics and Systems, Mines PARISTECH, 35, rue St. Honoré, 77 300 Fontainebleau Cedex, France*
[b] *Unité mixte de recherche CNRS-UMLV-ESIEE, UMR 8049, Université Paris-Est, Cité Descartes B.P.99, 93 162, Noisy le Grand Cedex, France*

**Abstract**

Many useful morphological filters are built as long concatenations of erosions and dilations: openings, closings, size distributions, sequential filters, etc. This paper proposes a new algorithm implementing morphological dilation and erosion of functions. It supports rectangular structuring element, runs in linear time w.r.t. the image size and constant time w.r.t. the structuring element size, and has minimal memory usage.

It has zero algorithm latency and processes data in stream. These properties are inherited by operators composed by concatenation, and allow their efficient implementation. We show how to compute in one pass an Alternate Sequential Filter ($\mathrm{ASF}^n$) regardless the number of stages $n$.

This algorithm opens the way to such time-critical applications where the complexity and memory requirements of serial morphological operators represented a bottleneck limiting their usability.

*Keywords:* Mathematical morphology, Nonlinear filters, Alternate sequential filters, Real-time

## 1. Introduction

Since its introduction in late sixties, the Mathematical morphology provides a complete set of image processing tools from filtering [1, 2], multi scale image analysis [3] to pattern recognition [4, 5, 6]. They have been used in unrivalled number of applications [7, 8]. The most significant examples include biomedical and medical imaging, video surveillance, industrial control, video compression [9], stereology or remote sensing [10].

Nonetheless, not all useful operators can be easily implemented in real time with reasonable memory requirements. In demanding image-interpretation applications requiring a high correct-decision liability, one often uses robust but costly multi-criteria and/or multi-scale analysis.

These applications often consist of a serial concatenation of alternating atomic operators dilation and erosion with progressively increasing computing window called structuring element (SE).

Such operators cannot be parallelized due to the sequential data dependency of the individual atomic operators. The only possibility is to minimize the latency of each atomic operator and consider computing in stream. The latency minimization reduces the time to wait for individual pixel results. The stream computing allows transferring them immediately to the next atomic operator, as soon as they are available and before the entire image is processed. Thus, these atomic operators can work simultaneously, on data delayed in time. In such implementation, one has to sum the individual working memories of every atomic operator. Then also the memory may become penalizing for large, high-resolution images.

The work presented in this paper, aims to propose a new dilation/erosion algorithm with a constant processing time, low latency and low memory requirements for implementation of the individual atomic operators. Consequently, it allows to implement advantageously the following:

1. Alternate Sequential Filters (ASF) - that are a concatenation of openings and closings with a progressively increasing structuring element, useful for multi-scale analysis [1].

2. Size distributions (granulometries) - that are a concatenation of openings allowing to assess the size distribution of a population of objects [3, 11, 12].

3. Statistical learning - a selected set of morphological operators $\xi_i$ can be separately applied to an image $f$. Then for every pixel $f(x, y)$, the vector of values $(\xi_i(f)(x, y))$ can serve as vector of descriptors for pixel-wise learning and classification [6]. Obtaining them may be computationally intensive.

### 1.1. Paper organization

The remainder of the Introduction lists the most known fast algorithms of morphological dilation and erosion and discusses their properties, followed by the explanation of Novelties in this paper.

The Preliminaries, Section 2, introduce the basic principles of dilations, erosions and their combinations.

The Section 3 outlines the Principle of the new Algorithm: i) the 2-D decomposition preserving sequential access to data and zero latency, ii) elimination of useless values, iii) the conversion of an anti-causal structuring element into a causal one, necessary to preserve the sequential access to data, and iv) the encoding used to reduce the memory requirements and acceleration of computations.

*Principal corresponding author

Sections 4 to 5 discuss the properties and section 6 presents the case of the four stage ASF as an example.

The paper concludes by Benchmarks, general Conclusions and Future extensions, sections 7 to 9. The commented pseudo code is given in the Appendix.

### *1.2. Existing work*

The mathematical morphology relies on two fundamental, complementary operations: erosion and dilation. They are local, defined within a computing window, specified by the so-called structuring element (SE), characterized by its size, its shape and origin. It is well known that, with the increasing size of the SE, the direct implementation leads to an extremely high computing cost. Although the fastest existing algorithms [13, 14, 15, 16] concentrate mainly on the reduction of the number of comparisons, few of them deal with the latency and memory requirements [17]. Moreover, the minimization of comparison number is not always proportional to the overall performance improvement [15].

In the following paragraphs, we concentrate on the presentation of the existing state of the art. We discuss it from the classical point of view of complexity, based on the number of comparisons. We bring it face to face with the latency and the memory requirements. For each algorithm, we analyze the possibility of its stream implementation since it is a key feature allowing efficient chaining of the atomic operators.

Lets start by the definition of the used basic terms. Consider a system $Y = f(X)$ with $X$ and $Y$ the input and output data streams. By *latency* understand the distance between the same positions in the two streams. It is a dimensionless value, expressed in number of data samples. It is the sum of several factors:

1) *operator latency* - is induced by non causal operators due to the fact that the value to output depends on future signal samples. Consider a basic max filter $y_j$=max$(x_{j-w/2}, \ldots, x_{j+w/2})$. One cannot output $y_j$ before having read all $x_i$ until $x_{j+w/2}$,

2) *algorithm latency* - some algorithms continue reading the input even after all needed input data are available. Several morphological dilation/erosion algorithms run in two (forward and backward) data scans, e.g. [13, 18]. Typically, in [18], before processing one image line, one needs to read the entire line. For 2-D dilation by a rectangle, implemented separately in the horizontal and vertical direction, one would need to wait the bottom of the image before writing the result. In [13] these forward and backward scans can be done on $w$ pixels long intervals.

For example, the naive implementation of the morphological dilation (Eq. 3) has a considerable computation complexity $\mathcal{O}(w\text{-}1)$ per pixel, with $w$ the SE width (or area in 2-D), but no algorithm latency.

The operator latency - inherent to the operator - is incompressible. Consequently, the optimization effort should focus on the algorithm latency and the computational complexity.

The first concern related to the latency is therefore the time response of the system. Another concern related to the latency is the memory requirements. This can intuitively be explained by the fact that the latent (meaning "hidden") data need to be temporarily stored somewhere to not to get lost. Obviously, a large latency requires large storage. An interesting conclusion is that using larger SE will have larger memory requirements.

#### *1.2.1. State of the art*

The scientific community has adopted several approaches to speed up the erosion/dilation computation. The first one, we call *direct computation*, consists of a straightforward optimization of the computation given the SE shape.

The second approach relies on the *SE decomposition* into a sequence of reduced SE. Consequently, the optimization effort concentrates on the computation of this smaller SE. The special attention is paid to the SE decomposition into a series of *1-D* SE, very popular in numerous applications [19, 20]. It allows better data access, reuse of intermediate results and is easy to parallelize.

In the following, refer to Tab. 1 and 2 summarizing the properties of some algorithms cited below. By data memory understand the temporary storage for input or output data if the algorithm uses random data access. For instance the direct implementation needs random access to input data, whereas the output is written sequentially. It includes also the image transposition used by some algorithms. The working memory is any supplementary memory space required by the algorithm. It includes the data structures like FIFOs, LUTs, histograms, etc. Temporary constants, scalar variables, counters, etc, are omitted.

*Direct 2-D computation.* Optimized algorithms reduce the computing redundancy by using some well-suited data structures to keep the intermediate results. The most natural way is the approach used by Huang *et al.* [21] for median filtering, by Chaudhuri *et al.* [22] for rank-order filtering, and later by Van Droogenbroeck and Talbot [23]. They use a histogram to store the values within the span of the SE at some position in the image. During the translation of the SE over later image positions the histogram is updated by inclusion/deletion of the values of the entering/leaving points. The family of available shapes for the SE is arbitrary. On the other hand, using histogram makes that the input data have to be integers.

*SE decomposition.* It has soon become evident that the SE decomposition offers another possibility to obtain a fast implementation of more complex SEs both on specialized hardware as well as on sequential computers, and the literature soon became abundant see e.g. [24, 25, 26, 27, 28, 29, 30, 31]. The speedup is obtained by dividing the effort in two independent key aspects, an efficient decomposition and the algorithm used for computing the atomic operations.

Various types of decompositions have been proposed. Perhaps the most known decomposition of linear sets is the *linear decomposition* which comes from the associativity of the dilation, see Matheron [28]. Pecht [29] has proposed a more efficient logarithmic decomposition based on the *extreme set* of some SE. For example, for a polygon, the extreme set contains the vertices.

Table 1: 2-D algorithms comparison

| Algorithm | SE type | Complexity per pixel | Algorithm Latency | Data Memory | Working Memory |
|---|---|---|---|---|---|
| Naive 2-D | User | $\mathcal{O}(WH)$ | 0 | $MN$ | 0 |
| Urbach-Wilkinson | User | $\mathcal{O}(N_c + \log_2(L_{max}(C)))$ | $MN$ | $MN$ | $NH\log_2 W$ |
| Van Droogen-broeck-Talbot | User | $\mathcal{O}(H \log_2(G))$ * | 0 | $NH$ | $WHG$ |
| This paper 2-D | Rect. | $\mathcal{O}(1)$ | 0 | 0 | $2(NH+W)$ |

$W \times H$ = SE size (Width×Height); $N \times M$ = image size; $G$ = number of gray levels; $L_{max}(C)$ = maximum chord length; $N_c$ = number of chords; * square SE

Table 2: Fast 1-D algorithms comparison.

| Algorithm | SE type | Comparisons per pixel | Algorithm Latency | Overall Memory |
|---|---|---|---|---|
| Naive 1-D | User | $W-1$ | 0 | $N$ |
| van Herk Gil-Wermann | Sym | $3 - \frac{4}{W}$ | $W$ | $N+2W$ |
| Gil-Kimmel | Sym Even/Odd | $1,5 + \frac{\log_2 W}{W} + +\mathcal{O}(\frac{1}{W})$ | $W$ | $N+3W$ |
| Lemire | Left | 3 | 0 | $N+W$ |
| Lemonnier | Sym | nc | $N$ | $2N$ |
| Van Droogen-broeck-Buckley | Sym Even/Odd | nc | 0 | $2N+G$ |
| This paper 1-D | User | $\mathcal{O}(1)$ | 0 | $2W$ |

Sym = symmetric SE; Left = Left sided SE; User = user defined; $W$=SE size; $N$=line size; $G$ = number of gray levels; nc = not communicated.

Van den Boomgaard and Wester [30] show that the Pecht decomposition can be improved for convex shapes. They propose a decomposition of an arbitrary shape into the union of convex shapes taken from a fixed collection of basis, efficiently decomposable shapes. Coltuc and Pitas [31] propose a factorization based algorithm running efficiently for $2^n$ signals. Soille *et al.* [27] propose an extension of the 1-D van Herk algorithm to 2-D. The SE is a line oriented in an arbitrary angle. The decomposition is obtained by saving the 2-D image as an 1-D array, and recomputing the pixel indices correspondingly to the given orientation of the line. Another efficient algorithm has been recently proposed by Urbach and Wilkinson [16]. It decomposes a flat, arbitrary-shape SE by using a set of 1-D chords. The min/max statistics of the chords are stored in LUT.

*1-D algorithms.* The 1-D algorithms compute the partial 1-D dilations after the SE decomposition into lines.

One of the earliest, and most often used 1-D algorithms, is the van Herk algorithm [13] proposed in 1992. The same algorithm completed by theoretical background was also published by Gil and Werman [32], and later improved by Gevorkian *et al.* [33] and Gil and Kimmel [14]. The computational complexity is independent of the SE size. It requires two passes on the input data: causal and anti-causal. Consequently, computing in stream is impossible. Another, similar algorithm was proposed in [34] using ring-type buffers. Recently, Clienti *et al.* [35] propose an interesting modification of the Van Herk algorithm reducing the memory to $2W$, implemented on an FPGA.

A different approach has been used by Lemonnier [18]. It identifies and propagates local extrema as long as it is required by the SE size. Again, two passes are needed: causal and anti-causal. Hence, the algorithm latency is $N$. The stream execution is impossible.

Van Droogenbroeck and Buckley [15] publish an anchor based algorithm for erosions and openings. The anchors are these portions of signal that remain unchanged by the operator. The algorithm gives good performance in terms of the computing time. The erosion can not run *in place* and stream processing is probably impossible. The principal disadvantage is in using histograms (suited only for integer values, and making the algorithm irregular).

Lemire proposes a fast, stream-processing algorithm for a left-sided SE [17], and later for symetric SE [36]. Both versions simultaneously compute 1-D dilation and erosion, run on floating point data and have low memory requirements and zero latency. However, even though the algorithms are supposed to run in stream, the intermediate storage of coordinates of local extrema actually represents a random access to the input data.

### *1.3. Latency issues*

In order to asses the latency of 2-D algorithms we need to consider separately these different algorithm categories:

• For decompositions of rectangles using $R = H \oplus V$ (R=rectangle, H/V=horizontal/vertical segment respectively) the latency in 2-D is a multiplicative factor of the latency in 1-D and the image width. For two pass algorithms, e.g. Lemonnier [18], the 2-D latency equals one image frame. For locally two-pass algorithms, [13, 32, 33, 27] a specific decomposition would have to be found to optimize the latency in 2-D. The same holds also for other 1-D algorithms that in 1-D allow streaming processing [17, 36, 15].

• Regarding the direct computation in 2-D, though UW [16] could theoretically read/write the input/output images sequentially, the possibility of streaming processing is not mentioned; they also use random accesses to intermediate data. Van Droogenbroeck-Talbot [23] and the naive implementation write output sequentially, but use random accesses to input data.

• The computational complexity, used in the Tab. 1, may introduce to the result a supplementary delay - the *time to compute the result*. Whereas the two latencies are relative to the stream rate, the additional delay depends on the implementation and the computation platform. It can be i) negligeable as in most R=H⊕V decompositions, where the latency prevails, or 2) dominant like in the direct implementation with large SE or in 3-D.

### *1.4. Novelty of this paper*

Although one can find several 1-D algorithms running with zero algorithm latency, none of the above cited algorithms combines all the features necessary for efficient implementation of composed operators in the form $\xi = \delta_{B_n}\varepsilon_{B_{n-1}} \ldots \delta_{B_2}\varepsilon_{B_1}$ for 2-D images.

Suppose the atomic operators $\delta, \varepsilon$ implemented using an algorithm with sequential access to data. This allows to run in parallel the entire $\xi$ despite its internal sequential data dependence. If the atomic algorithm, in addition, has zero algorithm latency, then the entire chain $\xi$ inherits the same properties: sequential data access and zero algorithm latency. This is an interesting property, since computing $\xi$ suddenly becomes very efficient: in stream, with only the (further irreducible) operator latency of $\xi$. See the application example Fig. 4.

In this scope, the novelty of this paper is multiple. It is the only algorithm that combines all necessary features for efficient, parallel implementation of serial morphological operators. It uses a strictly *sequential* access to data, and can also run *in place*. The output is produced with *zero algorithm latency*. The algorithm runs in *linear time* w.r.t. the image size and *constant time* w.r.t. the SE size.

Its additional features include: very low *memory requirements*. A natural support of *floating point* data (not all previous algorithms can support floating point data). *The origin* can be arbitrarily placed within the structuring element, which is useful for even sized SE or specific SE decompositions.

## 2. Preliminaries

### *2.1. Morphological Dilation and Erosion*

Let $\delta$, $\varepsilon$: $\mathcal{L} \to \mathcal{L}$ be a dilation and an erosion, performed on functions $f \in \mathcal{L}$, defined as $f : D \to V$. Below assume $D = \text{supp}(f) = \mathbb{Z}^n$, $n = 1, 2, \ldots$ and $V = \mathbb{Z}$ or $\mathbb{R}$. $\delta_B$, $\varepsilon_B$ are parameterized by a structuring element $B$, assumed rectangular and flat i.e. $B \subset D$ and translation-invariant.

Functional (operating on functions) erosion and dilation by a flat SE defined by extension to functions of the Minkowski set addition/subtraction definitions are given by

$$[\delta_B(f)](x) = [\bigvee_{b \in B} f_b](x) \tag{1}$$

$$[\varepsilon_B(f)](x) = [\bigwedge_{b \in \widehat{B}} f_b](x) \tag{2}$$

where $\widehat{\ }$ denotes the transposition of the structuring element, equal to a set reflection $\widehat{B} = \{x \mid -x \in B\}$, and $f_b$ denotes the translation of the function $f$ by some vector $b \in D$. Hence, the definitions Eqs. (1, 2) can be implemented by

$$[\delta_B(f)](x) = \max_{b \in B} f(x - b) \tag{3}$$

$$[\varepsilon_B(f)](x) = \min_{b \in B} f(x + b) \tag{4}$$

Dilations and erosions combine to form other operators. We shall focus on combinations obtained by concatenations that this algorithm implements optimally.

The basic products obtained by concatenation[1] are opening $\gamma_B = \delta_B\varepsilon_B$ and closing $\varphi_B = \varepsilon_B\delta_B$. Hence from, one forms the so called Alternating Filters obtained as $\gamma\varphi$, $\varphi\gamma$, $\gamma\varphi\gamma$ and $\varphi\gamma\varphi$. The number of combinations obtained from two filters is rather limited. Other filters can be obtained by combining two *families* of filters. This leads to morphological Alternate Sequential Filters (ASF), originally proposed by [37], and studied in [1] Chap. 10. In general, it is a family of operators parameterized by some $\lambda \in \mathbb{Z}^+$, obtained by alternating concatenation of two families of increasing, resp. decreasing filters $\{\xi_i\}$ and $\{\psi_i\}$, such that $\psi_n \leq \ldots \leq \psi_1 \leq \xi_1 \leq \ldots \leq \xi_n$.

The most known ASF are those based on openings and closings, obtained by taking $\psi = \gamma$ and $\xi = \varphi$ :

$$ASF^\lambda = \gamma^\lambda\varphi^\lambda \ldots \gamma^1\varphi^1 \tag{5}$$

starting with a closing, and

$$ASF^\lambda = \varphi^\lambda\gamma^\lambda \ldots \varphi^1\gamma^1 \tag{6}$$

starting with an opening.

This brief survey of theory allows to intuitively appreciate the complexity and the challenge involved by the usage of such long compound operators. If the algorithm does not deal at the same time with the memory management as well as with the latency, the overall performances could be (and generally they are) significantly lowered. We address this problem in Section 6 where we show how to efficiently implement the concatenation of dilations and erosions.

[1] to be read from right to left

## 3. Principle of the Algorithm

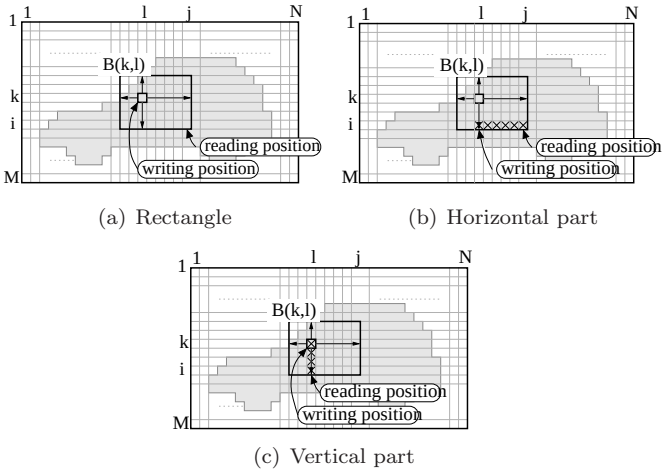
According to the algorithm classification presented in the Introduction, the proposed algorithm belongs to the *SE decomposition* approach using an improved *1-D dilation algorithm*.

### *3.1. Separation of 2-D into 1-D*

Recall that separable operators are run in all directions separately. This requires intermediate data storage between individual runs. If the 1-D parts use sequential data access, it allows to compose n-D dilations also using sequential data access. This eliminates the necessity of intermediate data storage.

The input image is read in the raster scan order, line by line. Every line is dilated horizontally. The result of the horizontal dilation is immediately read, pixel by pixel, by the vertical dilation in the corresponding column. The result of the vertical dilation part is written to the output. The output image is also written in the raster scan order.

Fig. 1a illustrates the computation of the result at position $(k, l)$. Assume that the input data have already been read until line $i$, column $j$. The ongoing computations (depicted by $\times$) are: the horizontal dilation part (Fig. 1b) is running on line $i$, with the reading position $(i, j)$ in the input image, writing position $(i, l)$, immediately read by the vertical dilation (Fig. 1c) with reading position $(i, l)$ and writing position $(k, l)$, directly written to the output. The lines 1 to $i$–1 have already been horizontally dilated, and all columns have already been vertically dilated up to the line k.




(a) Rectangle (b) Horizontal part

(c) Vertical part

Figure 1: Separation of computing: (a) rectangular element into (b) the horizontal and (c) vertical part. $\times\times$ denote the data stored in the queue of the corresponding line or column.

### *3.2. 1-D algorithm*

#### *3.2.1. Elimination of useless values*

An efficient coding of the function profile can avoid a number of comparisons during the computation of a dilation or an erosion. One can drop all values that will never take over in the result of the max or min, Eqs. (3, 4).

Consider a 1-D, connected structuring element $B$ containing its origin. Then, computing $\delta_B f(x)$ needs only those values of $f(x_i)$ that can be seen from $x$ when looking over the topographic profile of $f$. The valleys shadowed by mountains contain unneeded values, see Fig. 2. Notice that the masked values depend on $f$, and not on $B$.

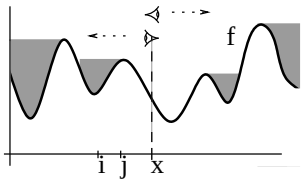



Figure 2: Computing the dilation $\delta_B f(x)$: Values in valleys shadowed by mountains when looking from $x$ over the topographic relief of $f$ are useless.

Now, lets place ourselves in the context of streaming algorithms. For simplicity assume a *causal* SE, i.e. containing its origin at the right hand side. For causal SE, one only needs to look leftwards, over the past samples. The search of the useless values can be formalized by the Prop. 1 showing that values useless at some time instant $x$ remain useless also for the "future".

**Proposition 1.** [*Useless values*] *In computing the dilation $\delta_B f$, with $f : \mathbb{Z}^+ \to \mathbb{R}$, by some causal, connected structuring element $B$ (a linear segment) containing its origin, no $f(i)$ such that $f(i) \leq f(j)$, and $i < j$, will influence the dilation*

$$\delta_B f(x), \text{ for } \forall x \geq j \tag{7}$$

**Proof.** *From Eq. 3, any $x$ such that $i < j \leq x$, if $i \in B(x)$ then $j \in B(x)$. If $f(i) < f(j)$, then $f(i) < \max_{b \in B} f(x - b)$, and $f(i)$ has no impact on the dilation result.*

*This means that all $f(i)$ such that*

$$f(i) \leq f(j), \text{ with } i < j, \tag{8}$$

*may be dropped from the computations.*

This is a strong proposition that allows a considerable reduction of the computational redundancy. One comparison $f(i) \leq f(j)$, done upon reading $f(j)$, avoids computing $j-i$ useless comparisons for any later $B(x)$ that covers $i$ and $j$.

Two important points are to be noticed.

1. $\forall x \geq j$ in Eq. 7 means that all values that become useless at the position $j$ remain useless in the future, $\forall x \geq j$.

2. Using a bounded and causal $B \subset \mathbb{Z}$, i.e. an interval $B(x) = [x - b, x]$, with $b < \infty$, means that for computing $\delta_B(x)$, one can also discard all values outside the SE span, i.e. $f(x_i)$, with $x_i < x - b$.

**Remark**. This proposition does not hold for non causal SE. The values useless at time $x$ may become useful for some $k > x$. This algorithm utilizes the commutation of dilation with translation to convert an anti-causal SE to a causal one.

*3.2.2. Anti-causal to causal SE conversion*

Every SE $B$, $B \subset D$ is equipped by an origin $x \in D$. Assuming a sequential access to the input data, the dilation $\delta_B f(x)$ depends of points read before but also after $x$. We say that $B$ is non causal.

One can transform a non causal SE to a causal SE by utilizing the property that dilation commutes with translation ($t \in D$)

$$\delta_{B+t} f(x) = \delta_B f(x - t) \tag{9}$$

The translation consists of writing the result at the correct place in the output. The horizontal and the vertical shifts are handled by the 1-D horizontal or vertical dilation part, implemented by the Fnct. 1 page 9.

*3.2.3. Function coding*

Similarly as binary objects can be coded by using the distance to their boundaries, functions need to be coded by computing the distance to every change of the value. Using the Prop. 1 and relative indexing, the samples $f(x)$ used in computation $\delta_B f(x_i)$ are coded by pairs (*distance, value*) as given in Fig. 3.

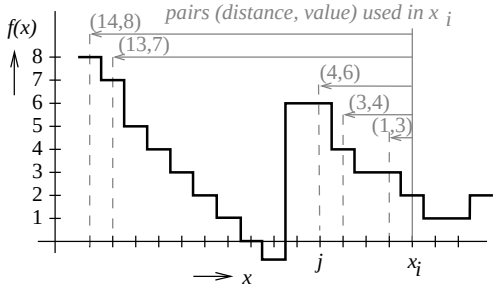



Figure 3: Function coding. The useless values are discarded.

The arrows indicate those values that enter in the computation of $\delta_B f(x_i)$. The values non indicated by an arrow are smaller or equal to $f(j) = 6$ and have no impact on the result $\delta_B f(x)$, for $x \geq j$.

The Eq. 8 is used by the 1-D dilation algorithm to exclude from the computation all useless values.

## 4. Algorithm Complexity and Latency

In this section we shall analyze the latency and the complexity of the algorithm.

*4.1. Latency*

The overall algorithm latency is function of two factors: i) the latency of the 1-D dilation (Fnct. 1), and ii) the latency of the 2-D decomposition (Algorithm 2).

• 1-D dilation : The Function 1 writes the output as soon as the reading position *rp* reaches the last position covered by the structuring element (code lines 6 to 7). This corresponds to the last output-to-input data dependency position, i.e. the operator latency.

Remark: The *while* loop, clearing from the FIFO the useless values, operates on past signal samples. Consequently, it doesn't enter in the latency count. The latency of the Function 1 is therefore strictly equal to the operator latency.

• 2-D dilation : The 2-D dilation is decomposed in the way that result of the horizontal 1-D dilation is directly fed to the corresponding 1-D vertical dilations. Their results are recombined into the output stream. Therefore, the algorithm latency of the 2-D decomposition is zero.

*4.2. Computation complexity*

The 2D_Dilation algorithm iterates over all coordinates of the output image. The inner complexity of the 2D algorithm is $\mathcal{O}(N)$, where $N$ is the number of pixels in the image. At every coordinate, the 2D_Dilation calls twice the 1D_Dilation function: once for the vertical dilation and once for the horizontal dilation part.

The 1D_Dilation part (Fnct. 1) contains a sequence of $\mathcal{O}(1)$ operations, and one *while* loop (lines 1-2), clearing useless values from the FIFO. We shall see that this *while* loop is executed at most once per pixel, making the complexity of the 1D_Dilation algorithm constant per pixel.

Every incoming pixel is stored in the FIFO once and only once (line 5). Every pixel is cleared from the FIFO once and only once, either i) when it becomes "too old", i.e. uncovered by the current SE span (lines 3-4) or ii) when it gets masked by another, higher value (lines 1-2).

The deletion (lines 1-2) of every pixel can be delayed. Delete pixels from FIFO later or sooner has no impact on the algorithm complexity. However, it occurs that several pixels are deleted at the same time. This implies using the loop *while* (lines 1-2). The loop iterates at most once per pixel. Other pixels that become "too old" are deleted at lines 3-4. Hence, both ways of deletion have the same complexity $\mathcal{O}$ (1) per pixel.

This allows to make the following conclusions:

1) The FIFO size is upper-bounded by the SE width. This determines the memory requirements (detailed in the next section).

2) For every pixel, the number of the iterations of the *while* loop is lower-bounded by zero and upper-bounded by the SE width.

3) The average number of iterations of the *while* remains in $[0, 1]$ per pixel.

4) The worst-case complexity of the 1D_Dilation per pixel is bounded by $\mathcal{O}(W)$.

Hence, we shall conclude that the overall complexity of the 2-D dilation algorithm is $\mathcal{O}(N)$, i.e. linear w.r.t the size of the image (N pixels) and constant w.r.t the SE size.

Note: Although both ways of deletion have the same complexity $\mathcal{O}(1)$, they do not have the same cost (in terms of instruction count). The deletion by shadowing is slower in C because of the overhead of the loop *while*. The different timings obtained on various data (constant, random or natural images) are due to this overhead. For nonincreasing intervals (e.g. a constant image - see Benchmarks) the loop (lines 3-4) never executes. The probability of either deletion being data dependent explains the slight variation of the execution time on the image content.

## 5. Memory Requirements

In 1-D, the FIFO size is upper-bounded by the width of the SE, which is the memory-worst case encountered

whereever there are no useless data to eliminate. This occurs at all monotonically decreasing intervals of the signal that are longer than the SE width. This is equivalent to results obtained in [15], where for computing $\varepsilon_B f(x)$, only the values $f(x_i)$, with $x_i = B(x)$, are needed. Similar results hold for the dilation.

In 2-D, the memory requirements are given by the decomposition of the rectangle as $R = H \oplus V$ (R=rectangle, H/V=horizontal/vertical segment respectively). The raster-scan data access makes that the computations window (the SE) slides over the image from left to right and from top downwards.

The vertical part of the 2-D dilation runs at all columns simultaneously, one pixel per each column at a time. All columns have the memory-worst case equivalent to the height of the SE.

Consider an erosion (or a dilation) of an $N \times M$ image (width by height) by a $W \times H$ rectangular (width by height) SE. The memory requirements $M$ are

$$M = 2(NH + W)$$

This means, $N$ memory blocks of size $2H$ (vertical part) and one memory block of size $2W$ (horizontal part). The multiplicative factor 2 comes from the fact that the stored data are indexed by their coordinates (see Fig. 3, and Fnct. 1, line 5).

For example, an erosion of an 800×600 image by a 20×20 square will require 2×(800×20+20)=32,040 bytes. Compared to this, storing an 800×600 image is costly, requiring 480,000 bytes (with 1 byte/pixel coding). Neither the input nor the output image need to be stored in memory.

The memory requirements graphically correspond to storing the image data from the lines currently intersected by the SE.

## 6. Application

In the following we give as example the implementation of an $ASF^\lambda$ given by Eq. (5). Rewrite the filter as a concatenation of erosions and dilations

$$ASF^\lambda = \delta_{B_\lambda}\varepsilon_{B_\lambda}\varepsilon_{B_\lambda}\delta_{B_\lambda} \ldots \delta_{B_1}\varepsilon_{B_1}\varepsilon_{B_1}\delta_{B_1} \qquad (10)$$

and reduce it into its canonical form

$$ASF^\lambda = \delta_{B_\lambda}\varepsilon_{B_\lambda \oplus B_\lambda}\delta_{B_\lambda} \ldots \delta_{B_1}\varepsilon_{B_1 \oplus B_1}\delta_{B_1}$$

This $ASF^\lambda$ can be implemented in a stream in one raster scan of the input image. The writing position of the preceding operator in the cascade becomes the reading position of the following operator. The operator latency of the entire $ASF$ will be given by the one introduce by the result of Minkowski sum of all SE in Eq. 10, that is $\bigoplus_{i=1}^{n} B_i$.

Figure 4 illustrates the propagation of real image data through an $ASF^4$ after having read approximately one third of the input image. The SE is a square of size $s$+1 for the $s$-th stage. The individual operators, with sequential data dependence, are running simultaneously. There is no intermediate data storage between the stages; the intermediate results are pipelined.

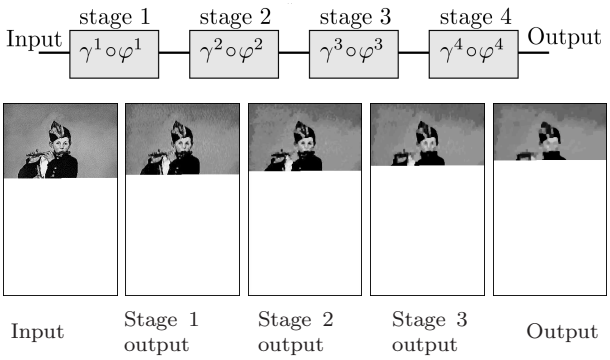



Figure 4: Propagation of data throughout $ASF^4$ after having read approximately one third of input image (Manet's painting "Le fifre")

Table 3: Execution time in ms for 2-D dilation of the 'mountain.pgm' image (800×600) by a 21×21 square for various data coding types.

| Data type / CPU type | int | float | unsigned char | double |
|---|---|---|---|---|
| Intel Core2 Duo 2.4GHz (32 bits) | 17.49 | 18.05 | 25.77 | 20.32 |
| Dual Core AMD Opteron 2.4GHz (64 bits) | 22.34 | 23.64 | 25.02 | 22.06 |

## 7. Benchmarks

This section illustrates the execution time of this algorithm, w.r.t. various criteria, measured on an Intel Core 2 2GHz CPU, with 2GB 800MHz Dual Port RAM, running Linux. The time reported below is the processor time spent in the dilation/erosion algorithm as reported by a profiler (obtained as a mean after several runs to reduce inaccuracy).

The first experiment, see Fig. 5, illustrates the running time w.r.t. the content of the image. We have used a constant and a white-noise image to illustrate the fastest and the worst-case running time, and a natural image to illustrate the "expected" running time on a natural scene. The measured time follows a linear function of the image size. Note also that the performance on the natural image actually coincides with the worst case performance obtained on the white noise. (The various sizes of the natural image were obtained by tiling side to side the original photography mountain.pgm from [38].)

We have evaluated the performance of the algorithm against different data types, see Tab. 3. The best performance has been obtained for the word width corresponding to the used CPU architecture, i.e. the integer and float for a 32 bit CPU, and the double for a 64 bit CPU. The penalty obtained for the unsigned char (8 bits) is due to inefficient memory access on both architectures.

We have compared our algorithm with Urbach-Wilkinson[2] [16] and Van Droogenbroeck-Buckley[3] [15], see Fig. 6.

The best timing was obtained with [15], the worst with [16], which allows - on the other hand - SE of ar-

[2]code courtesy of Erik Urbach
[3]code available at [38]

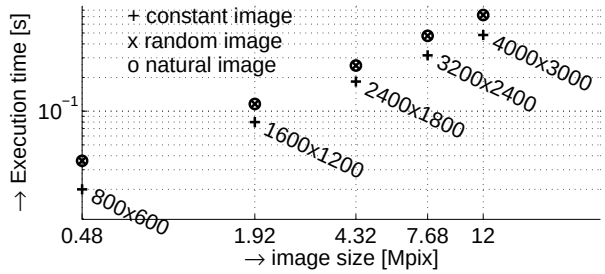



Figure 5: Execution time of erosion, with respect to the content of the image. Data read from memory.

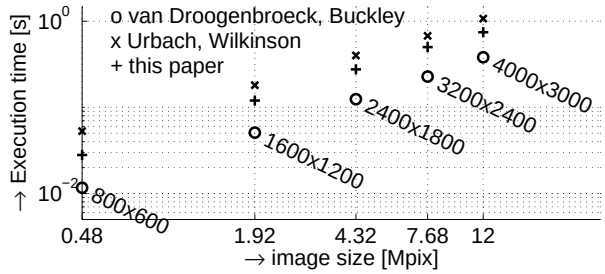



Figure 6: Execution time of erosion, with respect to the size of image. Structuring element 21×21. Data read from memory.

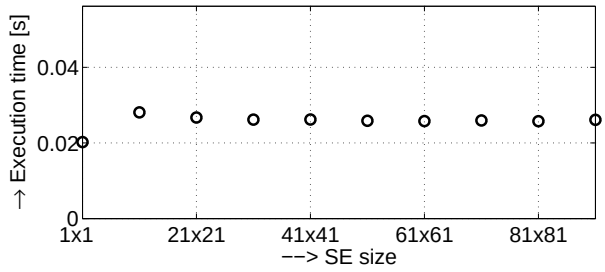



Figure 7: Execution time of erosion with respect to the size of structuring element. Image size 800×600.

bitrary shapes. The algorithm from this paper remains competitive with the speed of the other algorithms, even though the speed has been traded off for memory consumption and latency.

Finally, the last experiment, see Fig. 7, illustrates that the complexity of the algorithm is independent of the SE size. The SE is a centered, $10k{+}1 \times 10k{+}1$ square, for $k = 0 \ldots 9$.

Note: The FIFO queues are implemented in C by using pointer-addressed arrays.

## 8. Conclusions

This paper proposes a new algorithm for functional dilation or erosion by a flat, rectangular structuring element for 2-D data (easily extensible to n-D images, and SE in form of n-D boxes).

The algorithm has zero algorithm latency and strictly sequential access to data. The combination of these two properties allows their inheritance to operators composed by concatenation. The entire concatenation chain, despite its internal sequential data dependence, can run simultaneously. The algorithm runs in linear time w.r.t. the image size and constant time w.r.t. the SE size.

Regarding serial filters, if all the operators in the concatenation run simultaneously - then result can also be obtained in constant time w.r.t. the length of the concatenation.

The algorithm has low memory requirements, which eases its implementation on systems with space constraints, such as embedded or mobile devices, intelligent cameras, etc. The linear time and zero latency allow efficient implementations on demanding industrial systems with severe time constraints.

Even though the speed is not the principal concern here, the algorithm remains competitive in term of execution time compared to the recent proposed fast implementations.

## 9. Future extensions

### *9.1. Other SE shapes*

The algorithm, described above with rectangles, can also be extended to other shapes decomposable into linear segments (e.g. polygons as in [27]).

### *9.2. Spatially variant SE*

This paper is the third step of a wider work towards an efficient implementation of morphological operations with spatially variant structuring elements that are useful for adaptive filters. The first step has been the stream implementation of dilation/erosion of sets [39]. It has the same algorithmic properties: zero latency and optimal memory, sequential access to data. The second step was the extension to the functional morphology; preliminary results have already been published: 1-D spatially variant morphology [40] and approximations of 2-D spatially variant rectangles [41].

The present algorithm is a simplified version of [41] limited from spatially variant to translation invariant SE. This simplification has brought a 10× speed increase.

The goal is to obtain an algorithm for spatially-variant functional dilations and erosions with structuring elements of unconstrained shapes.

### *9.3. Real-time HW accelerator*

This algorithm can be easily implemented as a finite state machine, interesting for HW implementation. The sequential access allows to read the image from a camera, process it, and write it out for further processing or visualization. This allows processing large, or *infinite* industrial images without storing them in the memory.

## Acknowledgements

The authors wish to thank the anonymous reviewers for their comments and remarks that allowed to improve the paper.

## Appendix

### Notation convention and preliminaries

$\leftarrow$ shall denote the assignment and $=, <, \leq, ...$ tests. Curly braces denote a collection of values, e.g. $A \leftarrow \{5, 8\}$. To obtain an element in a collection by its index we use brackets, e.g. $A[1] = 5$. The empty set is denoted by $\{\}$. In a function call, parentheses denote the collection of arguments; funct() is a function call without arguments.

**FIFO**: The algorithm uses FIFO (First In First Out) queues. The FIFO supports the following operations: *push* - insert a new element, *pop* - retrieve the oldest element, and *dequeue* - retrieve the most recent element, and queries: *front* read the oldest element, *back* - read the newest element. The operations modify the content of the FIFO whereas the queries do not. `fifo`$\leftarrow\{\}$ initializes the fifo to empty.

In this algorithms, the elements inserted/read into/from the fifo are always couples {*value*, *position*}. Hence, e.g. the query `fifo.front()[1]` yields the *value* of the oldest element in the queue.

**Input/output** images are assumed 2D, read and written in the raster scan order one pixel at a time by x $\leftarrow$ in_stream.read() and out_stream.write($x$). The reading/writing position is always implicitly incremented by 1.

### Algorithm Description

This section details the algorithm principles in link to the algorithm pseudo-code, see page 9.

*9.4. 2-D Dilation Algorithm*

The 2-D Dilation, Algorithm 1, is to decompose the 2-D SE into columns and to assemble the partial 1-D computations into a 2-D stream, cf. Fig. 1.

The horizontal and vertical dilation parts are computed by the same function 1D_DILATION. It encodes the input data from the current line or column and stores them in the FIFO. There is one FIFO for the horizontal part dilation - `h_fifo`. For the vertical part, there is an $1 \ldots N$ array - `v_fifo` - one FIFO per image column.

The input image is read at lines 10 to 12. Missing data (to the right of the image) are completed by the padding constant, line 14. The horizontal dilation is computed at line 16.

If the horizontal dilation part outputs a valid (non empty) value, line 19, it is sent to the vertical dilation part, computed by the same function, line 20. The vertical dilations also may require padding - typically below the image - where the horizontal dilation is not called. Instead, dFx is directly set to the padding value, line 18.

Provided the vertical dilation outputs a valid result, line 21, it is directly written to the output image, line 22.

*9.5. 1-D Dilation Function*

Assume computing $dF{=}\delta_B F$, where $F$, $dF : [1,\ldots,N] \rightarrow \mathbb{R}$. The structuring element $B$ is a linear segment, `SE1 + SE2 + 1` pixels long, with `SE1`, `SE2` the offsets of the origin from the left- or the right-most end.

**Calling conventions:** The function `1D_Dilation`, see Function 1, page 9, is a function computing one sample of `dF`, to be written at writing position `wp`.

Upon every call, `rp` must be incremented by one by the calling function. Similarly, every time that `1D_Dilation` outputs a valid sample, `wp` is to be incremented by one.

The current fifo queue needs to be passed by reference.

**Principle**: The function proceeds in three steps:

1. *Dequeue all smaller or equal values*, lines 1 to 2. Removes from the FIFO all values that become useless.
2. *Delete too old value*, lines 3 to 4, removes from the FIFO the value that gets uncovered by the current SE $B(wp)$.
3. *Enqueue the current sample* in the FIFO in the form of a couple {*value, position*}.
4. Provided enough data have been read, line 6, return the dilation result `dF`, line 7. At any moment, this value is found at the oldest position of the FIFO.

**Note**: To obtain erosion instead of dilation:
1) Fnct. 1, line 1, replace $\leq$ by $\geq$.
2) Alg. 2, line 1, set the padding constant PAD to $\infty$.

### Algorithm pseudo-code

**Function 1**: `dF` $\leftarrow$ `1D_DILATION`(*rp, wp, F, SE1, SE2, N, fifo*)

**Input**: rp, wp - reading/writing position; F - input signal value (read at rp); SE1, SE2 - SE size towards left and right; N - length of the signal; fifo - the FIFO
**Result**: dF - output signal value (to be written at wp)

```
   // Dequeue all queued smaller or equal values
1  while fifo.back()[1] ≤ F do
2  |  fifo.dequeue()

   // Delete too old value
3  if (wp - fifo.front()[2] > SE1) then
4  |  fifo.pop()

   // Enqueue the current sample
5  fifo.push({F, rp})

6  if rp = min (N, wp + SE2) then
7  |  return ( fifo.front()[1] ) ;      // return a valid value
8  else
9  |  return ({}) ;                    // return empty
```

## References


[1] J. Serra. *Image Analysis and Mathematical Morphology*, volume 2. Academic Press, NY, 1988.

[2] E. Dougherty. *Mathematical morphology in image processing*. Taylor and Francis, Inc., 1992.

[3] P. Maragos. Pattern spectrum and multiscale shape representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(7):701–716, 1989.

[4] J. Serra. Morphological filtering: an overview. *Signal Processing*, 38(1):3–11, 1994.

[5] S. Mukhopadhyay and B. Chanda. An edge preserving noise smoothing technique using multiscale morphology. *Signal Processing*, 82(4):527–544, 2002.

[6] A. Cord, D. Jeulin, and F. Bach. Segmentation of random textures by morphological and linear operators. In *Proc. International Symposium on Mathematical Morphology ISMM*, pages 387–398, 2007.

[7] R. Haralick, S. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(4):532–550, 1987.

[8] M. Wilkinson and J. Roerdink, editors. *Mathematical Morphology and Its Application to Signal and Image Processing*. Proc. 9th International Symposium on Mathematical Morphology. Springer, 2009.

[9] P. Salembier, P. Brigger, J. Casas Montse Pardas, and J. Casas. Morphological operators for image and video compression. *IEEE Trans. on Image Processing*, 5:881–897, 1996.

[10] P. Soille and M. Pesaresi. Advances in mathematical morphology applied to geoscience and remote sensing. *IEEE Trans. on Geoscience and Remote Sensing*, 40(9):2042 – 2055, 2002.

[11] R. Sabourin, G. Genest, and F. Prêteux. Off-line signature verification by local granulometric size distributions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(9):976–988, 1997.

[12] L. Vincent. Granulometries and opening trees. *Fundamenta Informaticae*, 41(1-2):57–90, January 2000.

[13] M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recog. Letters*, 13(7):517–521, 1992.

[14] J. Gil and R. Kimmel. Efficient dilation, erosion, opening, and closing algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(12):1606–1617, 2002.

[15] M. Van Droogenbroeck and M. Buckley. Morphological erosions and openings: Fast algorithms based on anchors. *J. Math. Imaging Vis.*, 22(2-3):121–142, 2005.

[16] E. Urbach and M. Wilkinson. Efficient 2-D Grayscale Morphological Transformations With Arbitrary Flat Structuring Elements. *IEEE Trans. Image Processing*, 17(1):1–8, 2008.

[17] D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *Nordic J. of Computing*, 13(4):328–339, 2006.

[18] F. Lemonnier and J.-C. Klein. Fast dilation by large 1D structuring elements. In *IEEE International Workshop on Nonlinear Signal and Image Processing, Halkidiki, Greece*, 1995.

[19] L. Najman. Skew detection. European Patent, 2002. Filled at 27, 2002 as a European filing the French Patent Office.

[20] B. Obara. Identification of transcrystalline microcracks observed in microscope images of a dolomite structure using image analysis methods based on linear structuring element processing. *Comput. Geosci.*, 33(2):151–158, 2007.

[21] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Trans. Acoustics, Speech and Signal Processing*, 27(1):13–18, February 1979.

[22] B. Chaudhuri. An efficient algorithm for running window pel gray level ranking 2-D images. *Pattern Recogn. Letters*, 11(2):77–80, 1990.

[23] M. Van Droogenbroeck and H. Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recog. Letters*, 17(14):1451–1460, 1996.

[24] X. Zhuang. Decomposition of morphological structuring elements. *J. of Math. Imaging and Vis.*, 4:5–18, 1994.

[25] X. Zhuang and R. Haralick. Morphological structuring element decomposition. *Computer Vision, Graphics, Image Processing*, 35:370–382, 1986.

[26] G. Anelli, A. Broggi, and G. Destri. Decomposition of arbitrarily shaped binary morphological structuring elements using genetic algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(2):217–224, 1998.

[27] P. Soille, E. Breen, and R. Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(5):562–567, 1996.

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

[29] J. Pecht. Speeding up successive minkowski operations. *Pattern Recog. Letters*, 3(2):113–117, 1985.

[30] R. van den Boomgaard and D. Wester. Logarithmic shape decomposition. In C. Arcelli, L. P. Cordella, and G. Sanniti di Baja, editors, *Aspects of Visual Form Processing*, pages 552–561. World Scientific Publishing Co., Singapore, 1994. Capri, Italy.

[31] D. Coltuc and I. Pitas. On fast running max-min filtering. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 44(8):660–663, 1997.

[32] J. Gil and M. Werman. Computing 2-D Min, Median, and Max Filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):504–507, 1993.

[33] D. Gevorkian, J. Astola, and S. Atourian. Improving Gil-Werman Algorithm for Running Min and Max Filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(5):526–529, 1997.

[34] T. Miyatake, M. Ejiri, and H. Matsushima. A fast algorithm for maximum-minimum image filtering. *Systems and Computers in Japan*, 27(13):74–85, 1996.

[35] C. Clienti, M. Bilodeau, and S. Beucher. An efficient hardware arhcitecture without line memories for morphological image processing. In *Advanced Concepts for Intelligent Vision Systems*, 2008.

[36] D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lowerbound. *Pattern Recognition*, 42:2169–2180, 2009.

[37] S. Sternberg. Grayscale morphology. *Comput. Vision Graph. Image Process.*, 35(3):333–355, 1986.

[38] R. Dardenne and M. Van Droogenbroeck. The libmorpho library. Available for download from http://www2.ulg.ac.be/telecom/research/libmorpho.html.

[39] H. Hedberg, P. Dokládal, and V. Öwall. Binary Morphology with Locally Adaptive Structuring Elements: Algorithm and Architecture. *IEEE Transactions on Image Processing*, 18(3), 2009.

[40] P. Dokládal and E. Dokládalova. Grey-Scale 1-D dilations with Spatially-Variant Structuring Elements in Linear Time. In *European Signal Processing Conference*, 2008.

[41] P. Dokládal and E. Dokládalova. Grey-scale Morphology with Spatially-Variant Rectangles in Linear Time. In *Advanced Concepts for Intelligent Vision Systems*, 2008.

**Algorithm 2**: 2D_Dilation

**Input**: in_stream - input image pixels stream
M,N - height, width of the image
SE1, SE2, SE3, SE4 - struct. element size
**Result**: out_stream

```
1  const. PAD ← 0 ;      // Set the Padding Constant
2  vfifo ← array [1..N] of FIFO ; // array of N empty
   FIFOs for the vertical dilation part
3  line_rd ← 1 ;                  // read line counter
4  line_wr ← 1 ;                 // written line counter
   // iterate over all image lines
5  while line_wr ≤ M do
6      hfifo ← FIFO ; // FIFO for the horizontal part
7      col_rd ← 0 ;              // read column counter
8      col_wr ← 1 ;          // written column counter
       // iterate over all columns
9      while col_wr ≤ N do
           // horizontal dilation on the line_wr line
10         if line_rd ≤ M then
11             if col_rd < N then
12                 F ← in_stream.read()
13             else
14                 F ← PAD ;      // Padding constant
15             col_rd ← min(col_rd +1, N)
16             dFx ← 1D_Dilation (col_rd, col_wr, F,
               SE1, SE3, N, hfifo)
17         else
18             dFx ← PAD ;        // Padding constant
           // vertical dilation of the col_wr column
19         if dFx ≠ {} then
20             dFy ← 1D_Dilation (min(line_rd,M),
               line_wr, dFx, SE2, SE4, M, vfifo[col_wr
               ])
21             if dFy ≠ {} then
22                 out_stream.write(dFy)
23             col_wr ← col_wr +1
24     line_rd ← line_rd+1
25     if dFy ≠ {} then
26         line_wr ← line_wr +1
```