Binary Morphology With Spatially Variant Structuring Elements: Algorithm and Architecture

Hedberg, Hugo; Dokladal, Petr; Öwall, Viktor

Published in:
IEEE Transactions on Image Processing

DOI:
10.1109/TIP.2008.2010108

2009

Binary Morphology With Spatially Variant Structuring Elements: Algorithm and Architecture

Hugo Hedberg, Petr Dokladal, and Viktor Öwall, *Member, IEEE*

***Abstract*—Mathematical morphology with spatially variant structuring elements outperforms translation-invariant structuring elements in various applications and has been studied in the literature over the years. However, supporting a variable structuring element shape imposes an overwhelming computational complexity, dramatically increasing with the size of the structuring element. Limiting the supported class of structuring elements to rectangles has allowed for a fast algorithm to be developed, which is efficient in terms of number of operations per pixel, has a low memory requirement, and a low latency. These properties make this algorithm useful in both software and hardware implementations, not only for spatially variant, but also translation-invariant morphology. This paper also presents a dedicated hardware architecture intended to be used as an accelerator in embedded system applications, with corresponding implementation results when targeted for both field programmable gate arrays and application specific integrated circuits.**

***Index Terms*—Hardware implementation, mathematical morphology, spatially variant structuring elements.**

I. Introduction

MATHEMATICAL morphology is a nonlinear image processing framework used to manipulate or analyze the shape of functions or objects, originally developed by Matheron and Serra in the late 1960s [1]. Mathematical morphology is set theory-based methods of image analysis and plays an important role in many digital image processing algorithms and applications, e.g., noise filtering, object extraction, analysis or pattern recognition. The methods, originally developed for binary images, were soon extended and now apply to many different image representations, e.g., grayscale, color or vector images, and more recently to matrices and tensor fields.

Real-time image processing systems have constraints on speed or hardware resources. In addition, in embedded or mobile applications, these systems require low power consumption and low memory requirements. An example of such a system may be found in [2], in which a real-time automated digital surveillance system with tracking capability is presented. The system is intended to be included in a self-contained network camera and the characteristics of the surveillance scene together with camera placement have a direct impact on system performance. By letting a locally adaptive morphological filter process the binary segmentation result, thereby exploring the depth information in the scene, a more accurate tracking may be observed. Therefore, due to the constraints and the performance increase in such applications, the need for efficient hardware (HW) architectures in terms of computational complexity and memory requirement with low power characteristics for this image representation becomes evident.

This paper is organized as follows: The remainder of this section addresses the motivation of using locally adaptive, spatially variant structuring elements (SV SEs), discusses the application context, and puts it into perspective by comparing to previously published work. Section II discusses basic morphological concepts and properties together with SV SEs in general and inferred restrictions. Section III details the algorithm and Section IV proposes a corresponding HW architecture. Section V presents implementation results of the architecture when targeted for both FPGA and ASIC. Section VI concludes the paper.

A. Application Context

Although translation-invariant structuring elements (TI SE) are sufficient in many image processing applications, SV SEs outperform them by their ability to adapt to local features. SE functions are studied and several examples are given by Serra [3, Ch. 2.2 and Ch. 4], and an early evaluation of the performance of SV SEs versus TI SEs can be found in Chen *et al.* [4].

Generally, there are two strategies to control the shape and size of the SE:

1) *image-exogenous information*, usually used to correct or to adapt to an image anamorphism;
2) *content dependent processing*, e.g., contour preserving filters, and image restoration.

1) Image Exogenous Information: Processing images deformed by anamorphism (such as perspective or wide-angle deformations) can be done in two ways: i) apply usual TI operations after a previous distortion correction or ii) as proposed here, use SV operators that adapt to the distortion. An example of an application that will benefit from anamorphism-aware processing is the road-traffic surveillance scene shown in Fig. 1(a), where the images are deformed by the perspective, see, e.g., Beucher [5]. Extracting individual vehicles from the motion mask, Fig. 1(b), may be done by an alternate morphological filter [6], starting with an opening to eliminate noise, followed

Manuscript received December 10, 2007; revised October 06, 2008. Current version published February 11, 2009. This work was supported in part by the Competence Center for Circuit Design at Lund University and in part by the Center of Mathematical Morphology at Mines Paris PARITECH. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Philippe Salembier.

H. Hedberg was with the Department of Electrical and Information Technology, Lund University, SE-22100 Lund, Sweden He is now with Prevas, Stockholm, Sweden (e-mail: hugo.hedberg@prevas.se).

P. Dokladal is with the Center of Mathematical Morphology (CMM), Mines Paris Paritech, 35, 77305 Fontainebleau Cedex, France (e-mail: petr.dokladal@mines-paritech.fr).

V. Öwall is with the Department of Electrical and Information Technology, Lund University, SE-22100 Lund, Sweden (e-mail: viktor.owall@eit.lth.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2008.2010108

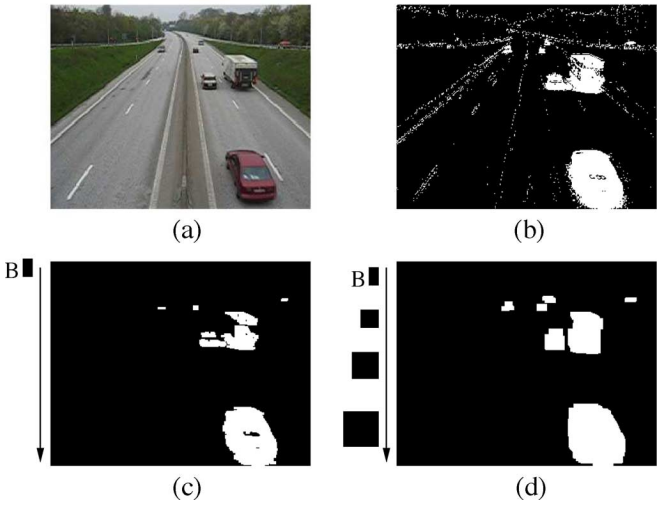

Fig. 1. (a) Typical road surveillance application input image, (b) binary motion mask, (c) and filtered motion mask using a TI SE. (d) Filtered motion mask using a SV SE, increasing in size from top to bottom.

by a closing to close holes and smooth the contours of the vehicle masks. One needs to use a SE sufficiently large to filter the noise, but small enough to preserve the individual vehicles. Due to perspective deformation, a TI SE will not produce a satisfactory result in all regions of the image, Fig. 1(c). However, using a SV SE, increasing in size from top to bottom of the image, eliminates noise and correctly extracts all vehicles, Fig. 1(d). Another application where a SV SE is useful is licence plate detection. Provided the resolution suffices, the SV SE may be used to compensate for the change of the apparent size of the licence plate. Furthermore, wide-angle-camera compensation may be used to correct anamorphism. This is used by Roerdink and Heijmans to measure forest density [7], where a progressively changing SE is used, different in the center and on the periphery of the image, to compensate for the distortion.

For ordinary cameras, the SE size may be set manually, proportional to the apparent size of the objects (as used here). Introducing other imaging techniques, such as range imagery, may give access to distance-to-camera information. Since this information relates to the apparent size of the objects, it allows direct control of the size of the SV SE [8].

2) Content Dependent Processing: In the second use case, the SE size is controlled by the content of the image. Note that there is no alternative approach to restore the image distortion and apply a TI operator as in the case of correction of anamorphism. There are several examples of such content dependent filtering, e.g., the reversible image coding and its restoration from skeleton [9], and also [1] and [10]–[13]. An example of a binary object is shown in Fig. 2(a), which is represented by its skeleton in 2(b). The skeleton is obtained by connecting the centers of maximal balls contained in the object in 2(a). One can associate weights to points on the skeleton which are equal to the Euclidean distance to the complement, i.e., the radius of the balls, and reconstruct the object by dilating the skeleton with the corresponding ball (radius equal to the distance to complement). During the restoration, the skeleton is dilated by large SV SEs, Fig. 2(c).

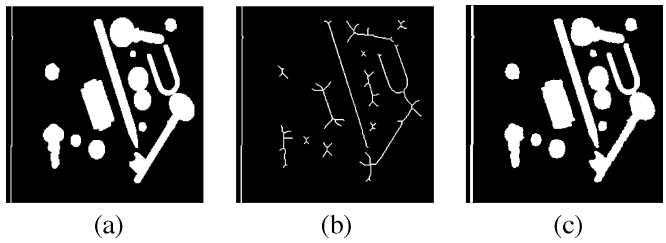

Fig. 2. Object reconstruction from skeleton. (a) Binary object "tools" (b) its skeleton, weighted by distance to the complement, (c) reconstruction from the skeleton.

Filters with adaptable pixel neighborhoods have been thoroughly investigated over the years. Illustrations may be found in the Nagao filter in Natsuyama [14], later in Wu and Maître [15], or more recently in Lerallut [16]. Such SV filters together with their pyramids and derived segmentation aspects are also studied by Debayle and Pinoli in [17] and [18]. They illustrate applications of image denoising, enhancement, filtering, and segmentation with SV SEs, which are compared with results obtained with TI SEs. The idea behind these filters is to define a SV SE that fits inside the objects to prevent blurring when the SE stretches across the boundaries of the objects. At the same time, the SE increases in size towards flat zones to obtain a stronger filtering effect.

SV morphology has also been investigated by Charif-Chefchaouni and Schonfeld [19], and more recently by Bouyanaya *et al.* in [20] and [21] for set-wise SV morphology and SV morphology on functions.

All these references propose application examples or theoretical advances but no efficient implementations. Therefore, the following sections will present an algorithm with a corresponding architecture that is suitable for such applications as discussed above.

B. Previous Optimization Efforts

In *naive* implementations of mathematical morphology operations, outputting one pixel requires the examination of its entire neighborhood defined by the SE at this position. Consequently, using large neighborhoods become particularly computationally intensive and efforts have been made to make these operations efficient.

Although considerable advances have been achieved in conception of fast algorithms and HW accelerators for TI morphology, little has been done for optimization of SV SE. Existing efforts group into different frameworks.

1) Fast Recursive Algorithms for Translation Invariant Structuring Elements: The implementations by Van Herk [22] and by Lemonier and Klein [23] support large linear SEs but need three and two scans, respectively, to complete each operation, requiring intermediate storage. The HW complexity is of $O(1)$ per pixel and memory requirement is of $O(n^2)$, where n is the width of a quadratic image. In addition, their extension to SV SEs in not possible.

Van Droogenbroeck and Talbot [24] propose an algorithm based on histograms. The histogram is updated as the SE slides over the image. The respective value for the needed rank filter (dilation, erosion, or median) is taken from the histogram. This

algorithm naturally extends to SV SEs. However, computing the histogram requires additional resources, and becomes cumbersome for finely quantized data and impossible for $\mathbb{R}$.

2) FFT-Based Algorithms for Translation Invariant Structuring Elements: There are also methods for fast computation of morphological operations with large structuring elements by thresholding convolutions computed as a product of Fourier transforms, see [25]. However, SV structuring elements cannot be written as a product of FFTs. Furthermore, even if there is no increase in computational complexity for large structuring elements, the computational complexity, and memory requirements of FFTs exceed the ones for recursive algorithms (item a), or the one in this work.

3) Efficient Hardware Implementations: Concerning efficient HW implementations, Klein and Peyrard [26] have designed a neighborhood processor for binary mathematical morphology, executing dilations/erosions, thinnings/thickenings, and geodesic operations (reconstructions).

Fejes and Vajda [27] and Velten and Kummert [28] both propose delay-line implementations. This classical approach supports arbitrary shaped SEs, but the computational complexity is of $O(n^2)$, where n is the width of a quadratic SE, and the memory requirements is of $O(n^2)$, where n is the width of a quadratic image. Therefore, this type of implementation becomes unsuitable for large SEs and high resolution applications. This is due to that each element in the SE increases the fan-in of the computations as well as the required amount of memory to delay the rows to extract the pixel neighborhood. In [29], an architecture is proposed based on the observation that many calculations between two adjacent pixels are redundant and can be reused, giving the architecture its name: partial-result-reuse (PRR). By this procedure, the computational complexity can be reduced to $O(2\lceil \log_2(n) \rceil)$, where n is the width of a quadratic SE ($\lceil . \rceil$ is the ceiling function). However, it uses the same type of delay-lines as in [27] and [28], thus resulting in the same memory requirement.

Hedberg *et al.* [30] propose a low-complexity (LC) and low memory requirement architecture. The complexity is reduced to $O(1)$ and memory requirement to $O(n)$, where n is the width of the input image, at the cost that the class of supported SEs is limited to flat rectangles of arbitrary size. Erosions and dilations are accomplished with only two summations and two comparisons independent of the structuring element size and resolution.

4) Spatially Variant Morphology: Recently, Cuisenaire [31] proposes a fast algorithm for binary spatially variant morphology based on thresholding the distance transform, widely used for efficient implementation of dilations and erosions [32], [33]. The class of allowed shapes is restricted to balls of various norms. Various algorithms exist for computing the distance map. They are either i) image scan operations [34], or ii) equidistant propagations from the sources, (see surveys [35] and [36] for overview and other citations). The former have high memory requirements since they use a large intermediate storage for partial results between the scans. The distance is computed on the entire image, penalizing the performance when small SEs are used. The latter, based on equidistant propagation from the sources do not necessarily compute the distance on the entire image and are more efficient. However, they use ordered structures and random memory accesses, penalizing performance on large data sets and are difficult to implement in HW, see Dejnozkova [37] for discussion.

C. Main Contribution

This work fits into the framework of binary Mathematical Morphology and represents the first step towards arbitrary shaped SV SE in efficient HW and SW implementations. The main contribution of this paper is twofold.

1) A new algorithm supporting a rectangular SV SE for binary mathematical morphology with very low computational complexity and memory requirements. An extension to a richer class of structuring elements is possible.
2) A corresponding HW architecture, suitable for embedded or mobile applications. The architecture has several important properties from a HW perspective, i.e., sequential pixel processing, low-computational complexity, and low memory requirement. Implementation results of the proposed architecture are presented in terms of resource utilization when targeted for both FPGA and ASIC.

The architecture proposed in this paper is a development from the one published in [30]. The new architecture allows changing the size of the rectangle within an image from pixel to pixel, and can thereby locally adapt its size. Although having mainly the same memory requirement, the SE flexibility comes at the cost of increased computational complexity from $O(1)$ to $O(n)$, n being the SE width.

II. Algorithmic Issues

Let I be an input image $I: D \rightarrow V$, with $D = \text{supp}\{I\} \subseteq \mathbb{Z}^2$ being the domain and V the set of values. In this paper, we place ourselves in the context of binary images $I : D \rightarrow \{0, 1\}$, where objects are represented by 1, i.e., the object X contained in a binary image I is $X = \{x | I(x) = 1\}$.

All morphological operations are based on logical or arithmetic calculations (for binary or valued images, respectively) on a local neighborhood of a pixel. The neighborhood is a subset of pixels defined by the shape of the *structuring element* $B \subset D$, which has a corresponding $origin \in B$, that determines the position of the calculated value in the output image. The translation of B by some $x \in D$ is often denoted by $B(x)$.

When using a SV SE, the fixed set $B \subset D$ is replaced by a flexible set given by $B : D \rightarrow \mathcal{P}(D)$ with $\mathcal{P}$ denoting the set of subsets. This means that instead of a fixed $B \subset D$ one uses $B : D \rightarrow \mathcal{C}$, where for every point $x \in D$, the mapping $B(x)$ is not a translation but chosen as an element from the class $\mathcal{C}$ of allowed shapes, used locally at x.

Spatially variant binary erosion and dilation are defined by means of Minkowski addition and subtraction (see Serra [3, pp. 41 and 42]) according to

$$\varepsilon_B X = \bigcap_{x \in X^c} [B(x)]^c \tag{1}$$

and

$$\delta_B X = \bigcup_{x \in X} B(x). \tag{2}$$

Alternatively, SV binary erosion and dilation may be defined based on set intersection and inclusion as

$$\varepsilon_{\widehat{B}}X = \{y|B(y) \subset X\} \tag{3}$$

and

$$\delta_{\widehat{B}}X = \{y|B(y) \cap X \neq \emptyset\} \tag{4}$$

where $\widehat{B}$ denotes the transposition of B, which may be defined as the set of ancestors of B according to

$$\widehat{B}(x) = \{y|x \in B(y)\} \tag{5}$$

with $y \in D$. This definition is non local, and cumbersome since the computation is done by exhaustive search. Notice the difference from the case of a TI SE where the transposition is a mere set reflection, i.e., $\widehat{B} = \{x|-x \in B\}$.

Adding arithmetic, (1)–(4) can be used to perform other operations and algorithms, e.g., morphological gradient $g = \delta - \varepsilon$ or morphological Laplacian $\mathcal{L}(I) = \delta I - 2I + \varepsilon I$. To form morphologic filters, e.g., opening $\gamma_B = \varepsilon_B \circ \delta_{\widehat{B}}$, closing $\varphi_B = \delta_B \circ \varepsilon_{\widehat{B}}$, or more complex filters, one generally has two options: i) combine adjoined definitions (1) and (4), or (2) and (3), or ii) use (5) to transpose B [3], [20].

The SE transposition (5), as well as the set inclusion/intersection versions of erosion/dilation, i.e., (3) and (4), are non local. This means that to compute the result at some point x, one needs to examine the input at unknown points y. Therefore, the result cannot be generated directly by the presented algorithm. To obtain adjunction and form filters, one needs to use the Minkowski addition/subtraction-based definitions in (1) and (2), together with precomputing the transposed SE according to (5).

A. Supported Structuring Elements

The structuring element B defines which pixel values in the input image I to include in the calculation of the output value. Whereas the geometric shape of a TI SE is constant throughout the input image, the shape of a SV SE may change from pixel to pixel in the generalized form. Restricting the set of allowed shapes $\mathcal{C}$ and the size distribution allows design of more efficient algorithms.

Shape: The algorithm is based on computation of distance function to object edges. Decomposing the computation of B into columns brings restriction to $\mathcal{L}_1$. Hence, in the HW realization presented below, the allowed SE shapes are restricted to rectangles (including $\mathcal{L}_1$-balls, squares). Note that Section V-A discusses an extension to a richer class of shapes.

Scan Order: Other restrictions are required if the algorithm is to be implemented in low complexity and low memory architectures with no intermediate storage. Usually, pixels arrive in a stream in raster scan order and output pixels are produced in a stream. Therefore, the output at location (i,j) cannot be produced until the entire neighborhood has been processed. Consequently, there is a latency L between the input and the output stream. For some pixel (i,j), the latency is given by

$$L(i,j) = N_d(i,j) \cdot I_w + N_r(i,j) \tag{6}$$

where I_w is the width of the image I, N_d and N_r are the coordinates of the origin offset from the bottom-right corner ($\mathrm{d} = \mathrm{down}$ and $\mathrm{r} = \mathrm{right}$), illustrated in Fig. 3.

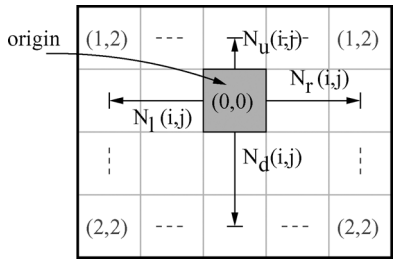

Fig. 3. Example of a 5×4 ($\mathrm{width} \times \mathrm{height}$) rectangular SE with $B(i,j) = (N_u, N_l, N_d, N_r) = (1,2,2,2)$.

When using a TI SE for the entire image, i.e., $B(i,j) = B$, $\forall i,j$, the latency is constant and the raster scan order is maintained. However, if the structuring element changes from pixel to pixel, the latency varies. For unconstrained SE sizes, the output pixels will be produced in a different order with the necessity to store them in intermediate memory to retain raster scan order.

Under constraints, the intermediate storage may be dropped. From (6), $\Delta L/\Delta N_r = 1$ and $\Delta L/\Delta N_u = \Delta L/\Delta N_l = 0$ may be derived, which means that increasing/decreasing the size of the SE by one pixel to the right will increase/decrease the latency by one. Adding above and to the left has no impact on latency since these pixels have already been read.

Unitary changes of L from pixel to pixel, i.e., $|\Delta L/\Delta i,j| = 1$ can be handled with no additional memory, by stalling the input or output. Indeed, if N_r increases/decreases, the latency L increases/decreases, and the output/input is stalled. Stalling the output means that two input pixels are read before the next output value is calculated, whereas stalling the input means that two pixels are output before the next input pixel is read.

In order to avoid additional intermediate storage, for the rest of the paper, a restriction is placed on the class $\mathcal{C}$ of allowed shapes to be rectangles, not necessarily symmetric around the origin.[1] Therefore, $B(i,j)$ becomes a function $B : \mathbb{Z}^{+2} \rightarrow \mathbb{Z}^{+4}$, i.e., for every pair of coordinates (i,j). The function $B(i,j)$ yields a quadruple (N_u, N_l, N_d, N_r) defining the position of the origin with respect to the edges of the rectangle. These parameters are tied to the width and height of B by $B_w = N_l + N_r + 1$ and $B_h = N_u + N_d + 1$. The maximum width and height found in the collection of $B(i,j)$, $\forall i,j$, are denoted $\max(B_w)$ and $\max(B_h)$, respectively. Fig. 3 shows an example of a structuring element $B(i,j) = (1,2,2,2)$, being a 5 by 4 ($\mathrm{width} \times \mathrm{height}$) rectangle with the origin offset by 2 rows and 2 columns from the lower-right corner.

From (6), $\Delta L/\Delta N_d = I_w$ means that increasing the size of the SE by adding one bottom row, will increase the latency L by the entire width I_w of the image. This substantial change of latency can not be handled without using an additional buffer. This means that from pixel to pixel, the rectangle can grow/diminish by one at all sides, except of adding/deleting one bottom row, authorized only between two image row

$$\left|\frac{\Delta N_{u,l,r}}{\Delta x, \Delta y}\right| \leq 1, \left|\frac{\Delta N_d}{\Delta y}\right| \leq 1 \text{ and } \left|\frac{\Delta N_d}{\Delta x}\right| = 0. \tag{7}$$

[1]An asymmetric origin is useful for even widths or when approximating eccentric amoebas by rectangles.

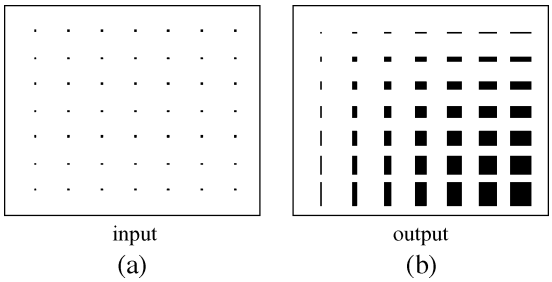

Fig. 4. (a) Synthetic test image containing black dots on a grid, corresponding to the foreground. (b) Dilation (2) obtained with similar SE as in Fig. 1, i.e., a rectangle increasing in size from the top left corner of the image downwards to the right.

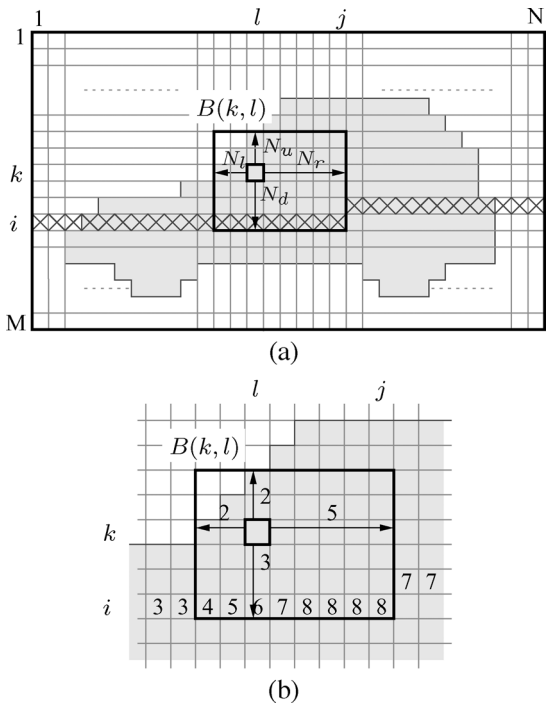

Fig. 5. (a) Synthetic input image when processing a pixel at location (k,l) with $B(k,l)$. (b) Zoom in of when processing the same pixel using $B(k,l) = (2,2,3,5)$ as structuring element. The numbers in bottom row of $B(k,l)$ show the current distance values, which saturates at the value 8.

An example of synthetic test data (640 × 480 pixels) is illustrated in Fig. 4(a). The image contains a set of black spots (on a uniform grid of 60 pixels). Applying a dilation to this input image will enlarge the black spots. Fig. 4(b) illustrates a dilation obtained with a rectangular SE progressively increasing in size from the top-left towards the bottom right corner of the image: $B(i,j) = (N_u, N_l, N_d, N_r) = (i/20, \mathrm{j}/20, \mathrm{i}/20, \mathrm{j}/20)$.

Note that these restrictions are not valid in applications where the SE size depends on the content of the image, e.g., contour filtering, object restoration (Fig. 2). However, as discussed previously, the constraints in (7) only concern the stream implementation capability, and can be relaxed if an intermediate storage is available, see Section V-A.

III. Algorithm Description

The algorithm reads the input image I and writes the output image O sequentially in raster scan order. Let (i,j) denote the current reading and (k,l) current writing position. Fig. 5(a) gives a synthetic example image $I(M,N)$ containing one object—a car. The object constitutes of pixels equal to 1, and the background constitutes of pixels equal to 0. Obviously, by causality, the reading position (i,j) precedes the writing position (k,l). The latency L between reading and writing the data depends on the size and location of the origin of the currently used structuring element $B(k,l)$, defined in (6). Since $B(k,l)$ varies for different coordinates, the latency L will also vary.

The SE shape function B is a parameter of the morphological operation and is also read in the raster scan order at the same rate and position as the output image O.

The reading (i,j) and writing (k,l) positions are bound by

$$\begin{aligned} i &= \max\left(1, \min\left(k + N_d(k,l), M\right)\right) \text{ and} \\ j &= \max\left(1, \min\left(l + N_r(k,l), N\right)\right). \end{aligned} \tag{8}$$

Suppose the currently processed pixel be (k,l) and that the corresponding structuring element $B(k,l)$—placed by its origin at (k,l)—has just been read. Recall the size of $B(k,l)$ is coded by (N_u, N_l, N_d, N_r), e.g., equal to (2, 2, 3, 5) in the example shown in Fig. 5. The input data need to be read to the bottom right position of $B(k,l)$, indicated as (i,j).

The algorithm proceeds by decomposing the erosion into columns. In each column $1,\ldots,N$ of the input image I, the algorithm keeps track of the distance $d(1,\ldots,N)$ from the currently processed line to the closest upward zero (background). For each column j, the distance $d(j)$ is updated as I is sequentially being scanned according to

$$d(j) = \begin{cases} 0 & \text{if } I(i,j) = 0 \\ d(j) + 1 & \text{if } I(i,j) = 1. \end{cases} \tag{9}$$

If $I(i,j) = 1$, i.e., belongs to the object, the distance $d(j)$ is incremented, otherwise the pixel belongs to the background, and $d(j)$ is reset to zero.

In Fig. 5(a), currently known distances are indicated by $\times$. Notice that for the currently processed pixel $O(k,l)$, the distances are calculated on a different row i. The corresponding distance values for this particular example are shown in Fig. 5(b). These distances are then compared, column-by-column, to the height of the currently used structuring element $B(k,l)$, given by $N_u + N_d + 1$. This evaluation, at position $O(k,l)$ in the output image, can be formalized as

$$\text{if the comparison} \quad d(j) \geq N_u + N_d + 1 \tag{10}$$

yields TRUE, for all $j \in [\max(1, l - N_l), \min(l + N_r, N)]$, then at position $O(k,l)$ write 1, else write 0. The whole algorithm can be written as follows.

Algorithm:

```
for k = 1...M
    for l = 1...N
        read B(k,l)
        read I up to (i,j)                    (8)
        update d up to j                      (9)
        O(k,l) = AND(d(j) ≥ N_u + N_d + 1)    (10)
        write O(k,l)
    end
end
```

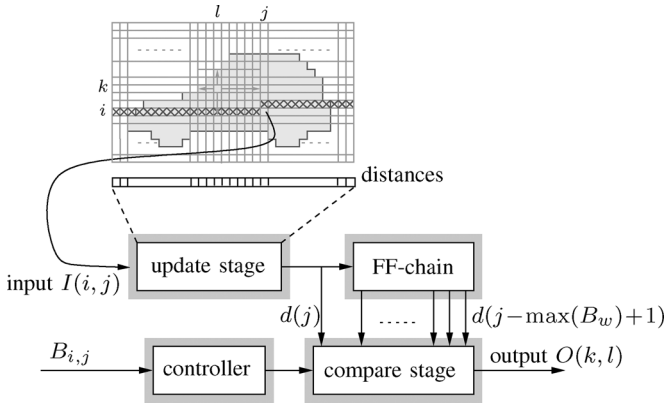

Fig. 6. Block diagram of the proposed algorithm.

AND means 1 if comparisons for all j such that $j \in [\max(1, l - N_l), \min(l + N_r, N)]$, yield TRUE, else 0 [see (10)]. For example, since the distances in the example shown in Fig. 5, i.e., $d(l-2) = 4$ and $d(l-1) = 5$, are smaller than the height $N_u + N_d + 1 = 6$ for the pixel located (k, l), the output at $O(k, l)$ is 0.

The distance calculation is an independent process of the morphological operation being performed, resulting in that the memory content is unrelated to the dimensions and origin of $B(i, j)$. This means that no information about a former B propagates in the algorithm. It is this algorithmic property that allows an adaptable SE, different for each individual pixel.

A. Block Diagram

A block diagram of the proposed algorithm is illustrated in Fig. 6. A controller is needed to stall the input and output depending on how parameters for the structuring element change. Based on these control signals, the distances to the closest upward zero, stored in the update stage, are updated. The output value from the update stage is always equal to the last calculated distance for the column of the current pixel according to (9). This distance is used as input to the compare stage and to the serially connected Flip-Flops (FF-chain), in order to let the distances propagate to be used in multiple calculations. The distances stored in the FF-chain (for the previous columns) are all used as inputs to the compare stage and the controller selects for each pixel which of these distances to include in the calculation, i.e., which distances that are to be used in (10). The selected distances are then compared to the height given by the current B. If all are greater or equal to this height, output 1 else 0 at the current location of the origin.

B. Software Implementation

Due to algorithmic properties such as the stream-like processing and in-place execution, the algorithm is applicable for software applications. As an example, if coded in C, the algorithm uses a small amount of memory (one image line) and runs very fast even for large images. Experiments on an Intel Centrino 2-GHz PC running Linux show that when eroding an image with a resolution of 1,000 × 1,000 using a SV rectangular SE of up to 100 × 100 pixels (similar to the one used in Fig. 4), takes ≈81 ms. Eroding an even larger image with a resolution of 10,000 × 1,000 image using the same SE takes 760 ms. The execution time scales linearly with the image size even for extremely large images, mainly coming from the stream-like memory access pattern.

IV. Architecture

A HW architecture for the proposed algorithm is illustrated in Fig. 7. The architecture is divided into three stages: update, FF-chain, and compare (refer to Fig. 6). In the update stage, a row memory $(\mathrm{mem_{row}})$ stores the distances for each column in the input image and for each incoming pixel: if a 0 is encountered, the sum is reset to 0, else increased by 1. This is implemented as an incrementer and a multiplexer (placed in the middle of this stage in the figure). The input from the FIFO (First In First Out) is the control signal to the multiplexer, which outputs the reset or the increased sum for further processing. If the distance is equal to the maximum supported SE height $\max(B_h)$, the sum saturates at this value, which also is the initial value in order to leave the result unaffected at the image borders.

The FF-chain contains delay elements that stall the distances $d(j - \max(B_w) + 1)$ to $d(j)$, which may be used in the current calculation, i.e., may be evaluated against the columns in the current $B(i, j)$. The FF-chain has individual access to the entries (distances), and is implemented as a series of FFs that enables each distance to propagate as long as they are to be reused in a calculation. The block also includes multiplexers for initialization on a new row in the input image.

The compare stage compares stored distances to the height of the SE. The number of comparators equals the maximum supported SE width, $\max(B_w)$. The results from the comparators serve as input to the logic AND-operation. Notice that the fan-in to this unit increases linearly with $\max(B_w)$ and, thus, affects the critical path and is the major bottleneck of the architecture. Hence, for large SEs or high speed applications, a pipeline may be inferred. Using pipelining, one or several additional delays are required to synchronize the output with the data valid signal.

The CTR block in Fig. 7 manages all control signals in the architecture based on $B(i, j)$: the enable signal to decide the number of active comparators (enable), which operation to perform (ε/δ), and also border handling. By default, the architecture performs a logic AND-operation (minimum) on the selected distances, i.e., a subset of $d(j - \max(B_w) + 1)$ to $d(j)$, which in mathematical morphology corresponds to an erosion. To perform a dilation, simply calculate the distances to the closest upward one for each column and perform a logical OR-operation (maximum). This is due to the duality nature, i.e., $\varepsilon_B I = (\delta_B I')'$, where $'$ is the bit inverse. Therefore, the other way to obtain a dilation and still use the default operation is to simply invert the input and the output, accomplished in HW by placing a multiplexer and an inverter at the input and the output of the architecture, shown in Fig. 7.

A. Handling the Borders

Sliding the structuring element over the input image, some output values are based on evaluating neighborhoods that require pixels located outside the image borders. These pixels, or

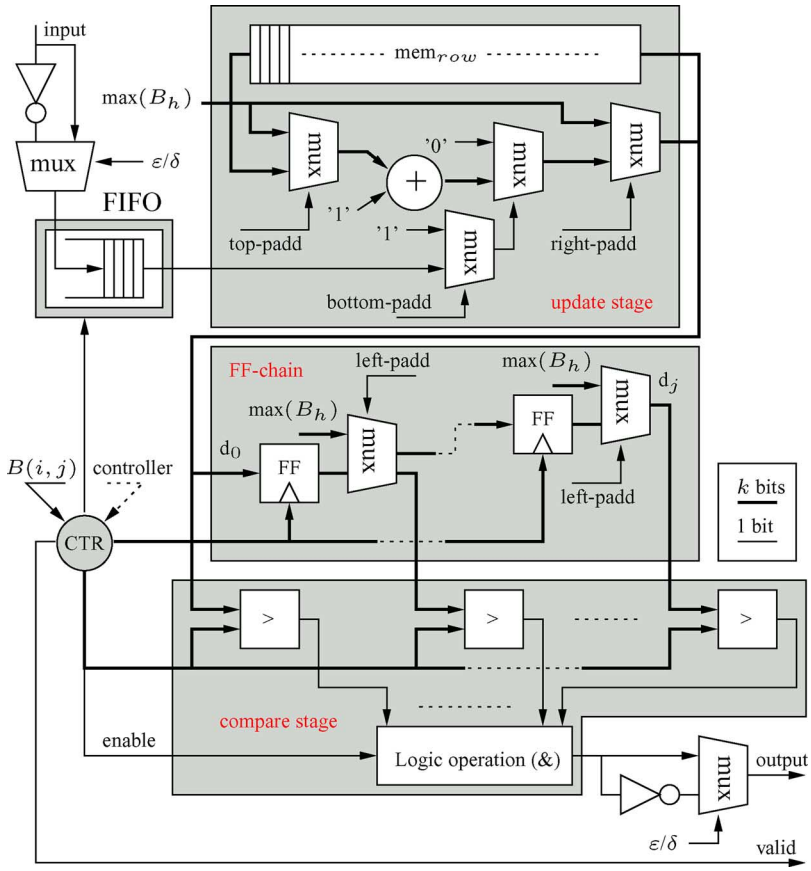

Fig. 7. Overview of the implemented architecture. Note that the data valid signal is generated by the CTR block.

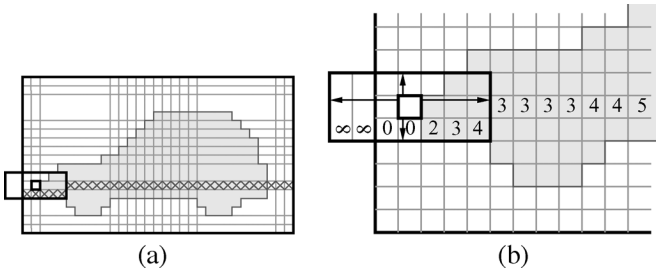

Fig. 8. (a) Illustration of when the structuring element stretches outside the image (b) and when the distances outside the image are assumed infinite; they will not affect the erosion.

in our algorithm distances, are referred to as padding. An example of a SE requiring left padding is shown in Fig. 8(a). The current architecture manages the padding pixels in one of two ways: precalculated initial values (top, right, and left padding) or pixels inserted into the data stream (bottom padding). The result is a less complex controller but with the drawbacks of requiring two clock domains and an input FIFO. The padding control is included in CTR in Fig. 7 with corresponding control signals, i.e., left-, top-, right, and bottom-padd.

Assume a rectangular SE, e.g., $B_{i,j} = 1, 3, 1, 3$, calculating the second output pixels of a new row is an example requiring left padding [Fig. 8(a)]. When starting at a new row, the distances to the left of the first column are assumed to be infinite, as illustrated in Fig. 8(b). This assumption is implemented as initial values equal to $\max(B_h)$, which are inserted simultaneously by using the multiplexers in the FF-chain stage in Fig. 7. This procedure causes the distances located beyond the image borders not to affect the calculation. When reaching and extending the structuring element beyond the right image border, the same initial value is inserted into the data stream and sent to the compare stage through the rightmost multiplexer in the update stage.

Using the same assumption as above when processing the first row in an image, the distances to the closest upward zero for the preceding row is assumed infinite. Again, this is implemented as initial values equal to $\max(B_h)$ (inserted into the adder through the leftmost multiplexer in the update stage in Fig. 7). The initial values are updated with the pixel value in the input image and the result is sent to both the compare stage and written back into the row memory.

Reaching the bottom segment of the input image, the structuring element can stretch outside the bottom border. Depending on the actual height of $B_{i,j}$, additional "1"s are inserted in the pixel stream (at most $\lceil \max(B_h)/2 \rceil$) through the lower multiplexer in the update stage. This insertion is necessary to handle the different latency that will occur in a video stream if different sizes of the structuring element are used at the end of one image to the beginning of the next. During the insertion of these extra pixels, the input data stream is stalled (requiring the FIFO on the input). Once the last pixel has been processed, the erosion operation is complete and starts over with the next frame.

B. Coding the Structuring Element Size

The structuring element size is controlled by the function $B(i,j)$ through the parameters N_u, N_l, N_d, N_r, defined in Section II-A. The parameters are generated outside the architecture and are sent as input in parallel with the input pixel stream to the controller in Fig. 7. Formally, $B(i,j)$ becomes $B(i,j,t)$ with $I(t)$ for video sequences and it is the user's responsibility to design the application-dependent $B(i,j,t)$ generation process, which must fulfill the conditions in (7).

In order to reduce the bandwidth of $B(i,j)$, one can use efficient coding. For example, the simplest coding scheme consist of coding the difference $\Delta N_{u,l,d,r}$ between two adjacent pixels on a line, instead of coding the size $N_{u,l,d,r}(i,j)$ directly. Limiting the difference to $|\Delta N_{u,l,r}/(\Delta i)| \leq 1$, the coding can be represented by using two bits, i.e., increase, decrease, no-change, reset, corresponding to a simple Δ-code. The reset value can be used to restore B to an initial setting at the beginning of each line. Thus, coding $B(i,j,t)$ will require $3 \times 2 = 6$ bits between two adjacent pixels on a line (since N_d is not allowed to change in the middle of a line), and two more bits in between two consecutive lines to represent N_d, ending up with a total number of 8 bits to code $B(i,j,t)$.

Virtually any appropriate coding system can be used, e.g., a run-length coding applied separately to each N_u, N_l, N_d and N_r will be useful if the size remains at least partially constant in some zones. Using an efficient coding will be profitable especially if more complex shapes are used (see Section V-A), since describing arbitrary shapes requires by far more information, especially for large SEs.

C. Memory Requirements

The row memory located in the update stage stores the distances for each column and is the largest single internal component in the architecture (excluding the input FIFO). The requirement is linearly proportional to the resolution according to

$$\mathrm{mem_{row}} = \lceil \log_2 (\max(B_h)) \rceil \cdot I_w \text{ bits} \tag{11}$$

where $\max(B_h)$ is the maximum supported SE height which determines the number of bits per stored value according to $k = \lceil \log_2(\max(B_h)) \rceil$. Additional registers in the FF-chain are needed to delay the stored distances ($\mathrm{mem_{row}}$ content) serving as input to the comparators, Fig. 7. The number of registers is proportional to the maximum allowed SE width. Since their content should be compared to the maximum SE height, the number of bits in these registers is

$$\mathrm{FF_{chain}} = k \cdot \max(B_w) \text{ bits.} \tag{12}$$

Combining (11) and (12), the total memory requirement for the algorithm is equal to

$$\mathrm{mem_{tot}} = \mathrm{mem_{row}} + \mathrm{FF_{chain}} = k\,(I_w + \max(B_w)) \text{ bits.} \tag{13}$$

D. Memory Organization

Concerning the implementation of $\mathrm{mem_{row}}$ in the update stage, ideally, a value should be read, updated, and written back to this memory in a single cycle. This require simultaneous read and write operations that are normally implemented using a dual-port memory. However, this type of memory introduces an area overhead mainly due to the dual address decoders. Another observation is that the memory access pattern is the same as in a FIFO, resulting in that the address generation becomes trivial and can be implemented as a simple modulo-counter. Based on these facts, $\mathrm{mem_{row}}$ can be advantageously implemented using a single-port memory of double width and half length, two registers, a multiplexer and a controller, running on the same clock domain as the input data. As an example, consider using a resolution of 640 × 480 (width × height), supporting a maximum structuring element of size 63 × 63. Normally, a memory of size $k \times I_w = 6 \times 640$ bits with dual-port functionality is required (11). Here, instead of using a dedicated dual-port memory, a double-width, half-length single-port memory with a size of $2k \times (I_w/2) = 12 \times 320$ can be used that reads and writes two samples every other clock cycle. The memory architecture is illustrated in Fig. 9 together with a simple controller that manages the FFs and the multiplexer.

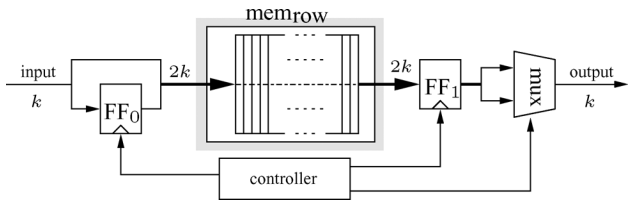

Fig. 9. Row memory implemented with one double-width single-port memory, which performs a read and write operation every other clock cycle. Note that the width of each bus is expressed in k.

The functionality of the k-bit flip-flop $\mathrm{FF_0}$ is to delay an input value in order to concatenate it with the following one. By this procedure, a bus is formed (of doubled width) constituting of two values that are written into the memory. The $2k$-bit flip-flop $\mathrm{FF_1}$ is used when reading from the memory. The multiplexer gives access to one of these two values stored on each position in the memory.

E. FIFO

In streaming data application environments, supporting a TI SE, the padding pixels (discussed in Section IV-A) may be addressed on a controller level by simply omitting them from the calculation without the need to stall the input data stream. However, supporting a SV SE requires the possibility to stall the input data stream since the latency can vary from one side of the image to the other (6). This requires two separate clock domains, separated by an asynchronous FIFO located at the input.

The size of this FIFO is a trade-off between operating frequency and memory resources. The size depends on many parameters, e.g., the relation between input data speed f_{in} and operating speed f_{op}, image size, and maximum supported structuring element size. It can optimized with respect to two objectives: i) low memory requirement, or ii) low power.

The total time it takes to stream a complete frame may be written as $t_{\mathrm{in}} = (1/f_{\mathrm{in}})(I_h I_w)$, where $I_h I_w$ is the image height × width. Furthermore, the total time t_{tot} required by the architecture to process a complete frame is determined by four factors: f_{op}, the image size, the size of the structuring element, and

the location of the origin. Assuming a centered origin, the total time for the architecture to process a complete frame (including padding) is equal to $t_{\rm tot} = (1/f_{\rm op})(I_h + \lfloor \max(B_h)/2 \rfloor)(I_w + \lfloor \max(B_w)/2 \rfloor)$, since half of the values can be inserted as initial values in the FF-chain, recall Section IV-A ($\lfloor . \rfloor$ is the floor function). The overall timing constraint for the architecture may be written as $t_{\rm in} \geq t_{\rm tot}$, or expressed in operating frequency as

$$f_{\rm op} \geq f_{\rm in} \frac{\left(I_h + \left\lfloor \frac{\max(B_h)}{2} \right\rfloor\right)\left(I_w + \left\lfloor \frac{\max(B_w)}{2} \right\rfloor\right)}{I_h \cdot I_w} \quad \text{MHz}. \tag{14}$$

Assuming that the maximum supported structuring element is small compared to the resolution, i.e., $\lfloor \max(B_h)/2 \rfloor \ll I_h$ and $\lfloor \max(B_w)/2 \rfloor \ll I_w$, then $f_{\rm op} \approx f_{\rm in}$ according to (14). Using this approximation, the architecture must at most stall $\lfloor \max(B_h)/2 \rfloor (I_w + \lfloor \max(B_w)/2 \rfloor)$ pixels during padding at the lower boundary of the image. A resolution of 640×480 at a frame rate of 25 fps, and supporting a maximum structuring element of 63×63, results in input data speed of $f_{\rm in} = 7.68$ MHz. Setting $f_{\rm op}$ to 10 MHz, the required FIFO capacity becomes 21 kb (dimensioned for the padding at the lower boundary). With this FIFO size and using (13), the total amount of memory for the complete architecture is $\approx$ 25 kb. When increasing $f_{\rm op}$ to 100 MHz, the architecture requires a FIFO with a capacity of $\approx$2 kb, reducing the total amount of memory to $\approx$ 6 kb.

The FIFO size has impact on the both the dynamic power consumption according to $P_{dyn} \propto f_{\rm op}$ [38], and the static power dissipation (area dependent). In practice, if minimizing the dynamic power is of high priority, this means operating at the lowest possible speed (for a given supply voltage), i.e., minimizing $f_{\rm op}$, resulting in a large FIFO. To summarize, the memory requirement is dependent on the operating speed $f_{\rm op}$ and memory resources can be traded for low power properties.

V. Implementation Results and Performance

Application: The algorithm runs optimally whenever the structuring element conforms to (7). This is verified in applications where $B(i,j)$ is generated by a continuous function such as in application Fig. 1, where B conforms to the image anamorphism given by the perspective.

The result in Fig. 1(d) has been obtained by applying both opening and closing operations on the motion mask in Fig. 1(b). If implemented by definition, i.e., $\varepsilon\delta\circ\varepsilon$, using (1–4), it requires three image scans, storage of two intermediate images, random memory accesses, and a latency of three frames.

The sequential memory access of our algorithm allows composing cascade filters without intermediate storage. Hence, using our algorithm, the result in Fig. 1(d) can be obtained from 1(b) in one image scan, with very low computational complexity, low intermediate storage (three image lines), and low latency (several image lines).

Architecture: The architecture has been implemented in VHDL using a resolution of 640×480 and supporting a flexible structuring element up to 63×63. Indeed, in order to correctly filter the largest objects found in the image, we have chosen the largest B to be approximately 1/10 of the image width. In general, there are no algorithmic restrictions on the largest supported structuring element size, but k in (13) will increase accordingly.

TABLE I
Resource Utilization in a Xilinx Virtex II-PRO FPGA and in UMC 0.13 μm CMOS Process. Image 640×480 and SV Rectangular SE Up to 63×63

FPGA	used	available	**ASIC**	used
Slices	807	13696	**Area** $[mm^2]$	0.31
Block RAM	3	136	**Mem**$_{tot}$ $[kb]$	24.6
LUTs	1365	27392	**Gate count** $[k]$	60
Speed $[MHz]$	80	—	**Speed** $[MHz]$	260

The implementation has been targeted for both FPGA and ASIC: a Xilinx Virtex-II PRO FPGA (XC2VP30-7FF896) and the UMC 0.13 μm CMOS process, respectively. The most important implementation results and properties for both technologies are compiled in Table I, where the area is reported containing all memory blocks. This includes an asynchronous input FIFO of 21 kb, as discussed in Section IV-C (replaced by a dual-port memory of the same size in the ASIC implementation in order to support the simultaneous read and write functionality), resulting in that memory constitutes 86% of the total area in this particular implementation. The gate count is based on a 2-input NAND-gate (5.12 μm^2).

As mentioned in Section IV, the combinatorial critical path passes through the logical operation performed in the compare stage. Pipelining this operation will not necessarily increase the speed figures found in Table I since the bandwidth to the memory is the limiting factor.

In order to compare this work to the PRR and LC architectures discussed in Section I-B, important properties are compiled in Table II as a function of the resolution and the maximum supported SE. SE support refers to the class of supported structuring elements and SE flexibility to the ability to change the structuring element between two adjacent pixels. Naturally, this should be distinguished from the ability to change the structuring element in between frames which is supported by most architectures. The complexity refers to the number of operations per pixel, e.g., in the case of PRR, number of comparators; and in the case of LC, two summations and two additions. The memory requirement is basically the same as for the LC architecture but for the additional n delay elements found in the FF-chain. $\rm T_{exe}$ is reported in number of clock cycles (CC) to process a complete frame but does not include the latency, which is present in all architectures. Table II indicates that while still maintaining low memory requirements, the ability to support SV SEs comes at the cost of the complexity increase from 4 to n, found in the compare stage as an increased number of comparators, and multiplexers, making n proportional to the maximum supported SE width $\max(B_w)$.

A. Extensions

The present algorithm situates at the extreme end of optimization, imposing restrictions on the SE shape. For more demanding applications, there are two possible extensions that increase the applicability of the algorithm.

TABLE II
IMPORTANT PROPERTIES OF VARIOUS ARCHITECTURES, WHERE I_s AND B_s ARE THE SIDE IN PIXELS OF A SQUARE INPUT IMAGE AND STRUCTURING ELEMENT ($k = \lceil \log_2(B_s) \rceil$)

Design	**PRR**[29]	**LC**[30]	**This work**
SE support	Arbitrary	Rectangular	Rectangular
SE SV	No	No	Yes
Complexity	$2k$	4	B_s
mem [bits]	$(B_s - 1)\, I_s$	$k(I_s + 1)$	$k(I_s + B_s)$
$\mathbf{T}_{exe}$ [CC]	I_s^2	$I_s^2 + B_s\, I_s$	$I_s^2 + B_s\, I_s$

Extension to Richer Shapes: The currently supported class of shapes, noncentered rectangles, includes L_1-balls squares. The benefit of this restriction is a considerable reduction of the complexity (memory requirement, number of operations per pixel, and latency), far below other algorithms supporting SV SE. This shape restriction may be relaxed to include a richer class of shapes. Indeed, convex shapes may be supported by splitting them into two parts: above and below the origin, and applying these two halves sequentially in direct and reverse raster scan. For example, L_2 balls, disks can be implemented in two scans using two half disks, see Fig. 10.

Supporting richer shapes is paid by some increase in complexity. The number of operations per pixel becomes $2B_s$ comparisons, instead of B_s previously. The memory consumption becomes $N = I_w \times I_h$, since a complete frame needs to be stored, instead of previously one line I_w only. Applying two scans sequentially also increases the latency to more than one frame, roughly $I_h + B_h$ image lines.

Concerning the HW implementation aspects, a richer shape will require feeding the FF-chain stage of the architecture (see Fig. 7) with a different value for each column. This will require a more complex controller to manage all comparator inputs. Secondly, one will need a richer coding of the structuring element B. Having a richer shape will need additional resources just for reading—at every new pixel i, j of the input image—the exact shape $B(i, j)$. A better encoding, and possibly compression, of B will become very useful to reduce the memory bandwidth which can rapidly exceed the bandwidth of the input image.

Relaxing the Size Variability Constraints: As explained in Section V, this algorithm runs optimally when B is a continuous function and its most advantageous use case is anamorphism-aware filtering, allowing to obtain results in one raster scan. Besides that, it can also be used in other applications as discussed in the introduction. For example, image coding and restoration from skeletons in Fig. 2 belong to applications where the SE size depends on the image content, i.e., circular SV SEs. Since the radius of the circles are determined by the image content, such restriction as in (7) may not be maintained.

The restrictions in (7) concern only the streaming implementation of the algorithm and can be relaxed. Indeed, the only consequence of violating (7) is that the output pixels do not arrive in the raster scan order, and that the algorithm needs a memory to store the output image.

Hence, the result in Fig. 2(c) has been obtained in two scans by dilating the skeleton in Fig. 2(b) by two half circles (upper and lower halves) with memory requirements equal to one image size.

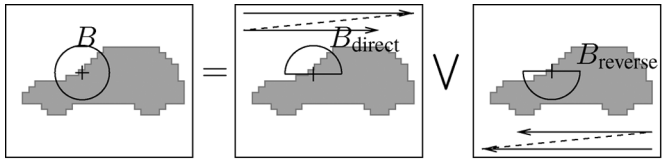

Fig. 10. Implementation of richer SE shape classes. Example: dilation by a disk, decomposed as sup of dilations by half-disks implemented in direct and reverse raster scans.

VI. CONCLUSION

This paper presents a novel algorithm for binary ε and δ supporting spatially variant, rectangular structuring elements. The memory data is decoupled from the structuring element size, which is the property that enables the structuring element flexibility. The complexity is far below other existing algorithms supporting a SV SE, which makes it compete with algorithms supporting only TI SEs. The sequential memory access pattern allows composing cascaded filters with low latency, and without intermediate storage. For more demanding applications there is an extension to support richer SE shapes (balls, diamonds) in two raster scans. Also, extending from binary to functional morphology is possible and is currently under investigation. The presented algorithm is interesting for various use cases: cascaded morphological filters running on systems under heavy time and space constraints such as embedded or communication systems or possibly also low-end user terminals.

A corresponding HW architecture of the algorithm is also presented, intended to be used as an accelerator in embedded systems. The memory requirement of the architecture is mainly proportional to the image width while the computational complexity is proportional to the maximum supported SE width. The image data is processed in raster scan order without storing the image in memory, which allows processing high resolution images on low memory systems. The architecture has been successfully verified on a Xilinx Virtex-II PRO FPGA and implemented as an ASIC in the UMC 0.13 μm CMOS process using a resolution of 640 $\times$ 480 and supporting maximum SE of 63 $\times$ 63 at 25 fps.

REFERENCES

[1] J. Serra, *Image Analysis and Mathematical Morpohology*. New York: Academic, 1982.
[2] F. Kristensen, H. Hedberg, H. Jiang, P. Nilsson, and V. Öwall, "An embedded real-time surveillance system: Implementation and evaluation," *J. Signal Process. Syst.*, vol. 52, no. 1, July 2008.
[3] J. Serra, "Structuring functions," in *Image Analysis and Mathematical Morphology*. New York: Academic, 1988, ch. 2.2, pp. 40–46.
[4] C.-S. Chen, J.-L. Wu, and Y.-P. Hung, "Statistical analysis of space-varying morphological openings withflat structuring elements," *IEEE Trans. Signal Process.*, vol. 44, no. 4, pp. 1010–1014, Apr. 1996.
[5] S. Beucher, J. Blosseville, and F. Lenoir, "Traffic spatial measurements using video image processing," presented at the SPIE Advances in Intelligent Robotics Systems, Cambridge Symp. Optical and Optoelectronic Engineering, Cambridge, MA, Nov. 1987.
[6] S. R. Sternberg, "Grayscale morphology," *Comput. Vis., Graph., Image Process.*, vol. 35, pp. 333–355, 1986.
[7] J. B. T. M. Roerdink and H. J. A. M. Heijmans, "Mathematical morphology for structures without translation symmetry," *Signal Process.*, vol. 15, no. 3, pp. 271–277, 1988.

[8] J. G. Verly and R. L. Delanoy, "Adaptive mathematical morphology for range imagery," *IEEE Trans. Image Process.*, vol. 2, no. 2, pp. 272–275, Feb. 1993.
[9] N. Bouaynaya, M. Charif-Chefchaouni, and D. Schonfeld, "Spatially-variant morphological restoration and skeleton representation," *IEEE Trans. Image Process.*, vol. 15, pp. 3579–3591, 2006.
[10] E. R. Dougherty and R. A. Lotufo, *Hands-on Morphological Image Processing,*. Bellingham, WA: SPIE, 2003.
[11] P. Maragos and R. Schafer, "Morphological skeleton representation and coding of binary images," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 5, pp. 1228–1244, Oct. 1986.
[12] D. Schonfeld and J. Goutsias, "Morphological representation of discrete and binary images," *IEEE Trans. Signal Process.*, vol. 39, no. 6, pp. 1369–1379, Jun. 1991.
[13] R. Kresch and D. Malah, "Morphological reduction of skeleton redundancy," *Signal Process.*, vol. 38, pp. 143–151, 1994.
[14] M. N. T. Natsuyama, "Edge preserving smoothing," *Comput. Graph. Image Process.*, vol. 9, pp. 394–407, 1979.
[15] Y. Wu and H. Maître, "Smoothing speckled synthetic aperture radar images by using maximum homogeneous region filters," *Opt. Eng.*, vol. 31, no. 8, pp. 1785–1792, 1992.
[16] R. Lerallut, E. Decencière, and F. Meyer, "Image filtering using morphological amoebas," *Image Vis. Comput.*, vol. 25, no. 4, pp. 395–404, 2007.
[17] J. Debayle and J. C. Pinoli, "General adaptive neighborhood image processing—Part I: Introduction and theoretical aspects," *J. Math. Imag. Vis.*, vol. 25, no. 2, pp. 245–266, 2006.
[18] J. Debayle and J. C. Pinoli, "General adaptive neighborhood image processing—Part II: Practical application examples," *J. Math. Imag. Vis.*, vol. 25, no. 2, pp. 267–284, 2006.
[19] M. Charif-Chefchaouni and D. Schonfeld, "Spatially-variant mathematical morphology," in *Proc. IEEE Int. Conf. Image Processing*, Austin, TX, Nov. 1994, pp. 13–16.
[20] N. Bouaynaya, M. Charif-Chefchaouni, and D. Schonfeld, "Theoretical foundations of spatially-variant mathematical morphology—Part I: Binary images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 823–836, May 2008.
[21] N. Bouaynaya, M. Charif-Chefchaouni, and D. Schonfeld, "Theoretical foundations of spatially-variant mathematical morphology—Part II: Gray-level images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 837–850, May 2008.
[22] M. van Herk, "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels," *Pattern Recognit. Lett.*, vol. 13, no. 7, pp. 517–521, 1992.
[23] F. Lemonnier and J. Klein, "Fast dilation by large 1D structuring elements," in *Proc. Int. Workshop Nonlinear Signal and Image Processing*, Halkidiki, Greece, Jun. 1995, pp. 479–482.
[24] M. V. Droogenbroeck and H. Talbot, "Fast computation of morphological operations with arbitrary structuring elements," *Pattern Recognit. Lett.*, vol. 17, no. 14, pp. 1451–1460, 1996.
[25] B. Kisačanin and D. Schonfeld, "A fast thresholded linear convolution representation of morphological operations," *IEEE Trans. Image Process.*, vol. 3, no. 4, pp. 455–457, Apr. 1994.
[26] J. C. Klein and R. Peyrard, "Pimm^1, an image processing ASIC based on mathematical morphology," in *Proc. 2nd Annu. IEEE ASIC Seminar and Exhibit*, Rochester, NY, Sep. 25–28, 1989, pp. P7 1.1–P7 1.4.
[27] S. Fejes and F. Vajda, "A data-driven algorithm and systolic architecture for image morphology," in *Proc. IEEE Int. Conf. Image Process.*, Austin, TX, Nov. 13-16, 1994, vol. 2, pp. 550–554.
[28] J. Velten and A. Kummert, "FPGA-based implementation of variable sized structuring elements for 2D binary morphological operations," in *Proc. IEEE 1st Int. Workshop Electronic Design, Test, and Applications*, Jan. 29-31, 2002, pp. 309–312.
[29] S. Y. Chien, S. Y. Ma, and L. G. Chen, "Partial-result-reuse architecture and its design technique for morphological operations with flat structuring element," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 9, pp. 344–371, Sep. 2005.
[30] H. Hedberg, F. Kristensen, and V. Öwall, "Low-complexity binary morphology architectures with flat rectangular structuring elements," *IEEE Trans. Circuits Syst. I*, vol. 55, pp. 2216–2225, 2008.
[31] O. Cuisenaire, "Locally adaptable mathematical morphology using distance transformations," *Pattern Recognit., J. Pattern Recognit. Soc.*, vol. 39, no. 3, pp. 405–416, 2006.
[32] *Mathematical Morphology in Image Processing*, E. R. Dougherty, Ed. New York: Marcel Dekker, 1992, ch. 8.
[33] I. Ragnelmam, "Fast erosion and dilation by contour processing and thresholding of distance map," *Pattern Recognit. Lett.*, vol. 13, pp. 161–166, 1992.
[34] P. E. Danielsson, "Euclidean distance mapping," *Comput. Graph. Image Process.*, vol. 14, pp. 227–248, 1980.
[35] G. Borgefors, "Distance transformations in digital images," *Comput. Vis., Graph., Image Process.*, vol. 34, no. 3, pp. 344–371, 1986.
[36] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, "2D euclidean distance transform algorithms: A comparative survey," *ACM Comput. Surv.*, vol. 40, no. 1, Feb. 2008.
[37] E. Dejnozkova, "Architecture dédiée au traitement d'image basé sur les équations aux dérivées partielles," Ph.D. dissertation, School of Mines, Mines, France, 2004.
[38] J. M. Rabaey, A. Chandrakasan, and B. Nikolić, *Digital Integrated Circuit*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2003.

Hugo Hedberg received the M.S.E.E. and Ph.D. degrees in electrical engineering from Lund University, Lund, Sweden, in 2001 and 2008, respectively. His doctoral thesis addressed hardware accelerators for automated digital surveillance systems.

His main research area is hardware implementations of image processing algorithms targeted for real-time embedded systems with a special interest in developing low-complexity architectures for morphological operations. He is currently with Prevas, Stockholm, Sweden.

Petr Dokladal graduated from the Technical University, Brno, Czech Republic, in 1994, as a telecommunication engineer and received the Ph.D. degree in 2000 from the University of Marne la Vallée, France, in general computer sciences, specialized in image processing.

He is a research engineer at the Centre of Mathematical Morphology, School of Mines, Paris, France. His research interests include medical imaging, image segmentation, object tracking, and pattern recognition.

Viktor Öwall, (M'90) received the M.Sc. and Ph.D. degrees in electrical engineering from Lund University, Lund, Sweden, in 1988 and 1994, respectively.

During 1995 to 1996, he joined the Electrical Engineering Department, University of California, Los Angeles, as a Postdoctorate, where he mainly worked in the field of multimedia simulations. Since 1996, he has been with the Department of Electrical and Information Technology, Lund University. His main research interest is in the field of digital hardware implementation, especially algorithms and architectures for wireless communication, image processing, and biomedical applications. Current research projects include combining theoretical research with hardware implementation aspects in the areas of pacemakers, channel coding, video processing, and digital holography.

Dr. Öwall was an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL PROCESSING from 2000–2002 and is currently an Associate Editor of the TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS.