HAL
open science

Real-Time Implementation of Morphological Filters with Polygonal Structuring Elements

Jan Bartovsky, Petr Dokládal, Eva Dokladalova, Michel Bilodeau, Mohamed Akil

To cite this version:

Jan Bartovsky, Petr Dokládal, Eva Dokladalova, Michel Bilodeau, Mohamed Akil. Real-Time Implementation of Morphological Filters with Polygonal Structuring Elements. Journal of Real-Time Image Processing, 2015, 10 (1), pp.175-187. 10.1007/s11554-012-0271-8 . hal-00789845

HAL Id: hal-00789845

https://hal.science/hal-00789845

Submitted on 19 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Journal of Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Jan Bartovský · Petr Dokládal · Eva Dokládalová · Michel Bilodeau · Mohamed Akil

Real-Time Implementation of Morphological Filters with Polygonal Structuring Elements

Received: date / Revised: date

Abstract In mathematical morphology, circular structuring elements (SE) are used whenever one needs angular isotropy. The circles – difficult to implement efficiently – are often approximated by convex, symmetric polygons that decompose under the Minkowski addition to 1-D inclined segments.

In this paper, we show how to perform this decomposition efficiently, in stream with almost optimal latency to compute gray-scale erosion and dilation by flat regular polygons. We further increase its performance by introducing a spatial parallelism while maintaining sequential access to data.

We implement these principles in a dedicated hardware block. Several of these blocks can be concatenated to efficiently compute sequential filters, or granulometries in one scan. With a configurable image size and programmable SE size, this architecture is usable in high-end, real-time industrial applications. We show on an example that it conforms to real-time requirements of the 100Hz 1080p FullHD TV standard, even for serial morphological filters using large hexagons or octagons.

Keywords Mathematical Morphology, Hardware Implementation, Alternating Sequential Filter, Parallel Computation, Polygonal Structuring Element

J. Bartovský
Faculty of Electrical Engineering, University of West Bohemia, Pilsen, Czech Republic.
Computer Science Laboratory Gaspard Monge, ESIEE Paris, University Paris-Est, Noisy-le-Grand, France.
E-mail: j.bartovsky@esiee.fr

E. Dokládalová, and M. Akil
Computer Science Laboratory Gaspard Monge, ESIEE Paris, University Paris-Est, Noisy-le-Grand, France.
E-mail: {e.dokladalova, m.akil}@esiee.fr

P. Dokládal and M. Bilodeau
Centre for Mathematical Morphology, Mines ParisTech, Fontainebleau, France.
E-mail: {petr.dokladal, michel.bilodeau}@mines-paritech.fr

1 Introduction

Since its first introduction in the late 1960's, mathematical morphology has becom in the field of image processing a useful tool for analysis of the shape or the form of spatial structures [17, 23, 24]. Over time it has found its application as a widely-used image processing technique [9, 18].

Thanks to the recent technological development of sensors, the resolution of images increased to tens of megapixels. Certain morphological operations, e.g., top-hat transformation, ultimate openings, granulometry, alternating sequential filters (ASF) [25], etc., on such large images require a large structuring element (SE), since its size should be proportional to the size of the image and its contents.

Existing hardware implementations either support rectangles using SE decomposition (fast computation, but angular anisotropic), or support arbitrarily-shaped SEs at the cost of significant performance decrease. Our work supports polygonal SEs at the performance rate of the rectangular SEs.

The paper is organized as follows: Section 2 makes a short survey of existing morphological algorithms and architectures. Section 3 outlines the basic aspects of morphological dilation and erosion, and show how to decompose the polygons into a set of lines. Section 4 describes the algorithm, and its use to decompose polygons while preserving the sequential access to data, minimal memory consumption and latency. Section 5 gives the functional implementation of the algorithm. The principal result, a parallel version using two levels of parallelism (temporal and spatial) is presented in Section 6. Finally, Section 7 presents experimental results obtained on an FPGA.

2 State of the art

First algorithms of morphological dilation were based on the definition (see Sec. 3), efficient for small structuring elements. The high computational complexity of large or arbitrarily-shaped SE motivated the search of decompositions, either by i) separation into lower dimensions, or ii) by

decomposition into atomic shapes, using the Minkowski addition. This section reviews the literature on the most known algorithmic decompositions and most efficient implementations.

i) The separation into lower dimensions allows the decomposition of n-dimensional rectangular SE into 1-D segments. The simplest method to compute 1-D dilation is an exhaustive search for maximum in the scope of SE B according to the definition (Eq. 1). Clearly, this naive solution tends to need a large number of comparisons. The number of comparisons is considered as a metric of algorithm complexity, so the naive algorithm has complexity $\mathcal{O}(l)$ as it has to carry out $l-1$ comparisons for l pixel long SE. Such complexity suggests that naive algorithm is inefficient for any large SEs. Pecht [20] proposed a method to decrease complexity based on logarithmic SE decomposition, thereby achieving $\mathcal{O}(\lceil \log_2(l) \rceil)$ complexity.

The first 1-D algorithm that reduced complexity to the constant is often referred to as HGW (it was published simultaneously in two papers: van Herk [28], and Gil and Werman [12]). The computation complexity is constant, i.e., of $\mathcal{O}(1)$, which means the upper bound of computation time is independent of the SE size. The HGW algorithm uses two buffers, which are filled by the forward, and backward, propagation of local maxima, respectively. Both buffers are then merged into the result. The major drawback of this algorithm is the requirement of two data scans: forward and reverse (so-called causal and anti-causal), which need temporary input data storing.

Lemonnier and Klein [16] propose another $\mathcal{O}(1)$ algorithm that also identifies local extrema and propagates their values. Again, the limiting forward and reverse scans are needed for every non-causal SE. Lemire [15] proposes a fast stream-processing algorithm $\mathcal{O}(1)$ for causal line SEs. It replaces the line buffers of the previous algorithms by more dedicated memory structure—double-ended FIFO (queue). The author proposed that only locally monotonous signal is necessary for a given operation. The double-ended queue serves well for such a purpose; however, the algorithm works with causal SEs only.

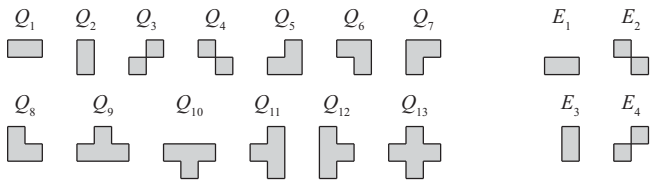
This downside was solved later in Dokládal and Dokládalová [11] who proposed another queue-based algorithm (see Section 4 for further description of his algorithm). The advantages of these queue-based algorithms are low memory requirements, zero latency, and strictly sequential access to data, the paper presents also a comparative study of these features (Table 1).

ii) the decomposition into atomic shapes allows decomposition of large SE into a sequence of small SE, see Sec. 3.1 for more information on SE composition. Xu [30] claims that any 8-convex polygon (convex on 8-connectivity grid, hence 8-convex) is decomposable into a class of 13 non-trivial indecomposable convex polygonal SEs $Q_1 - Q_{13}$ shown in Fig. 1 (a). Normand [19] reduces the class of shapes to only four 2-pixel SEs, see Fig. 1 (b), by allowing the union operator to take place in SE decomposition.

Table 1 Comparison of fast 1-D dilation algorithms.

Algorithm	SE type	Compar. per pixel	Alg. lat.	Data mem.	Working mem.
Naive 1-D	User	$l-1$	0	N	0
HGW	Sym	$3-4/l$	1	N	$2l$
Lemire	Causal	3	0	0	$2l$
Lemmonier	Sym	NC ($\mathcal{O}(1)$)	N	N	N
Dokladal	User	3	0	0	$2l$

Sym = symmetric SE; User = User-defined SE; l = length of a 1-D SE; N = line size; G = number of gray levels; NC = not communicated.



(a) Xu (b) Normand

Fig. 1 Classes of elementary SEs. Any 8-convex polygon SE is decomposable into either: (a) Xu class, or (b) Normand class while using union along with $\oplus$.

For instance, Q_{12} by Xu is obtained as $(E_3 \oplus E_3) \cup E_4$ by Normand.

The drawback of this approach is that large SEs need a long sequence of atomic operations, and may need iterate several times over the image.

For large, rotation-symmetric polygons, the problem has been solved in Soille *et al.* [26], who propose an approximation of polygons by a set of line SEs rotated by different angles. The complete dilation by a polygon still requires several iterations over the image. However, the number of iterations does not depend any longer on the size but rather on the shape. It needs three iterations for a hexagon, and four for an octogon. Each diagonal line is computed by the fast 1-D HGW algorithm oriented by the desired angle. The orientation of the SE is achieved through image partition into discrete lines (parallel, with no overlap), along which the image is processed.

More complex SEs can also be computed directly by dedicated algorithms, see Van Droogenbroeck and Buckley [27].

2.1 Hardware implementations

The hardware implementations of mathematical morphology are often called dataflow architectures in literature as they process an image in stream. They can be classified into three groups: (i) 3×3 neighborhood processors, (ii) partial-result reuse (PRR), and (iii) implementing efficient 1-D algorithm with $\mathcal{O}(1)$.

One of the first 3×3 neighborhood architectures was the texture analyzer [14]. It was optimized for linear and rectangular SE by decomposition into line segments. In [13] the authors devised PIMM1 (Processeur Intégré de Morphologie Mathématique) ASIC that contains one numerical unit for

gray-scale images and 8 binary units. However, the mechanism of computation was not communicated. Ruetz and Brodersen [22] proposed another ASIC chip that separates the supported 3×3 SE into line segments as well. Then only 4 diadic comparisons are necessary to compute the 3×3 SE.

More recently, Velten and Kummert [29] propose another delay-line based architecture for binary images supporting arbitrarily shaped 3×3 SEs. The computation of dilation is realized by OR gates (topology was not communicated, probably a tree of diadic OR gates) achieving good performance, which was further improved by spatial parallelism.

Clienti *et al.* [6] proposes a highly parallel morphological System-on-Chip. It is a set of neighborhood processors PoC optimized for arbitrarily shaped 3×3 SE interconnected in a partially configurable pipeline. The dilation itself is carried out by a tree of diadic $\max$ operators, which is pipelined for better performance.

All previous architectures use the naive method to compute the morphological operations. In order to decrease a number of comparisons for large SEs, the PRR method (name proposed in [5]) takes a partial result of a morphological operation by some neighborhood B_1 in an early stage, delays it, and reuses it later in computation by some other neighborhood B_2 obtaining thus even other, larger B_3.

One of the first PRR architectures for 1-D dilation was proposed in Pitas [21] and improved in Coltuc and Pitas [8]. The principle is based on so-called logarithmic SE decomposition. Even though it reduces a number of comparisons from naive $l - 1$ to $\lceil \log_2(l) \rceil$, it restricts a family of possible SE shapes to rectangles only.

The family of SE shapes has been enriched by Chien [5]. The authors presented more general concept of PRR that builds the desired SE by a set of distinct Xu elementary neighborhoods, see Fig. 1 (a). The decomposition process is computed by a dedicated algorithm. As a result, it supports arbitrary 8-convex polygon at the cost of additional comparisons. In [5] the PRR method was implemented as an ASIC chip supporting 5-diameter disk SE. Despite decent performance, the chip lacks possibility to control the shape of the SE.

A similar approach has been published by Déforges *et al.* [10]. Based on Normand SE decomposition [19] (a SE is decomposed into a number of causal 2-pixel SEs, which are applied in sequence or in parallel, see Fig. 1 (b)) and combined with a stream implementation, the authors propose a methodology for pipeline architecture design supporting arbitrary 8-convex SEs. The practical limitation comes from the need to create a long pipeline of atomic modules for large SE. Even thought it is not specified in the paper, such pipeline seems to be dedicated to the given SE shape and size.

Regarding the third group, implementations based on efficient algorithms, there are only two proposals in literature. Clienti *et al.* [7] implemented 1-D HGW and Lemonnier algorithms. In order to avoid a hardware-expensive reverse scan, they devised a ping-pong mirroring buffers that provide reverse-scanned data with minimal latency and memory requirements. The major drawback of the two implementations is incapability of supporting vertical, or 2-D, SE along with horizontal scan data reading.

Prior to this paper, Bartovsky *et al.* [2] reported an efficient parallel design based on the 1-D dilation algorithm [11]. However, it supports rectangular SEs only.

From the paragraphs above we can see that there are few hardware architectures capable of supporting polygonal SEs, and none of them is optimized for polygons. These architectures are usually suitable for small SEs but lose hardware resources efficiency for large SEs. In this paper we propose an architecture primarily dedicated to large polygonal SEs using an efficient algorithm.

2.2 Novelty

Most previous implementations can efficiently compute a single dilation or erosion with large or even arbitrarily-shaped SE using the decomposition into simpler, atomic operations. Provided the atomic operations concatenate, the computation can be efficiently implemented in a pipe. Obviously, more complex or larger shapes will require longer pipes. If the size is a priory unknown (e.g. function of a parameter) the complete pipe cannot be instantiated and one will need to iterate several times over the image, and store intermediate results in the memory.

Moreover, the morphological dilation is never (or rarely) used alone, but rather as a basic brick in filters. Also in a typical application consisting of several stages – for example: i) denoising, ii) object detection, and iii) measures – the dilation can even be used tens of times, with various SEs, from small (denoising) to large ones (object detection). Lastly, in geodesic operations (e.g. morphological reconstruction) the dilation can be used a variable number of times. Even though the resources of a long pipe are not excessive, such a realization lacks polyvalence and flexibility. It will only fit the targeted operator or application.

In this proposition, the atomic operation is the dilation by a large polygon (similarly to [2] with rectangles). Such atomic operation situates at a higher level of abstraction which allowing simpler decompositions, and consequently shorter pipes.

This operator has been embedded in a programmable block (dilation/erosion and the SE size and shape). It also retains all beneficial features of the inherent algorithm [11], e.g. the sequential access to data, zero latency and low memory requirements. These advantages are especially beneficial in more challenging applications, such as ASF filters, that can be computed in one or a few image scans.

3 Basic Notions

Let δ_B, ε_B: $\mathbb{Z}^2 \rightarrow \mathbb{R}$ be a dilation and an erosion on gray-scale images, parameterized by a structuring element B, as-

sumed to be flat (i.e., $B \subset \mathbb{Z}^2$) and translation-invariant, defined as

$$\delta_B(f) = \bigvee_{b \in B} f_b \; ; \qquad \varepsilon_B(f) = \bigwedge_{b \in \widehat{B}} f_b \tag{1}$$

The hat $\widehat{\ }$ denotes the transposition of the SE, equal to the set reflection $\widehat{B} = \{x \mid -x \in B\}$, and f_b denotes the translation of the function f by some scalar b. The SE B is equipped with an origin $x \in B$.

The basic concatenation of the dilation and erosion forms other morphological operators. The closing and opening on gray-scale images, φ_B, γ_B: $\mathbb{Z}^2 \to \mathbb{R}$, parameterized by a structuring element B, are defined as

$$\varphi_B(f) = \varepsilon_B[\delta_B(f)] \; ; \qquad \gamma_B(f) = \delta_B[\varepsilon_B(f)] \tag{2}$$

Furthermore, the concatenation of the closing and opening forms sequential filters, e.g., ASF. The λ-order ASF (referred to as ASF$^\lambda$) is composed of the sequence of λ closings and λ openings with a progressively increasing SE. It starts with either the closing or opening

$$\text{ASF}^\lambda = \varphi^\lambda \gamma^\lambda \varphi^{\lambda-1} \gamma^{\lambda-1} \ldots \varphi^1 \gamma^1 \tag{3}$$

$$\text{ASF}^\lambda = \gamma^\lambda \varphi^\lambda \gamma^{\lambda-1} \varphi^{\lambda-1} \ldots \gamma^1 \varphi^1 \tag{4}$$

Then i.e. the ASF$^\lambda$ starting by closing, can be written as

$$\text{ASF}^\lambda = \delta_{B_\lambda} \varepsilon_{B_\lambda} \varepsilon_{B_\lambda} \delta_{B_\lambda} \delta_{B_{\lambda-1}} \varepsilon_{B_{\lambda-1}} \ldots \varepsilon_{B_1} \delta_{B_1}. \tag{5}$$

The initial number of morphological operators 4λ can be reduced using associativity of dilations and erosions. Hence, every two consecutive dilations or erosions may be merged into one to obtain only $2\lambda + 1$ operators, such as

$$\text{ASF}^\lambda = \delta_{B_\lambda} \varepsilon_{B_\lambda \oplus B_\lambda} \delta_{B_\lambda \oplus B_{\lambda-1}} \ldots \varepsilon_{B_1 \oplus B_1} \delta_{B_1}. \tag{6}$$

3.1 SE Decomposition

The separability of n-D morphological dilation into lower dimensions is a well-known property. The decomposed dilations are then applied in a sequence according to the following equation

$$\delta_R(f) = \delta_{H \oplus V}(f) = \delta_H(\delta_V(f)) \tag{7}$$

where R denotes a rectangle, H and V horizontal and vertical line segments, respectively. This decomposition applies, in general, to convex shapes.

In order to suppress the angular anisotropy of rectangles (note the difference between a side length and a diagonal length), one prefers using circles. Regarding the implementation aspects, circles are often approximated by regular polygons (all sides have the same length) that are easily decomposable, originally described in [1, 30].

A $2n$-top ($n \in \mathbb{N}$) regular polygon SE P_{2n} can be decomposed into a set of n line SEs L_{α_i}

$$P_{2n} = \underbrace{L_{\alpha_1} \oplus \cdots \oplus L_{\alpha_n}}_{n \text{ times}} \tag{8}$$

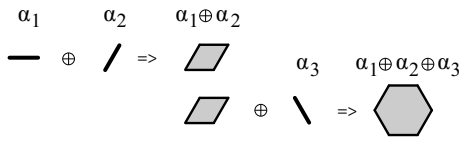
oriented at angle α_i, such as

$$\alpha_i = (i-1)\frac{180^\circ}{n} \quad [^\circ] \;\; ; i \in \mathbb{N}, i \leq n \tag{9}$$

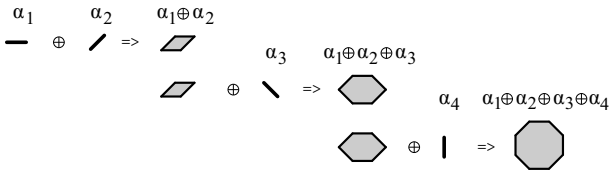
The length of all L_{α_i} is equal to the side of the desired polygon and can be computed from the circumcircle radius R as

$$\|L_{\alpha_i}\| = 2R \sin\left(\frac{180^\circ}{2n}\right) \tag{10}$$

For example, a hexagon can be obtained by three L_{α_i} oriented in $\alpha_i = \{\, 0^\circ, 60^\circ, 120^\circ\}$ on a 6-connected grid, and an octagon by four L_{α_i}, $\alpha_i = \{\, 0^\circ, 45^\circ, 90^\circ, 135^\circ\}$ using an 8-connected grid, see Fig. 2.



(a) Hexagon, $\alpha_i = \{\, 0^\circ, 60^\circ, 120^\circ\}$



(b) Octagon, $\alpha_i = \{\, 0^\circ, 45^\circ, 135^\circ, 90^\circ \,\}$

Fig. 2 Polygon SE composition of line SEs. (a) hexagon is composed of 3 segments, (b) octagon is composed of 4 segments. $\oplus$ operator stands for the Minkowski addition; α_i stands for L_{α_i}.

Hence, from (7) and (8) a 2-D dilation by a $2n$-top polygon $\delta_{P_{2n}}$ of some function $f : \mathbb{R}^2 \to \mathbb{R}$ can be obtained by n consecutive 1-D dilations $\delta_{L\alpha_i}$ by line segments oriented by α_i

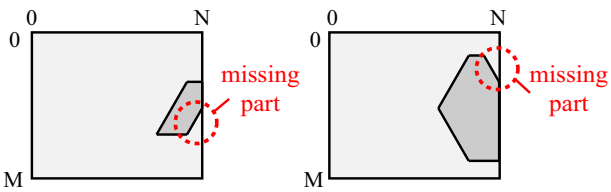
$$\delta_{P_{2n}}(f) = \underbrace{\delta_{L\alpha_1} \; (\; \ldots \; \; \delta_{L\alpha_n}}_{n \text{ times}} \; (f)). \tag{11}$$

The aforementioned decomposition holds true for the unbounded support $\mathbb{Z}^2$. However, when using real images with a bounded support $D \subset \mathbb{Z}^2$, $D = [1..M] \times [1..N]$, decomposition boundary effects appear if at least one $L_{\alpha_i} \neq \{0^\circ, 90^\circ\}$ is used. The cause is that the Minkowski addition of all decomposed line segments in (8), which are cropped by image boundaries after every L_{α_i} of that concatenation, does not necessarily correspond to P_{2n} cropped by image

boundaries just once as desired. It is expressed by the following expression where $D\cap$ represents intersection with the image support D

$$D \cap (L_{\alpha_1} \oplus \ldots \oplus L_{\alpha_n}) \neq D \cap (L_{\alpha_n} \oplus \ldots \\ \ldots \oplus D \cap (L_{\alpha_2} \oplus D \cap (L_{\alpha_1}))). \tag{12}$$

The illustrative example of such boundary effects with a hexagonal SE is depicted in Fig. 3. We can see that the composition $\alpha_1 \oplus \alpha_2$ is incomplete compared to the desired one in Fig. 2; a small part of the SE is missing. It holds true even for the entire hexagon, the composition $\alpha_1 \oplus \alpha_2 \oplus \alpha_3$ is also incomplete. It is caused by the right boundary cropping not only the final P_{2n}, but also all intermediate results. The cropped values are later missing to form an appropriate polygon section.



(a) $\alpha_1 \oplus \alpha_2$ incomplete (b) $\alpha_1 \oplus \alpha_2 \oplus \alpha_3$ incomplete

Fig. 3 Polygon SE composition without padding. The desired SEs presented in Fig. 2 are incomplete, a small triangle is missing.

This issue is solved by adding a padding to the image. The section of P_{2n} contained inside the image support is then complete, the missing part of P_{2n} is located in the padded area. The added padding contains recessive values, i.e., values that do not affect the computation of a particular morphological operator (for $f{:}D \to V$, $\wedge V$ for dilation, $\vee V$ for erosion). The thickness of the padding is different in the horizontal and vertical direction and is determined by the size of oblique segments, particularly by the half of vertical and horizontal projection

$$B_{\mathrm{H}} = \|L_{\alpha_i}\| \cos(\alpha_2)/2 \quad \text{[pixels]} \tag{13}$$
$$B_{\mathrm{V}} = \|L_{\alpha_i}\| \sin(\alpha_2)/2 \quad \text{[pixels]}. \tag{14}$$

4 Algorithm Description

This section explains the algorithmic principles involved in this paper. First, we expose the 1-D algorithm used for the dilation by line segments with arbitrary orientation. Next, we show how to combine these 1-D computations in order to obtain polygons running in stream. The issue of boundary handling is addressed afterwards.

4.1 1-D Dilation Algorithm

The implementation of (1) consists of searching the extremum of f within the scope of SE B

$$[\delta_B(f)](x) = \max_{b \in B} [f(x-b)] \tag{15}$$
$$[\varepsilon_B(f)](x) = \min_{b \in B} [f(x+b)] \tag{16}$$

The algorithm used below is based on property [11] that for some $B(x)$ (which contains its origin) the computation of the dilation $\delta_B f(x)$ needs only those values of $f(x_i)$ that can "be seen" from x when looking over the topographic profile of f, see Fig. 4. The values shadowed by the mountains - that cannot be maxima - are immediately excluded from the computation.

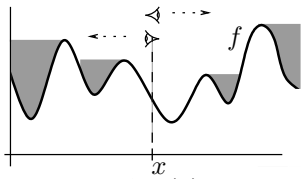


Fig. 4 Computing the dilation $\delta_B f(x)$: Values in valleys shadowed by mountains when looking from x over the topographic relief of f are useless.

From the implementation point of view, assuming a sequential access to the input data f, the dilation $\delta_B f(x)$ depends on points read after x. We say that B is non causal. One can transform a non causal SE to a causal one by utilizing the property that dilation commutes with translation

$$\delta_{B+t} f(x) = \delta_B f(x-t), \qquad \forall t \in D \tag{17}$$

These two principles are used by Alg. 1. For each pixel of some input signal $f : \mathbb{Z} \to R$, the algorithm reads one pixel $F = f(rp)$ at the so-called *reading position rp* and writes back one result pixel $dF = \delta_B f(wp)$ at the current *writing position wp*, such as $rp > wp$. The wp coordinate coincides with the origin of the SE, rp conforms to the most recent input pixel of B. Indeed, the reading position rp is its right-hand side end, which conforms to the intuitive necessity of having read all the samples covered by the SE before computing the dilation.

Alg. 1 is to be called from an outer loop iterating over the writing position in $\delta_B f$, such as *while wp<N*. The writing position wp is to be incremented whenever Alg. 1 outputs a valid value. We give below the pseudocode, for detailed description see [2].

4.2 Stream-Preserving Decomposition of Polygons

Alg. 1 can be used to compute the dilation by L_{α_i} segments in a stream. Its properties make it suitable for composing concatenated operators, namely the sequential access to input and output data, and minimal latency. Therefore, when

Algorithm 1: dF ← 1D_DIL (rp, wp, F, SE1, SE2, N)

Input: F - input signal sample $f(rp)$; rp, wp - reading/writing position; *SE1, SE2* - SE size towards left and right; N - length of the signal
Result: dF - dilated signal sample $\delta_B f(wp)$
Data: Q - Queue (first in, first out)

```
1 while Q.back()[1] ≤ F do
2     Q.dequeue() ;                          // Dequeue useless valuess
3 Q.push({F, rp}) ;                          // Enqueue the current sample
4 if wp − SE1 > Q.front()[2] then
5     Q.pop() ;                              // Delete too old value
6 if rp = min (N, wp + SE2) then
7     return (Q.front()[1] ) ;               // Return valid value
8 else
9     return ({}) ;                          // Return empty
```

the input image is read in a horizontal raster scan mode, i.e., line by line, and every line from the left to the right, the output of Alg. 1 instance conforms to the very same scan order, delayed by some latency defined by the distance between reading rp and writing wp positions. It allows a direct connection of several Alg. 1 instances in a sequence without any need of coupling elements. The resulting 2-D SE is then obtained with minimal latency, that is as soon as all necessary data have been read.

The example of decomposition of a hexagon into three L_{α_i} is depicted in Fig. 5. The image is sequentially read by horizontal L_{0° at the reading position of the polygon (a). The result of the horizontal segment is immediately provided as an input to the first oblique L_{60° at (b) so that the reading position of L_{60° coincides with the writing position of L_{0°. By the very same rule, the result of the L_{60° is brought as input data to the second oblique L_{120° at (c), the writing position of which is the writing position of the complete polygon (d). The total latency is then defined by distance between the reading (a) and the writing position (d) of the polygon.

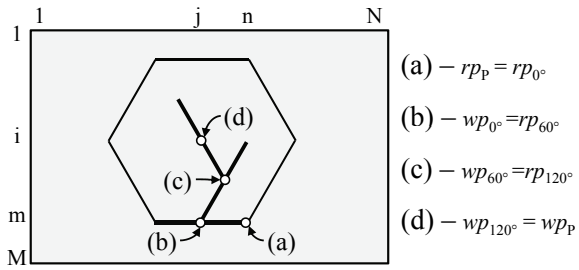


Fig. 5 Stream concatenation of three L_{α_i} into hexagonal SE P; rp/wp - reading/writing position.

The computational complexity of (11) remains almost constant w.r.t. the SE size (except the padding)

$$\mathcal{O}((N + 2B_{\mathrm{H}})(M + 2B_{\mathrm{V}})) \tag{18}$$

for an $N \times M$ image, and B_{H}, B_{V} padding sizes. Provided that of $B_{\mathrm{H}} \ll N$ and $B_{\mathrm{V}} \ll M$, it reaches the complexity of rectangles $\mathcal{O}(NM)$, see [2].

4.3 Discretely Inclined 1-D Segments

The oblique segments included in a hexagon and an octagon, i.e., L_{α_i}, $\alpha_i = 45, 60, 120, 135^\circ$, need appropriate addressing to determine the pixels to process. Note that all inclinations verify $\alpha_i \geq 45^\circ$, and the coefficients k_i verify $k_i = \tan \alpha_i \geq 1$. If we use 8-connectivity for $k_{45^\circ,135^\circ} = \pm 1$, and 6-connectivity for $k_{60^\circ,120^\circ} = \pm 2$, we can very easily generate the pixel adressing - for every inclination - by only modifying the original column index col by an additive constant $line/k_i$ such as

$$col_{\mathrm{shift}} = (col + line/k_i) \bmod (N + 2B_{\mathrm{H}}), \tag{19}$$

where the inclination deviation from the vertical direction $line/k_i$ is called *offset* and changes only with a new image line.

5 Hardware Implementation

In this section, we present hardware implementation (called line unit LU) of Alg. 1 for dilation by L_{α_i} with emphasis on the inclined segment computation. Then, we chain several elementary LU units into a pipeline to form the polygonal processing unit PU. Finally we propose the parallel polygonal unit PPU.

5.1 1-D Algorithm Implementation

Alg. 1 along with the col_{shift} addressing feature is seen as a simple Mealy finite state machine (FSM). This FSM controls all algorithm operations over the queue, rp and wp pointers etc. The state diagram (in Fig. 6) of the algorithm behavior consists especially of 2 main states $\{S1, S2\}$ and one auxiliary state EOL. The basic operation of the direct algorithm implementation can be found in [3, 4]. $S1$ state manages the dequeuing loop and pushing of a new value, code lines 1–3; the $S2$ state handles the deletion of outdated values and returns the result, code lines 4–9.

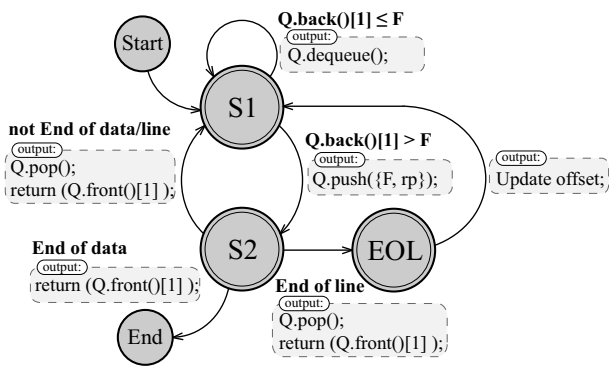


Fig. 6 State diagram of Alg. 1. Conditions of state transitions are typed in bold, output actions are located in gray rectangles.

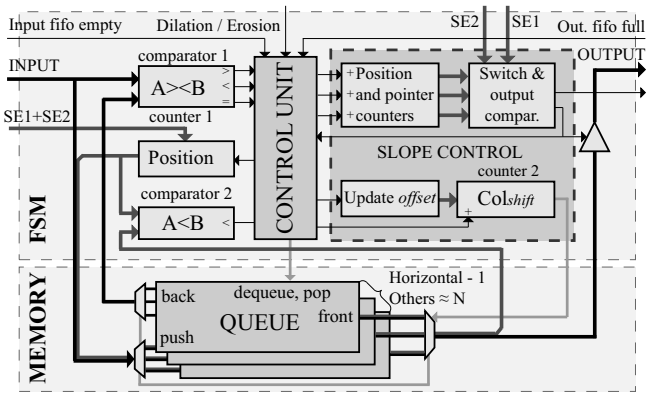


Fig. 7 Overview of the LU architecture. The FSM part manages computation, the memory part contains data storage–queues.

The auxiliary state EOL is entered only at the end of every image line. Its main purpose is to update the offset value, to determine the shifted column addressing. The generation of the necessary inclination is extremely easy since it requires only elementary operations like incrementing, decrementing or stalling.

5.2 1-D Line Unit Architecture

The architecture of the LU unit capable of dilation by different line segments is shown in Fig. 7. The basic description of the preceding version supporting only horizontal and vertical orientation can be found in [4]. Several modifications have been applied to the former version to allow inclined L_{α_i}. We have mainly added the Slope control unit that is highlighted in Fig. 7.

The LU comprises two parts: an FSM part and a memory part containing a collection of double-ended queues. FSM manages the whole computing procedure and temporarily stores values in the memory part. The memory instantiates one queue in the case of horizontal segment, N queues in the vertical case (N is the image width), or $N + 2B_{\mathrm{H}}$ queues in the oblique case. Input and output ports are multiplexed; hence a multiplexor select signal can easily address one queue to work with. The shifted column address (col_{shift}) is used as the select signal.

The processing of one pixel proceeds as follows: In the beginning of $S1$, the last queued pixel is invoked by the Back() operation from the queue and fetched to Comparator 1 where it is compared with the current sample, and dequeued if necessary (code lines 1–2). The current pixel is simply extended with the value of position counter 1 and enqueued (line 3).

The $S2$ invokes the oldest queued pair $\{F, rp\}$ by the Front() operation. This read pixel is a correct result if the set of output conditions (code line 6) is fulfilled. The deleting of an outdated value is managed by comparing the stored position value with the current one in Comparator 2.

The purpose of Slope control is to select the corresponding queue memory which is currently used by Alg. 1. The queues are addressed by the Col_{shift} counter, which is incremented with every pixel of the input image and reset at the end of the image line. The initial reset value of the col_{shift} counter is *offset* (see Section 4.3). The *offset* is updated at the end of every image line (state EOL); its value is incremented or decremented either every line or every other line according to k_i.

5.3 Polygon Unit Architecture

The LU units described above can be arranged in a sequence to form a 2-D Polygon Unit (PU). The overall architecture of the PU unit (see Fig. 8) is composed of three different-purpose parts: computation part, controller, and padding part.

The computation part mainly contains four LUs for distinct L_{α_i} orientations. There are the horizontal unit ($\alpha_1 = 0°$), the first inclined unit ($\alpha_2 = 45°$ or $60°$), the second inclined unit ($\alpha_3 = 135°$ or $120°$), and the vertical unit ($\alpha_4 = 90°$) connected in a simple pipeline; the output of each unit is read by the successive unit which processes the image by further L_{α_i}. The computation part is able to operate either with a hexagon or octagon SE. In the case of the hexagon SE, the vertical unit is bypassed.

Note that the output of every computation unit is an intermediate result image, which can be brought out for another purpose, e.g., a multi-scale analysis descriptor. Then the dilation by line, rectangle, and octagon SEs (all centered) can be obtained during a single image scan (considering units reordering). Only the Remove padding block is to be copied several times for each output data stream.

According to the boundary effects mentioned in Section 4, the inclined units need padding to extend the original image before the processing. The padding is removed after the last 1-D unit. It is carried out by a pair of dual padding blocks at the beginning and the end of the computation part.

The controller ensures the correct global system behavior. It accepts the SE diameter and the shape select signal, then it determines the particular SE sizes for every LU and padding from them, and initiates the computation. The entire set of parameters, i.e., the image width and height, all SE features (size and shape), and the morphological function select, is run-time programmable at the beginning of the frame. These parameters are run-time programmable within the upper bound specified during the synthesis.

For instance, consider the SE size upper bound a 91×91 bounding box, and octagon-capable architecture. This means, the architecture has four LU; each LU supporting at most l=31 pixels segments. During the operation – at the beginning of a frame – the SE can programmed to either of the following: a line up to 31 pixel long, a rectangle up to 31×31 pixels, or an octagon up to 91×91.

To enable processing a uniform input stream, one needs to handle unequal processing rates of LUs. It is caused by variable algorithm latency to compute a dilation for one pixel. Therefore, the balancing FIFO memories are inserted

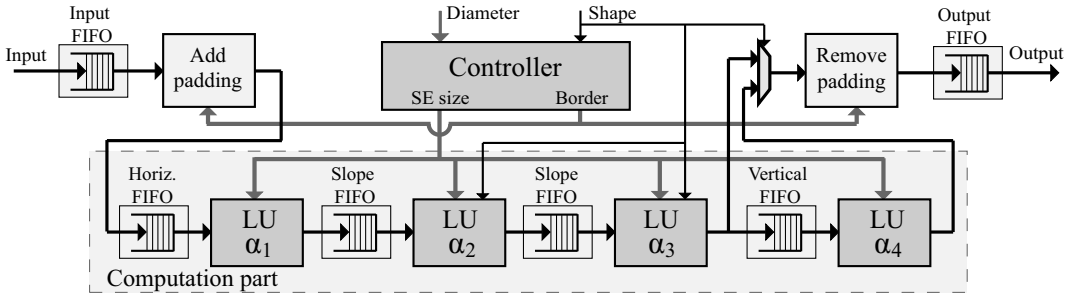


Fig. 8 Overall architecture of the polygonal PU unit. It contains one LU for each $\delta^{\mathrm{L}}_{\alpha_i}$ of (11), control, and padding units.

in front of each 1-D unit, and to the input and output ports. The depth of input and output FIFOs depends on the timing of input data stream (possibility of stalling, synchronization, etc.).

5.4 Memory Requirements

The most significant memory demand is made by the set of queues. Although the algorithm works with separated queues, the queues within each LU are merged into a single dual-port memory, mapped side by side in a linear memory space. Every queue has a related pair of front and back pointers which must be retained throughout the entire computation process in the pointer memory. This approach leads to more efficient implementation.

The LUs have the following memory requirements (considering N×M image including padding, L_{α_i} with bounding boxes $W_x \times H_x$, and bpp bits per pixel):

$$M_{\text{hor}} = W_{\text{H}}(bpp + \lceil \log_2(W_{\text{H}} - 1) \rceil) \quad \text{[bits]} \tag{20}$$

$$\begin{aligned} M_{\text{ver}} =& N((H_{\text{V}} - 1)(bpp + \lceil \log_2(H_{\text{V}} - 1) \rceil) \\ &+ 2\lceil \log_2(H_{\text{V}} - 1) \rceil) \quad \text{[bits]} \end{aligned} \tag{21}$$

$$\begin{aligned} M_{\text{slope}} =& (N + W_{\text{S}})((H_{\text{S}} - 1)(bpp + \lceil \log_2(H_{\text{S}} - 1) \rceil) \\ &+ 2\lceil \log_2(H_{\text{S}} - 1) \rceil) \quad \text{[bits]} \end{aligned} \tag{22}$$

Example: Consider a dilation of 8-bit, SVGA image (i.e., 800×600=$N \times M$) by a hexagon with radius 41 px. Such a SE is decomposed into horizontal SE 21 px wide, and 2 slope SE each 11 px wide and 19 px tall (hexagon SE bounding box is 41×37 px).

The computation memory (the queues) requires (20–22)

$$\begin{aligned} M_{\text{hor}} =& 21(8 + 5) = 273 \quad \text{[bits]} \\ M_{\text{slope}} =& (811 + 11)((19 - 1)(8 + 5) + 10) = \\ =& 200'568 \quad \text{[bits]} \end{aligned}$$

resulting in total consumption of $M_{\text{all}} = M_{\text{hor}} + 2M_{\text{slope}} \cong$ 392 kbits for the 2-D dilation by hexagon. This is far below the mere size of the image itself $M_{\text{image}} = 800 \times 600 \times 8bpp \cong 3.66$ Mbits which does not need to be stored at any moment.

6 Parallel Implementation

This section describes the Parallel Polygon Unit (PPU) that aims at increasing the computational performance while maintaining as much as possible the beneficial streaming properties of the proposed algorithm.

6.1 Partition of the Image

The parallelism is obtained by use of concurrently working units that simultaneously process different parts of the image (spatial parallelism). The number of instantiated units defines the parallelism degree (*PD*). Since the processing runs in stream, we propose a solution that transforms the input stream into a set of *PD* streams in a way to minimize the waiting-for-data periods of all units. For the sake of clarity, we use *PD*=2 in the description hereafter. A similar method has proven to be useful in [2].

The partition of the input image is twofold, see Fig. 9: an interleaved line-by-line partition for the horizontal α_1 units, and vertical stripes for the vertical and inclined $\alpha_2, \alpha_3, \alpha_4$ units. The final image partition of 2-D image is the intersection of both.

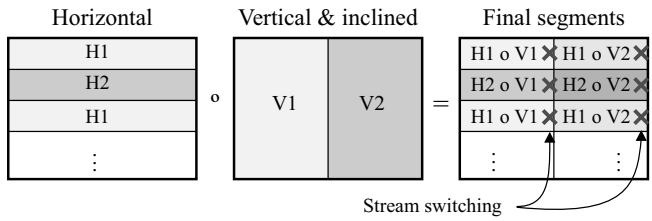


Fig. 9 Example of image partition for *PD*=2: line by line for horizontal orientation; vertical stripes for non-horizontal orientation.

Intuitively, the streams have to be transformed from one type to the other between α_1–α_2, and α_4–*output* in the PU. The transformation is done by simple circular stream switching when a partition edge is encountered. With the beginning of the image, it starts with the H1∘V1 segment on the first line. When the end of this segment is reached, the streams are switched so that segments H1∘V2 (1st line) and H2∘V1 (2nd line) are processed at the same time. Later, it proceeds

to segments H2∘V2 (2nd line) and H1∘V1 (3rd line) and so forth. In general *PD* segments located on a backward diagonal run simultaneously throughout the image (note that the streams are mutually delayed by N/PD pixels).

Processing the partition segments separately introduces undesired border effects on each partition edge. A common solution – similar to padding at image borders – is to introduce an overlap. Contrary to the padding that adds recessive values, the overlap extends a partition by a portion of the neighboring partition. The width of the overlap depends on the size of the SE, and is equal to the width of the horizontal padding B_{H}. Intuitively, the overlap introduces redundant computation, and slightly degrades the performance and minimal latency.

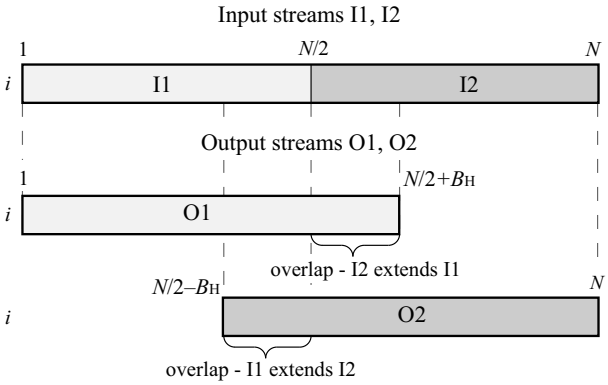


Fig. 11 Addition of overlap on one image line

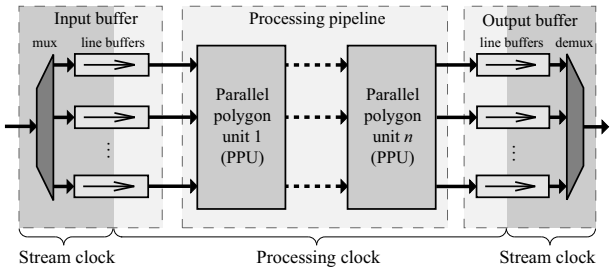


Fig. 12 Overall architecture of parallel ASF application.

6.2 Parallel architecture

At this point, all the previously mentioned principles are brought together to form the Parallel Polygon Unit (PPU). The PPU (see Fig. 10) is scalable with respect to *PD*, the number of parallel streams it can process at the time. Each stream needs one pipeline of four LUs ($\alpha_i, i = 1..4$, just like the PU), two *add overlap* blocks in front of inclined LUs, two *remove overlap* blocks behind inclined LUs, *add padding* at the front end, and *remove padding* at the back end. The PPU also contains a pair of switches to transform the streams from one type to the other, and a controller (ommited in Fig. 10).

Figure 11 shows the introduction of the overlap in the course of the i-th image line. As we know, this line is split into two streams. The streams are labelled I1, I2 before the addition and O1, O2 after (refer also to Fig. 10). The entire I1 stream plus B_{H} pixels of I2 form O1 output stream with overlap, and last B_{H} pixels of I1 and the whole I2 stream form O2.

During the overlap sections, either I1 or I2 stream is mapped to both output streams at the same time. This data duplication does not temporarily allow for parallel processing of both streams and may result in stalling of either stream. However, the effect of the overlap is negligible as long as $B_{\mathrm{H}} \ll N$.

Two important properties are to be noted: (i) input and output streams are mutually delayed by N/PD (ensured by stream switching); (ii) several PPUs can be chained into a pipe. The schematic of some application, e.g., ASF, may look like in Fig. 12. At the front end there is an input buffer transforming the input stream (which is *PD*-times faster than each of *PD* processing streams) into *PD* processing streams Hi ($i = 1..PD$). The transformation only needs i-th image line to be stored in $\{i \bmod PD\}$-th line buffer. In this manner, the processing streams are properly delayed by N/PD pixels. The output buffer transforms *PD* processing streams into one fast stream in the opposite way. One can place as many PPUs as desired between these two buffers in a pipeline or other topology.

The PPU involves the following limitations on the programmability: the image size is set before synthesis, the padding sizes $B_{\mathrm{H}}, B_{\mathrm{V}}$ are computed for the maximal SE, specified before the synthesis. The reason is that handling the varying SE and image sizes would introduce an unreasonable hardware overhead of image partition, padding, and overlap features. The SE features (size and shape) remain fully programmable as in the non-parallel case, see Section 5.3.

7 Experimental results

Hereafter we discuss the results of the proposed implementation. First, we discuss the results of a single 2-D PU and PPU unit, followed by their performance in an application, an ASF filter. We conclude by comparising them with other architectures.

The proposed PU and PPU architecture has been implemented in VHDL and targeted to the Xilinx FPGA Virtex-6 device (XC6VLX240T). The ultimate specification conforms to the following: 8-bit gray-scale images of size up to 1080p (1920×1080 px), height of L_{α_i} up to 31 px (thus, hexagon SE up to 61 px, and octagon SE up to 91 px), and support of uniform stream processing. Notice that all three previous factors affect the memory requirements that (in contrast to the PC), have significant influence on the clock frequency. Our specification implies the clock frequency of 100 MHz.

The timing with respect to (shortly w.r.t.) the image size and the size of the SE (Table 2 and 3) have been evaluated on a natural photo image. We report several measures.

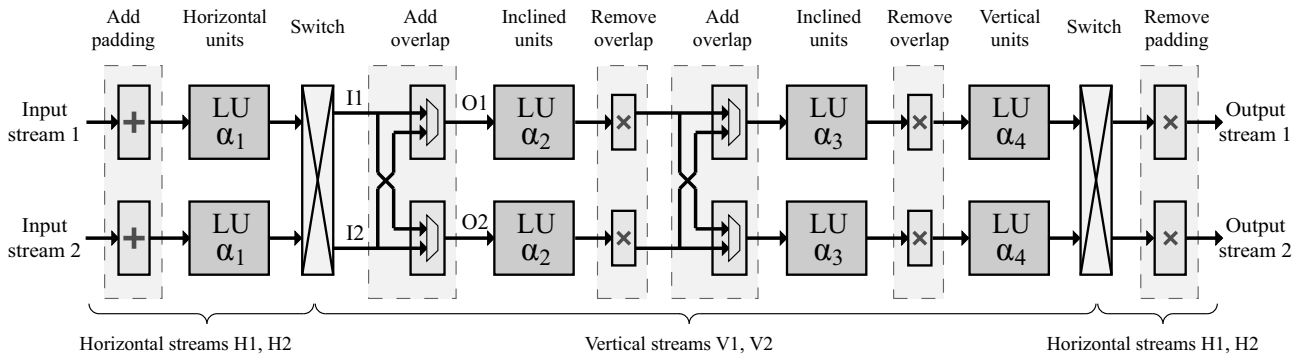


Fig. 10 Overall architecture of the parallel polygonal unit PPU for *PD*=2. The controller and balancing FIFOs are ommited.

Table 2 Timing of PU and PPU w.r.t. image size (SE size = 51 px, *PD*=6).

Image Size	VGA	SVGA	XGA	1080p
Pixel Rate (PU) [clk/px]	2.61	2.53	2.53	2.44
Latency [image line]	25	25	25	25
FPS (PU) [frame/s]	125	82	50	19
FPS (PPU) [frame/s]	599	406	257	105
Speed-up PPU vs. PU [-]	4.79	4.94	5.1	5.34

Table 3 Timing of PU w.r.t. SE size (SVGA image)

SE size [px]	21	31	41	51	61
Rate [clk/px]	2.42	2.46	2.49	2.53	2.58
FPS [frame/s]	85	84	83	82	81
Latency [image line]	10	15	20	25	30

Table 4 Speed-up of PPU w.r.t. *PD* (SVGA image, SE size = 31 px).

Parallelism degree *PD*	2	3	4	5	6
FPS [frame/s]	162	234	306	376	441
Speed-up [-]	1.92	2.77	3.62	4.44	5.22

Table 5 FPGA resources w.r.t. *PD* (SVGA image, SE size = 91 px).

PD	1	2	3	4	5	6
Registers (P)PU	787	1,644	2,469	3,215	4,019	4,850
LUTs (P)PU	2,656	4,831	7,330	9,301	11,540	14,221
Block RAM (P)PU	39	39	59	42	53	63
Registers buf	0	251	355	466	590	671
LUTs buf	0	1,296	1,929	2,545	3,158	3,748

1) The pixel rate gives the average number of clock ticks to process one pixel. It is given by the overall number of clock ticks divided by the image size. One can see that the rate is almost constant w.r.t. both the size of the image and the size of the SE. The slight variation is caused by the size of the SE which affects the size of the padding and overlap, increasing the number of effectively processed pixels, see (18).

2) Latency is expressed in a number of image lines. Note that it is strictly half the SE size. This is a further irreducible factor corresponding to the dependency of the output on the input. This corresponds to the half-height of the SE that needs time to have read enough data to compute the dilation.

3) The last measure is the throughput in terms of the number of frames per second (FPS). The ultimate result we obtain is 105 fps for the 1080p resolution, allowing the 100Hz 1080 FullHD TV standard to be processed in real time.

The speed-up PPU vs PU measures the acceleration obtained from the parallelization. The difference from the ideal upper limit (*PD*=6) is due to the overlap. With increasing image size the acceleration converges towards 6 because of the SE size (and consequently the overlap) becomes negligible with regard to the image size.

Table 4 outlines efficiency of the scalability (that is the parallelism degree *PD*) in terms of the FPS and speed-up. One can see that the real speed-up is somewhat lower than the PD. The difference is due to two factors: (i) the overlap, which demands redundant computation, and (ii) the stream switching that needs inter-stream synchronization which may introduce wait cycles.

Table 5 reveals the cost of parallelization on FPGA resources in terms of registers, LUTs, and BRAMs of the PPU and the pair of input and output buffer as shown in Fig. 12.

(a) Original (b) Filtered

Fig. 13 Example of ASF filtering. A zoom into original and the ASF[3] filtered “Mountain” image.

Table 6 Comparison of several FPGA and ASIC architectures concerning morphological dilation and erosion. N, M stand for the image width and height of respective architectures.

	Processing unit				Hardware System		Application Example ASF^6		
	Technology	Supported SE	Throughput [Mpx/s]	f_{max} [MHz]	Number of units	Supported image	Image scans	FPS [frame/s]	Latency [px]
Clienti [6]	FPGA	arbitrary 3×3	403	100	16	1024×1024	6	66.7	$5NM + 84N$
Chien [5]	ASIC	disc 5×5	190	200	1	720×480	45	12.2	$44NM + 84N$
Déforges [10]	FPGA	arbitrary 8-convex	50	50	1	512×512	13	14.7	$12NM + 84N$
This paper	FPGA	regular polygon	195	100	13	1024×1024	1	185	$84N$

7.1 Alternating Sequential Filter

The ASF filter is an essential method of morphological filtering, see example Fig. 13. Since the ASF is applied as a sequence of alternating dilations and erosions with a changing SE, it can be advantageously implemented by chaining instances of the proposed architecture into a pipeline structure. The output of each operator is immediately processed by a subsequent operator to achieve the following beneficial properties: (i) all the operators are being applied in parallel (temporal parallelism), (ii) the image is filtered with minimal latency inferred by the Minkowski addition of all SEs.

Table 7 illustrates the performance of ASF$^\lambda$ in terms of the experimentally achieved FPS and the inferred latency. Note that the frame rate of the whole ASF decreases with respect to the order λ since larger SE implies larger padding and overlap. However, the performance of the filters is comparable with the rate of a single unit in Table 2.

Table 7 Timing of ASF$^\lambda$; SVGA image size, *PD*=6.

Order of ASF λ	1	2	3	4	5	6
Number of δ, ε	3	5	7	9	11	13
Size of max. SE [px]	5	9	13	17	21	25
FPS by PUs [frame/s]	88	87	86	86	86	85
FPS by PPUs [frame/s]	491	483	466	440	415	387
Latency [image line]	4	12	24	40	60	84

7.2 Architecture Comparison

The implementation and performance comparison of our architecture with the others is outlined in Table 6. At first, we take into account single 2-D units only. Clienti [6] yields a high throughput for an elementary SE 3×3. The Chien [5] ASIC chip achieves a reasonable throughput with a small 5×5 diamond SE. Both architectures use homothecy to obtain larger SEs. On the other hand, one Déforges [10] unit supports various 8-convex SEs in one scan. As mentioned in the state of the art, the programmability of the modules, namely the possibility to control the SE shape after the synthesis is not clear. Smaller throughput is probably caused by usage of a less powerfull device than the rest of implementations.

Lets take as an application example of a compound morphological operator, consider ASF$^6 = \delta_{13\times13}\varepsilon_{25\times25}\ldots\varepsilon_{5\times5}\delta_{3\times3}$ that consists of 13 morphology operations. One Clienti's system instantiates 16 elementary 3×3 processing units. Hence, it will require 6 image scans (the entire image must be stored in the memory). Chien also uses the homothecy, therefore, as many as 45 scans are to be done. In the case of Déforges, neither FPGA occupation with respect to the the size of SE nor possibility of using multiple instances in a single chip was communicated. We consider that only one unit fits the FPGA, so 13 image scans are needed. Of course, if several units fit the FPGA surface, it will reduce proportionally the number of scans. This is true for all streaming architectures.

Obviously, between two consecutive scans the data are read/written from/into the memory that degrades performance and significantly increases latency to orders of several image scans. The dense memory traffic might also overwhelm the data bus.

From the estimated performance results for the ASF^6 in Table 6 we observe that the high use of homothecy tends to increase the number of necessary image scans. Indeed, all Clienti, Chien, and Déforges (a) whose solutions are efficient for small SE sizes and short concatenations become more or less penalized for longer concatenations; their performance drop down with the higher numbers of necessary image scans. On the other hand, Déforges (b) and our work, which does not need more than one image scan, attain the high performance for ASF^6 comparable to the performance of a single 2-D unit.

These features allow a temporal-parallel execution of all atomic operators that is essential to obtain the real-time performance for high demanding applications. In addition, the low memory requirements facilitate embedding several instances of proposed computation units into a single FPGA circuit.

8 Conclusions

It is widely known that the processing data in stream allows to reduce latency, memory consumption and increases the system throughput. Until recently, computing morphological dilations or erosions in stream was only possible for small, limited neighborhoods [5,6,10], or large rectangles [2]. Dilations by large polygons were computed iteratively, by using the homothecy. This required an external memory for intermediate data, limited the flexibility, and drastically increased system latency.

This paper opens the possibility of stream execution to morphological dilation with large polygons. Although the decomposition of polygons into the Minkowski addition of inclined lines has been known for years [1], we bring several suggestions that—combined together—allow the execution in stream.

We show how to implement dilation by inclined linear segments with sequential access to input and output data. We show how to handle border effects, and recall (since this is less known) that it requires large padding. Furthermore, we show how to partition an image to introduce efficient spatial parallelism while maintaining sequential access to data at all levels. This avoids increasing the system clock by dividing a fast data stream into several slower streams to process at a slower rate. We show how to efficiently handle the border effects on the partition.

The proposed polygon decomposition uses sequential access to both input and output data. This allows for temporal parallelism, where in concatenations like $\ldots\delta\varepsilon\delta\ldots$ all these operators run simultaneously on the time-delayed data. We attain a very low (nearly optimal) latency, which has beneficial impacts on memory consumption. No external memory is used even for large SEs and large images. All these aspects brought together allow for a considerable data throughput for sequential morphological filters. We have implemented a programmable IP block, usable in industrial systems running under heavy timing constraints satisfying up to the 100Hz 1080p FullHD TV requirements.

References

1. R. Adams. Radial decomposition of discs and spheres. *CVGIP Graphical models and image processing*, 55(5):325–332, 1993.
2. J. Bartovský, P. Dokládal, E. Dokládalová, and V. Georgiev. Parallel implementation of sequential morphological filters. *Journal of Real-Time Image Processing*, pages 1–13. 10.1007/s11554-011-0226-5.
3. J. Bartovský, P. Dokládal, E. Dokládalová, and V. Georgiev. Stream implementation of serial morphological filters with approximated polygons. In *17th IEEE ICECS*, pages 706–709, Dec. 2010.
4. J. Bartovský, E. Dokládalová, P Dokládal, and V. Georgiev. Pipeline architecture for compound morphological operators. In *IEEE ICIP'10*, pages 3765–3768, Sept. 2010.
5. S.-Y. Chien, S.-Y. Ma, and L.-G. Chen. Partial-result-reuse architecture and its design technique for morphological operations with flat structuring elements. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(9):1156 – 1169, Sept. 2005.
6. Ch. Clienti, S. Beucher, and M. Bilodeau. A system on chip dedicated to pipeline neighborhood processing for mathematical morphology. In *EUSIPCO 2008*, Lausanne, Aug. 2008.
7. Ch. Clienti, M. Bilodeau, and S. Beucher. An efficient hardware architecture without line memories for morphological image processing. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 147–156, Berlin, Heidelberg, 2008. Springer-Verlag.
8. D. Coltuc and I. Pitas. On fast running max-min filtering. *IEEE Transactions on Circuits and Systems II*, 44(8):660–663, aug 1997.
9. C. Coster and J.-L. Chermant. Image analysis and mathematical morphology for civil engineering materials. *Cement and Concrete Composites*, 23(2-3):133 – 151, 2001.
10. O. Déforges, N. Normand, and M. Babel. Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. *Journal of Real-Time Image Processing*, 2010. DOI: 10.1007/s11554-010-0171-8.
11. P. Dokládal and E. Dokládalová. Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation*, 22(5):411–420, 2011.
12. J. Gil and M. Werman. Computing 2-D min, median, and max filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):504–507, 1993.
13. J.C. Klein and R. Peyrard. Pimm1, an image processing ASIC based on mathematical morphology. In *ASIC Seminar and Exhibit*, pages P7 –1/1–4, sep 1989.
14. J.C. Klein and J. Serra. The texture analyser. *J. of Microscopy*, 95:349–356, 1972.
15. D. Lemire. Streaming maximum-minimum filter using no more than three comparisons per element. *CoRR*, abs/cs/0610046, 2006.
16. F. Lemonnier and J. Klein. Fast dilation by large 1D structuring elements. In *Proc. Int. Workshop Nonlinear Signal and Img. Proc.*, pages 479–482, Greece, Jun. 1995.
17. G. Matheron. *Random sets and integral geometry*. Wiley New York, 1974.
18. L. Najman and H. Talbot, editors. *Mathematical Morphology: From Theory to Applications*. ISTE Ltd and John Wiley & Sons Inc, 2010.
19. N. Normand. Convex structuring element decomposition for single scan binary mathematical morphology. In *Discrete Geometry for Computer Imagery*, volume 2886 of *LNCS*, pages 154–163. Springer Berlin, Heidelberg, 2003.
20. J Pecht. Speeding-up successive minkowski operations with bit-plane computers. *Pattern Recognition Letters*, 3(2):113 – 117, 1985.
21. I. Pitas. Fast algorithms for running ordering and max/min calculation. *Circuits and Systems, IEEE Transactions on*, 36(6):795–804, June 1989.
22. P.A. Ruetz and R.W. Brodersen. Architectures and design techniques for real-time image-processing IC's. *Solid-State Circuits, IEEE Journal of*, 22(2):233–250, apr 1987.
23. J. Serra. *Image Analysis and Mathematical Morphology*, volume 1. Academic Press, New York, 1982.
24. J. Serra. *Image Analysis and Mathematical Morphology*, volume 2. Academic Press, New York, 1988.
25. J. Serra and L. Vincent. An overview of morphological filtering. *Circuits Syst. Signal Process.*, 11(1):47–108, 1992.
26. P. Soille, E. J. 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.
27. M. Van Droogenbroeck and H. Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recogn. Lett.*, 17(14):1451–1460, 1996.
28. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recogn. Lett.*, 13(7):517–521, 1992.
29. J. Velten and A. Kummert. Implementation of a high-performance hardware architecture for binary morphological image processing operations. In *MWSCAS '04*, volume 2, pages 25–28, 2004.
30. J. Xu. Decomposition of convex polygonal morphological structuring elements into neighborhood subsets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(2):153–162, 1991.