GREY-SCALE 1-D DILATIONS WITH SPATIALLY-VARIANT STRUCTURING ELEMENTS IN LINEAR TIME

Petr Dokladal and Eva Dokladalova

Center of Mathematical Morphology, Mines Paris Paritech
60, Bld. Saint Michel, 75, Paris, France
email: petr.dokladal@ensmp.fr

Université Paris-Est, LABINFO-IGM, UMR CNRS 8049,
ESIEE, France
email: e.dokladalova@esiee.fr

ABSTRACT

Spatially variant morphological operators can significantly improve filtering capabilities or object detection score of various applications. Whereas an effort has been made to define the theoretical background, the efficient implementation of adaptable algorithms remained far less considered.

In this paper, we present an efficient, one-scan, linear algorithm for 1-D grey-scale dilations/erosions with spatially variant structuring elements. The proposed algorithm processes data in stream, can work in place and produces results with minimal latency. The computing time is independent of the structuring element size.

1. INTRODUCTION

The concept of adaptive filters is well known in the image processing community since years [16, 15, 12]. The general principle consists in applying a filtering process which depends on internal, local properties of image [11], [3] or on external, application dependent context, as e.g. in [16, 8], or [13] where the distance-to-camera or wide-angle deformations require adapting the structuring element.

In the domain of mathematical morphology, a considerable effort has been made to define the theoretical background. In [14], Roerding has assembled his work on spatially variant morphology in the general framework of Group Morphology. Charif-Chefchaouni and Schonfeld [4], and also Bouaynaya *et al* [1] and [2] examine a theory of spatially variant binary and functional mathematical morphology where the structuring elements can vary both in size and shape.

There is an evident lack of efficient implementations on gray-scale images needed for practical applications, e.g. Lerallut [11] states running time in 3-D images up to hours.

1.1 Efficient implementations

Whereas the efficient algorithms have beeen proposed for translation invariant structuring elements (SE) for both binary and grey-scale images, the efficient algorithms addressing the spatially variant SE are very limited.

Van Herk [17] proposes a 1-D fast algorithm, independent of the SE size, requiring three scans. The extension to translation invariant SE is not straightforward as the algorithm relies on a fixed division of the processed line by the size of SE.

Lemonnier and Klein [10] (also [9]) propose a fast dilation with large 1D for symetric SE that completes in two scans of each line. The principle is the propagation of maxima according to the SE size. The first scan can result an error to be corrected by the second scan. The extension to adaptable SEs is either difficult or impossible.

Van Droogenbroeck and Talbot [7] propose an algorithm based on a histogram, updated as the SE slides over the image. The extension to spatially variant SE is direct. Computing the histogram however requires additional memory resources and brings in an additional computing complexity. Notice also that the histogram is not usable on floating-point data.

In the domain of adaptable SE, several efficient algorithms have been published for binary domain, see e.g. [6]. It proposes a spatially variant binary morphology, using as kernels balls in various norms. Dilations are obtained by thresholding the distance function to objects. Local adaptability is achieved by thresholding at different levels. The computational complexity is the one of computing the distance. However, the only possible extension to the grey-scale domain is by decomposing the image into a set of binary images, bringing in an overwhelming increase of computational burden.

Hedberg *et al.* [8] propose an efficient algorithm for spatially variant binary morphology. The algorithm accepts non centered resizable rectangles, uses little memory and its latency strictly equals the one imposed by data dependency.

In the gray-scale domain, Cheng [5] proposes a fast algorithm for SE with adaptable shape and size. The algorithm proceeds by SE decomposition in smaller 2-D SE. Hence, the computing complexity heavily depends on the SE size and shape.

The algorithm presented in this paper is one-scan, linear algorithm for 1-D dilation with spatially variant, non centered SEs. The computing time is independent of the SE size. It's can be considered as an extension to gray-scale domain of the Hedberg algorithm.

The paper is organized as follows. Section 2 discusses the constraints applying to adaptable SEs. Then the algorithm is described in the Section 3. Finally Section 4 gives some application results. The text concludes with discussion of the properties and extension to higher dimensions.

2. ALGORITHMIC ISSUES

Consider a 1-D binary object $X : \mathbb{R} \to \{0,1\}$, see Fig. 1a. Consider a collection of flat, one-sided structuring elements B defined for all $x \in \mathbb{R}$ called origin by the length $\beta : \mathbb{R} \to \mathbb{R}$ measured from x. Hence, every B is an interval in $\mathbb{R}$ delimited by $(x, x+\beta(x)) \subset \mathbb{R}$. Notice that β can be positive or negative for SEs stretching right- or leftwards from the origin.

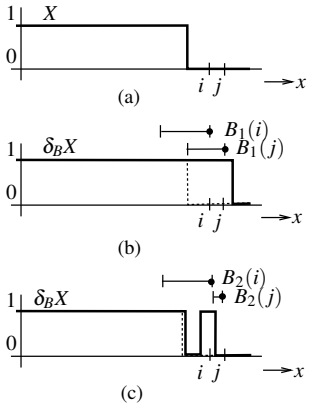


Figure 1: (a) A 1-D binary object X. (b) Dilation $\delta_B X$ with a flexible SE $B(x)$. (c) Example of a discontinuous result $\delta_B X$ with B varying "too fast".

The morphological dilation of X by B is defined $\delta_B X = \cup_{b \in B} X + b$. Fig. 1b shows an example of dilation of X by a function B_1, illustrated at i and j (the origin of $B_1(i)$ is given by the black dot above i). Notice that $B_1(j)$ may be different from $B_1(i)$. The dilated object $\delta_{B_1} X(x) = 0$, for $x = i, j$.

Fig. 1c shows the same X dilated by a different function B_2. The dilated object $\delta_{B_2} X(i) = 0$ but $\delta_{B_2} X(j) = 1$ since $B_2(j)$ doesn't extend inside X as does $B_1(j)$ in Fig. 1b. Dilating by B_2 has created an echo of X at i instead of dilating it.

SE size map B preventing this undesired behaviour needs to verify following restrictions :

$$\left| \frac{\partial \beta}{\partial x} \right| \leq 1 \tag{1}$$

For the rest of the paper, consider $\mathbb{Z}$ instead of $\mathbb{R}$, and bidirectional structuring elements as intervals $B : \mathbb{Z} \to (a,b) \subset \mathbb{Z}$, $a \leq x \leq b$, encoded by distance from the origin x towards left and right $L(x) = x - a$ and $R(x) = b - x$. Hence, $\forall x \in \mathbb{Z}$ we have $B(x) = (L, R)$, with $L, R \in \mathbb{Z}^+$.

The condition Eq.1 becomes

$$\left| \frac{\Delta L}{\Delta x} \right| \leq 1 \text{ and } \left| \frac{\Delta R}{\Delta x} \right| \leq 1, \tag{2}$$

These conditions allow a fast implementation of dilations, and appear several times below throughout the description of the algorithm.

3. ALGORITHM DESCRIPTION

Dilation and thresholding commute. This well known property is utilized for brute-force implementations of dilations of functions. A function is decomposed into a collection of binary cuts obtained by thresholding it at every level. Every cut is dilated and the function recomposed by superposition. The fact that not all binary dilations in this collection are necessary gave birth to the following algorithm. The algorithm consists of two main stages.

1) Anti-causal to causal conversion

Suppose input data f arriving in a stream, see Fig. 2b. Any operation utilizing a bi-directional structuring element $B = (L,R)$ with $R > 0$ is an anticausal operation, Fig. 2a. Computing the result $\delta f_B(x)$ depending of future samples of the signal will require storing f in memory. Utilising the property that dilation and translation commute

$$\delta_{B-t} f(x) \equiv \delta_B f(x+t)$$

any anticausal $B = (L,R)$ has a causal equivalent $B^< = (L+R, 0)$, see Fig. 2c, yielding, after translation, an identical result

2) Propagating values

The algorithm reads in a stream the input data and the local structuring element size and writes in a stream the output, every sample is read/written once and only once. No jumps ahead or backwards are used. Notice that the reading and writing positions are not the same. Their difference is the algorithm latency.

See the result by definition Fig. 2f, dilating f by some B assigns $\delta_{B(x)} f(x)$ the maximum found in the interval covered by $B(x)$. Using a causal $B^<$ can be seen as propagating some values $f(x)$ a number of positions towards right. The propagation distance, see Fig.2d, is determined by finding the last $B^<(x+k)$, $k > 0$, that still includes x. Formally, the signal value $f(x)$ is dilated at most until $x + d(x)$, i.e. $d(x)$ steps ahead:

$$d(x) = \max_{k \geq 0} \{k \mid x + k - L(x+k) \leq x\} \tag{3}$$

Utilizing Eq. 2, it can be shown that Eq. 3 can be implemented efficiently by

$$d(x) = \max_{k \geq d(x-1)} \{k \mid x + k - L(x+k) \leq x\} \tag{4}$$

a scheme which allows sequential reading on both signal and structuring element data.

Algorithm

Not all values can propagate as soon as they are read. This happens whenever a higher value is already being propagated. Smaller, more recent values need to be stored for *possible* later propagation. However, not all stored values are necessarily propagated. They can get masked by another incoming higher value.

Values stored for later propagation are memorized in a queue structure. This queue behaves like a FIFO, supporting inserting new elements with `push` and retrieving the oldest

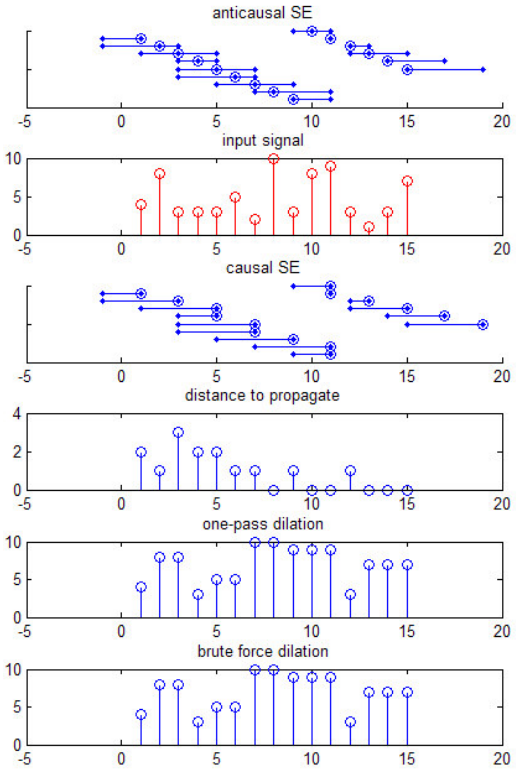


Figure 2: 1-D dilation algorithm: (a) input two-sided structuring element (circle denotes origin), (b) input signal, (c) causal structuring element, (d) distance to propagate the input signal samples, (e) algorithm result, (f) result by definition.

one with `pop`, and like a LIFO retrieving the most recent element with `dequeue`. The structure supports enquiries about the content of the oldest or most recent element `oldest`or `recent`, and about the emptiness of the queue `isempty`. The stored data are triplets *value* (`Fval`), *position* (`rp`) and *distance to propagate* (`dd`).

Refer to the pseudo code in Appendix. The propagation algorithm consits of following stages :

1) Read next signal value Fnext

2) Compute distance to propagate using scheme Eq. 4.

3) Compute the right and left edge (`RE`), (`LE`) of the structuring element at given writing position (`wp`). Note that before outputting a value (at position `wp`), all input data covered by its structuring elements need to be read: (cycle `while 1`), broken by one of breaks as soon as new value can be written.

4) Dequeue all stored smaller values that will never propagate, being masked by Fval.

5) Detect a downstep and store the corresponding triplet for the future propagation.

6) Compare the value currently being propagated with Fval. If Fval is higher, use Fval for propagation. Otherwise, dequeue all smaller values than Fval, then store Fval, its position x and distance d(x).

7) Detect the end of current propagation `TT(2)+TT(3)`<wp, meaning compare whether its initial position+distance is shorter that the current output position wp. If yes, find another value to propagate: erase too old values from the queue (no longer covered by current structuring element), and pop - if exists - next stored value higher that Fval, otherwise, use Fval for propagation. Memorize in TT its value, position and distance to propagate.

Whenever the reading position reaches RE(wp), write at wp of the output dF the maximum of Fval and the currently propagated value T.

3.1 Algorithm Features

Input/Output data are read/written sequentially in a stream, every sample is read/written once and only once.

The algorithm latency (difference between reading and writing position) is at any moment strictly equal to the data dependency.

Data don't need to be allocated in the memory, i.e. there's no memory limitation to size of images.

Algorithm is able to run in place, i.e. input data can be overwritten by output data.

3.2 Computation Complexity and Memory requirements

The algorithm complexity is $O(N)$, N being the amount of data (e.g. the number of pixels). Every sample is read and processed (or dequeued) only once. The complexity is constant with respect to the size of structuring element.

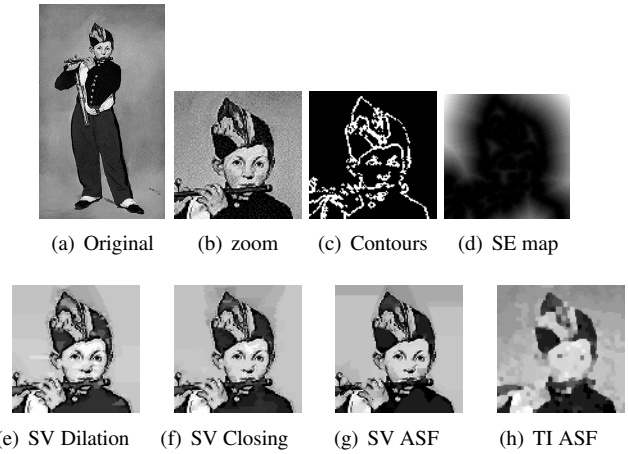
The data can be read and output in a stream, and don't even need to be stored in memory. Therefore, the only memory occupation is given by the maximal depth of the FIFO. The worst case memory occupation of the algorithm is obtained on a non-increasing interval, larger than the structuring element size used at this position.

4. EXPERIMENTAL RESULTS

Consider detail-preserving noise filtering or contour-aware image simplification. We illustrate the performance on the Manet's Le fifre painting, given by Fig. 3a. The contours to preserve (c) can be detected with any usual edge detector (we omit details here). Obviously, the contour preserving filtering must use spatially variant (SV) SE that do not cross these contours. Moreover, techniques based on mathematical morphology using SV SE yield in less iterations better results than with translation invariant (TI) SE.

After decomposition into 1-D, the present algorithm can be used as follows: (e) shows horizontal then vertical dilation by linear SV SE of size locally equal the distance (d) to contours. In this way it *approximates*[1] 2-D dilation with SV squares. The panel (f) gives approximation of closing, and (g) approximation of opening•closing (one stage ASF filter). Compare these results with TI ASF (h), obtained with

[1] unlike TI, unconstrained SV rectangular SE are not separable into 1-D.



(a) Original (b) zoom (c) Contours (d) SE map

(e) SV Dilation (f) SV Closing (g) SV ASF (h) TI ASF

Figure 3: Image simplification application

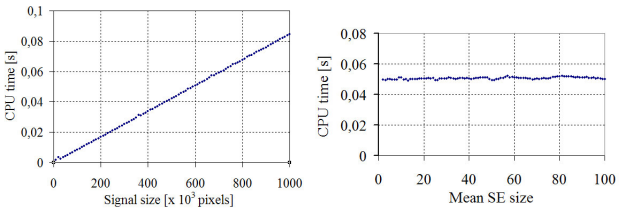
squares 3×3 that, though lesser filtering the background, makes disappear the details from the face.

4.1 Extension to higher dimensions

Although 2-D dilation by a TI rectangles can be implemented by decomposition into two 1-D dilations (horizontal and vertical), separability of SV SE is not straightforward. For the present algorithm, restrictions on the SE map have to be introduced during its generation. Exact formulation of these restrictions is under investigation and will be published separately.

4.2 Benchmarks

Experiments confirm linear execution time with respect to size of data and constant time with respect to mean size of SE, see Fig. 4, obtained on a 2GHz Intel Core 2 CPU, with 800MHz DDR2 RAM.



(a) (b)

Figure 4: Execution time (a) vs. size of data, (b) vs. mean SE size (image 800x800 pixels)

5. CONCLUSION

This paper presents fast, 1-D, spatially variant, grey-scale dilation, obtained in one scan of data. Its complexity is linear with respect to size of data, and constant with size of structuring elements. Its latency is the strict minimum imposed by the operator dependency on input data. The memory requirements are only bounded by the maximal size of structuring element. Input/output data being read/written sequentially, the implementation is independent of size of data. This is extremely interesting for filtering large data with cascade filters, e.g. ASF, under severe time or memory constraints.

These interesting properties make this algorithm useful not only for spatially variant morphological filtering, but also with translation-invariant structuring elements.

In general, this algorithm represents a next step in the effort towards efficient implementation of grey-scale morphology with spatially variant arbitrary-shaped structuring elements.

REFERENCES

[1] N. Bouaynaya, M. Charif-Chefchaouni, and D. Schonfeld. Theoretical foundations of spatially-variant mathematical morphology - part i: Binary images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):823–836, 2008.

[2] N. Bouaynaya, M. Charif-Chefchaouni, and D. Schonfeld. Theoretical foundations of spatially-variant mathematical morphology - part ii: Gray-level images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):837–850, 2008.

[3] U.M. Braga-Neto. Filters by adaptive neighborhood structuring functions. In *Proceedings of International Symposium on Mathematical Morphology ISMM'96*, pages 139–146, 1996.

[4] M. Charif-Chefchaouni and D. Schonfeld. Spatially-variant mathematical morphology. In *Proceedings of the IEEE International Conference on Image Processing, Austin, Texas, USA*, pages 13–16, November 1994.

[5] F. Cheng and A.N. Venetsanopoulos. Adaptive morphological operators, fast algorithms and their applications. *Pattern recognition*, 33:917–933, 2000.

[6] O. Cuisenaire. Locally adaptable mathematical morphology using distance transformations. *Pattern recognition*, 39(3):405–416, march 2006.

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

[8] Hugo Hedberg, Petr Dokladal, and Viktor Öwall. Binary Morphology with Locally Adaptive Structuring Elements: Algorithm and Architecture. *submitted to IEEE Transactions on Image Processing*, 2008.

[9] F. Lemonnier. *Architecture électronique dédiée aux algorithmes rapides de segmentation basés sur la morphologie mathématique*. PhD thesis, School of Mines of Paris, 1995.

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

[11] R. Lerallut, E. Decencière, and F. Meyer. Image filtering using morphological amoebas. *Image Vision Comput*, 25(4):395–404, 2007.

[12] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(7):629–639, 1990.

[13] J. B. T. M. Roerdink and H. J. A. M. Heijmans. Mathematical morphology for structures without translation symmetry. *Signal Processing*, 15(3):271–277, 1988.

[14] J.B.T.M. Roerdink. Group morphology. *Pattern Recognition*, 33(6):877–895, 2000.

[15] Philippe Saint-Marc, Jer-Sen Chen, and Gérard Medioni. Adaptive smoothing: A general tool for early vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):514–529, 1991.

[16] J. Serra. *Image Analysis and Mathematical Morphology*, volume 2. Academic Press, New York, 1988.

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

Appendix: One-Scan Dilation Algorithm pseudo code

The following symbols denote '←' affectation, '=' equal test and '!' logical negation.

Inputs: F - input signal
L, R - structuring element width
Output: dF - output signal

```
; initialization
Q.clear                 ; initialize empty queue
T ← 0;                  ; currently dilated sample
wp ← 1                  ; output writing position
rp ← 0                  ; input reading position
N ← length(F)           ; size of data

Fnext ← F(1);           ; read first sample

for all wp ∈ 1<N,
   RE ← wp + R(wp)      ; right edge of struct. element
   LE ← RE - L(wp)      ; left edge of struct. element
   while 1,
      if rp < RE,
         rp←rp+1 ; increment reading position

         ; compute the distance to propagate
         dd ← d (rp, R+L, R);

         ; read next input sample
         Fval ← Fnext;
         if rp<N,
            Fnext ← F(rp+1);
         else
            Fnext ← 0;
         end;
      end;

      ; dequeue smaller values
      while !Q.isempty and Q.recent<Fval,
         Q.dequeue
      end;

      ; downstep detection,
      if Fnext<Fval,
         ; store its value, position and distance
         if Q.isempty or Q.recent≠ (Fval; rp; dd),
            Q.push (Fval; rp; dd);
         end;
      end;

      ; compare last enqueued value
      if !Q.isempty and Q.recent(2)≤rp,
         ; if it is higher use it for propagation
         if Q.oldest(1) ≥T and rp - Q.oldest(2)...
                  ≤ Q.oldest(3),
            TT ← Q.pop    ; ok, use the enqueued one
            T ← TT(1);
         end;
      end;

      ; detect end of propagation
      if TT(2)+TT(3)<wp,
         T ← Fval;
         ; find new maximum value
         if !Q.isempty and Q.oldest(2)≤rp,
            while !Q.isempty,
               if wp - Q.oldest(2) > Q.oldest(3),
                  ; erase too old samples
                  TT ← Q.pop
               else
                  if Q.recent(1)≥T,
                     TT ← Q.pop
                     T ← TT(1);
                     break while;
                  end;
               end;
            end;
         else
            if !Q.isempty and Q.oldest(2)≤rp,
               TT ← Q.pop;
               T ← TT(1);
            end;
         end;
      end ; end of propagation

      ; write output
      if rp=RE,
         dF(wp) ← max(Fval, T)
         break while;
      end;
   end;
end;
```