Morphological erosions and openings: Fast algorithms based on anchors

M. VAN DROOGENBROECK and M. BUCKLEY

Abstract

Several efficient algorithms for computing erosions and openings have been proposed recently. They improve on VAN HERK's algorithm in terms of number of comparisons for large structuring elements. In this paper we introduce a theoretical framework of *anchors* that aims at a better understanding of the process involved in the computation of erosions and openings. It is shown that the knowledge of opening anchors of a signal f is sufficient to perform both the erosion and the opening of f.

Then we propose an algorithm for one-dimensional erosions and openings which exploits opening anchors. This algorithm improves on the fastest algorithms available in literature by approximately 30% in terms of computation speed, for a range of structuring element sizes and image contents.

1 Introduction

An important class of non-linear image operators compute rank statistics inside a moving window. The median filter for example, is known to be efficient for the removal of *salt-and-pepper* noise. Stack filters are a class of sliding window filters which include all rank-order operators. Local maximum and minimum filters are, in turn, particular kinds of rank-order operators. In mathematical morphology, these operators are referred to as *erosion* and *dilation* respectively, and the window itself is termed a *structuring element* or a *structuring set*. Some authors refer to these operators as *min-* or *max-filters*. Although the window shape might be arbitrary, it is common practice in applied image analysis to use linear, rectangular or circular structuring elements.

Over the last 10-15 years, the tools of mathematical morphology have become part of the mainstream of image analysis and image processing technologies. The growth of popularity is due to the development of powerful techniques, like granulometries [1] and the pattern spectrum analysis [2], that provide insights into shapes, and tools like the watershed [3, 4] or connected operators [5] that segment an image. But part of the acceptance in industrial applications is also due to the discovery of fast algorithms that make mathematical morphology competitive with linear operations in terms of computational speed. A breakthrough in the use of mathematical morphology was reached, in 1995, when morphological operators were adopted for the production of segmentation maps in MPEG-4. Nowadays morphological functions are available in almost any image analysis library.

For morphological and other non-linear filters, a simple transcription of the filter's definition can lead to the worst possible implementation (although such naive and inefficient algorithms do appear in many image analysis packages!). Fortunately many algorithms have been proposed to speed up the computation of morphological operators. These algorithms can be broadly divided into two families:

(1) algorithms that artificially reduce the size of the structuring set or function used to compare a signal with, via an analysis and decomposition of the structuring set or function into smaller pieces, and (2) algorithms that analyse the signal to reduce the number of redundant comparisons inherent to definitions like the morphological erosion.

Our work extends the second family of algorithms and it aims at a better understanding of methods for computing erosions and openings. We show that the concept of *anchors* summarises the fundamental characteristics of a signal with respect to the computation of operators like erosions and openings. We also show that algorithms based on anchors improve significantly on the algorithms available in literature in terms of computation speed.

The remainder of the paper is organised as follows: Section 2 recalls several definitions. Then we review existing algorithms and discuss implementation issues (Section 3). The concept of anchor is introduced in Section 4. In this section we develop a theoretical framework, show how erosions and openings can be entirely computed with the help of anchors, and provide properties related to the use of anchors in practical implementations.

Since morphological operators based on linear structuring elements are essential for many commercial software package, we study this particular case in details in Section 5. Properties specific to linear structuring elements, which are not valid for any general structuring element, are proposed. Then we describe new algorithms. Performances of these algorithms are analysed in Section 6. Finally, Section 7 concludes the paper.

2 Basic theory

We briefly recall the definitions and notations used in this paper. For clarity, we use different notations to distinguish between operators on sets and on functions. But it should be noted that operators on sets can be seen as the application of function operators on binary functions.

2.1 Set operators

Consider a space $\mathcal{E}$, which is the continuous Euclidean space $\mathbb{R}^n$ or the discrete space $\mathbb{Z}^n$, where $n \geq 1$ is an integer. Given a set $X \subseteq \mathcal{E}$ and a vector $b \in \mathcal{E}$, the translate X_b is defined by $X_b = \{x + b | \, x \in X\}$.

Let us take two subsets X and B of $\mathcal{E}$. MINKOWSKI defined the addition and subtraction of these sets, respectively as

$$X \oplus B = \bigcup_{b \in B} X_b = \bigcup_{x \in X} B_x = \{x + b | \, x \in X, \, b \in B\} \tag{1}$$

$$X \ominus B = \bigcap_{b \in B} X_{-b} = \{p \in \mathcal{E} | \, B_p \subseteq X\}. \tag{2}$$

For $X \oplus B$, X and B are interchangeable, but X and B play a different role in the case of $X \ominus B$. Therefore B is referred to as the *structuring element*, and we call $X \oplus B$ and $X \ominus B$ respectively the *dilation* and *erosion* of X by B.

Dilation and erosion are not inverse operators. If X is eroded by B and then dilated by B, one may end up with a smaller set than the original set X. This set, denoted by $X \circ B$, is called the *opening* of X by B and defined by $X \circ B = (X \ominus B) \oplus B$. Likewise the *closing* of X by B is the dilation of X followed by the erosion, both with the same structuring element. The *closing* of X by B may return a set which is larger than X; it is denoted by $X \bullet B$ and defined by $X \bullet B = (X \oplus B) \ominus B$.

Dilations and erosions are closely related. This is expressed in the principle of duality [6] that states that

$$X \ominus B = (X^c \oplus \check{B})^c \quad \text{or} \quad X \oplus B = (X^c \ominus \check{B})^c \tag{3}$$

where the *complement* of X, denoted X^c, is defined as $X^c = \{p \in \mathcal{E} | \, p \notin X\}$, and the *symmetric* or *transposed* set of $B \subseteq \mathcal{E}$ is the set $\check{B}$ defined as $\check{B} = \{-b | \, b \in B\}$. Therefore all statements concerning erosions and openings have a parallel statement for dilations and closings, and vice versa.

2.2 Function operators

In the following we model grey-scale images with functions and we restrict ourselves to real-valued functions defined on $\mathcal{E}$. If f is a function and $b \in \mathcal{E}$, then the spatial translate of f by b is defined by $f_b(x) = f(x - b)$.

Assuming that f is a function and that B is a set, *dilation* and *erosion* are defined respectively by

$$\delta_B(f)(x) = \bigvee_{b \in B} f_b(x) = \bigvee_{b \in B} f(x - b) \tag{4}$$

$$\epsilon_B(f)(x) = \bigwedge_{b \in B} f_{-b}(x) = \bigwedge_{b \in B} f(x + b). \tag{5}$$

The effects of dilations and erosions are well known. If, by convention, we choose to represent large values with white pixels and low values with dark pixels in an image, dilations enlarge white areas whereas erosions enlarge dark areas.

Just as in the binary case, the *morphological opening* $\gamma_B(f)$ and *closing* $\phi_B(f)$ are defined as compositions of erosion and dilation operators:

$$\gamma_B(f) = \delta_B(\epsilon_B(f)) \tag{6}$$

$$\phi_B(f) = \epsilon_B(\delta_B(f)). \tag{7}$$

Figure 1 shows the effects of several morphological operators on a 490×568 image.

Again, $\epsilon_B(f)$ and $\delta_B(f)$ as well as $\gamma_B(f)$ and $\phi_B(f)$ are duals of each other [6]. In addition HEIJMANS and RONSE [7, 8] have developed a general algebraic framework for morphological operators that provides other types of relationships. For example they showed that $(\epsilon_B(f),\, \delta_B(f))$ form an *adjunction*. The adjunction property, which is expressed as $\epsilon_B(f) \geq g \Leftrightarrow f \geq \delta_B(g)$, provides methods to derive erosions from dilations, and vice versa. It constitutes one of the main ingredients of the theory of morphological operators [9].

From a theoretical perspective, an operator is called an (algebraic) opening if it is increasing, anti-extensive, and idempotent. Therefore the family of algebraic openings encompasses morphological openings, but also area openings [10], openings by reconstruction [5], annular openings [11], openings by attribute [12], ... Our purpose in this paper is to focus on morphological openings.

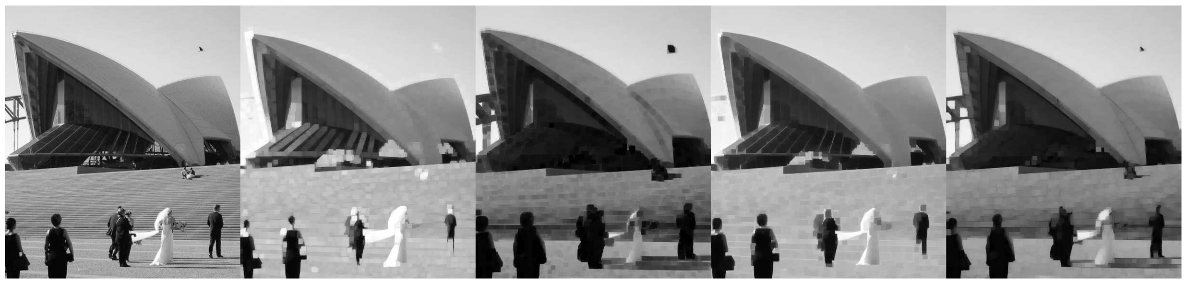


Figure 1: Original image (opera), dilation, erosion, closing and opening with a 21×21 square.

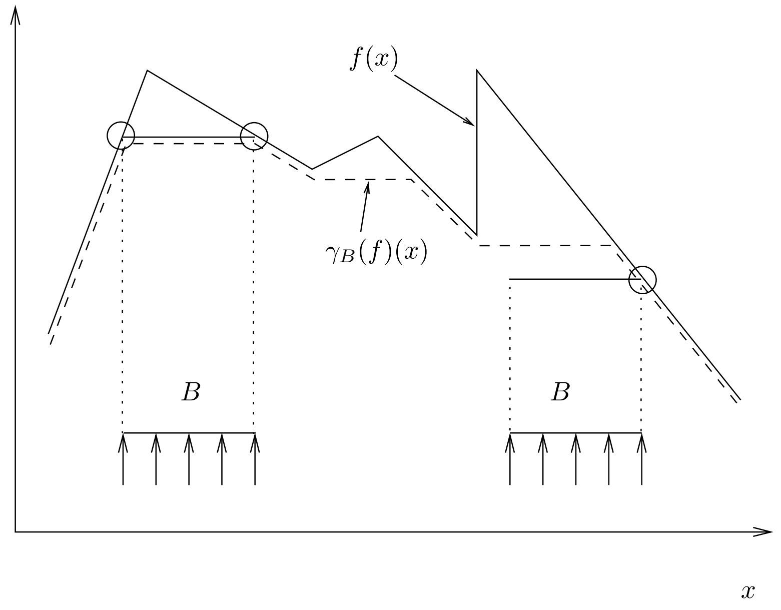


Figure 2: Geometrical definition of an opening by a segment. The opening is the upper envelope when the segment is moved along the axis and pushed as high as possible.

2.3 Geometrical definition of morphological openings

From an implementation point of view, it is interesting to note that there are alternative definitions for morphological operators. Even morphological openings should not necessarily have to be defined as erosions by B followed by dilations with the same B! To show this, let $f(x)$ be a one-dimensional function, and B be a segment. A geometrical definition of $\gamma_B(f)$ runs as follows (see Figure 2): translate B all along the x axis and for each translation, try to push the segment as high as possible below the graph of f. Then the envelope of the highest values reached while moving B is the opening of f by B.

3 Existing algorithms and computational complexity

We consider the simplest two-dimensional opening which is of practical interest: an opening by a rectangular structuring element $B = mH \oplus nV$ where mH, nV respectively are m-wide horizontal and n-wide vertical segments. We suppose that we have to compute $\gamma_B(f)$ for a given image f.

It would be valuable to have a fast algorithm to compute $\gamma_{mH\oplus nV}(f)$, but to our knowledge no algorithm for greyscale openings with rectangles has been proposed so far. We can try to find a satisfactory substitute for $\gamma_{mH\oplus nV}(f)$. First note that an opening with an horizontal segment mH followed by an opening with vertical segment nV does not provide an opening with rectangle $mH \oplus nV$. Also $\gamma_{nV}(\gamma_{mH}(f))$ is not an opening but the supremum $\gamma_{mH}(f) \bigvee \gamma_{nV}(f)$ is an opening. To gain some insight, Figure 3 compares four openings; it illustrates how $\gamma_{mH\oplus nV}(f)$ significantly differ from $\gamma_{mH}(f)$, $\gamma_{nV}(f)$, $\gamma_{mH}(f) \bigvee \gamma_{nV}(f)$, so that there is no satisfactory substitute for $\gamma_{mH\oplus nV}(f)$.

3.1 Review of existing algorithms

A direct implementation of the definition of $\gamma_B(f)$ leads to an algorithm with a high complexity. Indeed, in this most simple approach, the two-part cascade $\gamma_{mH\oplus nV}(f) = \delta_{mH\oplus nV}(\epsilon_{mH\oplus nV}(f))$ is used and the erosion and dilation are each implemented as separate searches for local minima and maxima at each location. The complexity of this algorithm is proportional to $m \times n$, i.e. to the area of the structuring element. A complexity related to the area of the structuring element is infeasible in many practical applications. As a consequence, a number of authors have developed methods to decrease the complexity for larger structuring elements.

Linear decomposition is based on the chain rule (see [13]) that states that $\epsilon_{mH\oplus nV}(f) = \epsilon_{nV}(\epsilon_{mH}(f))$ and that $\delta_{mH\oplus nV}(f) = \delta_{nV}(\delta_{mH}(f))$. The chain rule is essential for the generalisation of algorithms to higher dimensions as it is valid for any structuring element that can be described as the dilation of a set by another set. This is the main reason why 1-D algorithms are so useful for 2-D computations.

With respect to a rectangle, linear decomposition results in the 4-part cascade $\delta_{nV}(\delta_{mH}\ (\epsilon_{nV}\ (\epsilon_{mH}\ (f))))$. If each one-dimensional erosion and dilation is implemented as a separate minimum or maximum calculation at each location, then the complexity of this algorithm is proportional to $m+n$. This is clearly better than $m \times n$ for moderate or large structuring elements.

Further improvements have used linear decomposition but sought more efficient methods for implementation of the elementary 1-D erosions and dilations. The *logarithmic decomposition* as proposed by PECHT [14] and VAN DEN BOOMGAARD [15] removes much of the redundancy involved in repeated erosions or dilations with the same structuring element. This decomposition is based on the general result that $B \oplus B = B \oplus \partial(B)$ where $\partial(B)$ is the set of *border pixels* in B. This applies in any number of dimensions (see [16]) but the logarithmic decomposition was initially used for 1-D structuring elements of $L = 2^k$ pixels. The number of comparisons per pixel for an erosion or a dilation of length L using this decomposition is $k = \log_2 L$. COLTUC and PITAS [17] proposed a more general algorithm for 1-D structuring elements of arbitrary length –that is, not limited to powers of 2.

A quite different approach to the implementation of 1-D erosions and dilations was provided by CHAUDHURI [18] who extended the work of HUANG *et al.* [19]. Both authors based their algorithms on a *local histogram* computed in a sliding window. Other locally-adapting data structures were used by other authors to achieve the same goal; see, for example, PITAS [20], DOUGLAS [21]

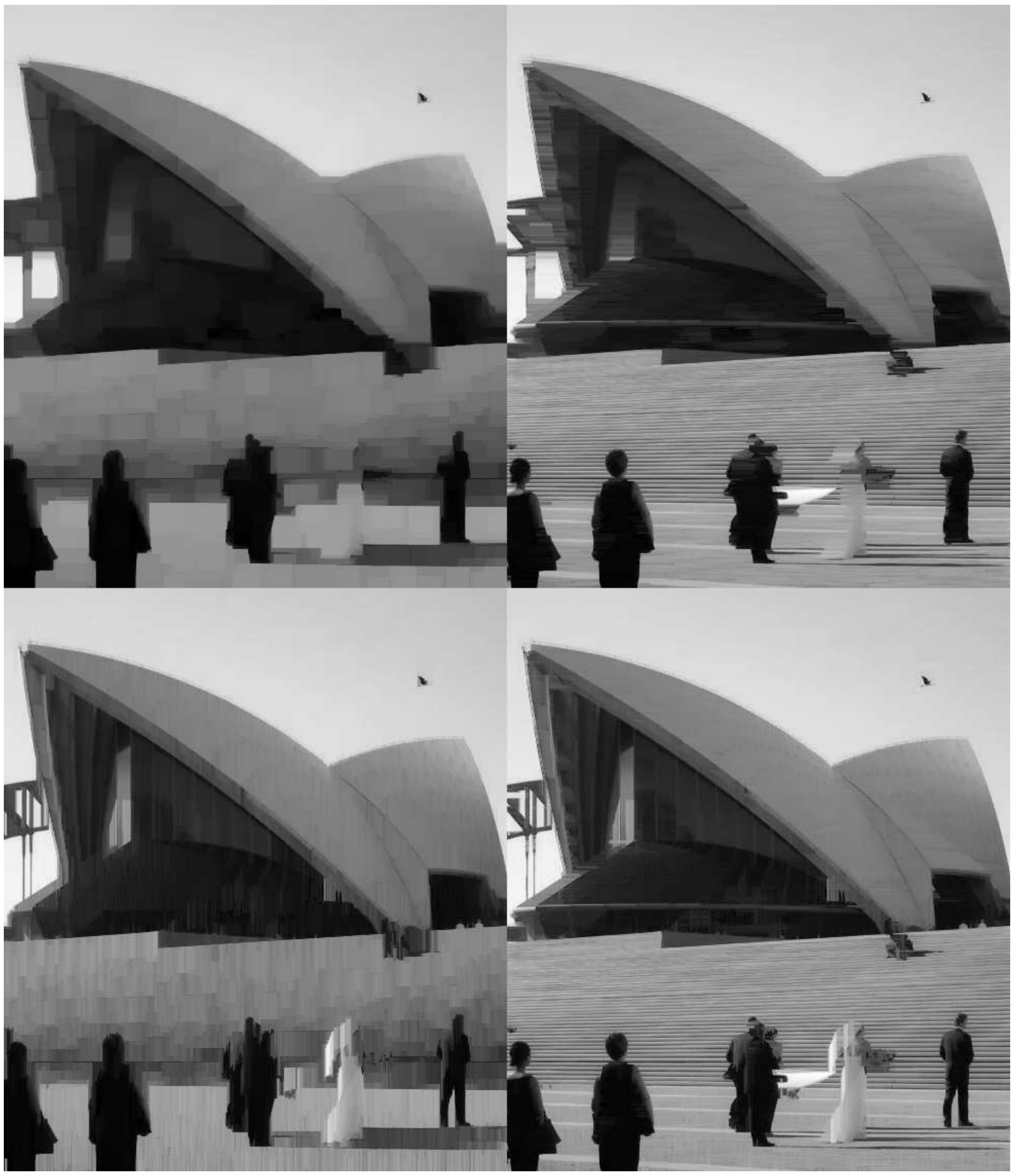


Figure 3: Openings: $\gamma_{mH\oplus nV}(f)$, $\gamma_{mH}(f)$, $\gamma_{nV}(f)$, and $\gamma_{mH}(f)\bigvee\gamma_{nV}(f)$.

and BROOKES [22]. The efficiency of histogram-based and similar algorithms can be good, and nearly independent of L, but typically depends on image content. This may be a problem in some contexts.

In an important advance VAN HERK [23] proposed a one-dimensional algorithm with computational complexity which is largely independent of L. Moreover it has been found that the overall complexity slowly *decreases* with L for the VAN HERK algorithm; see Figures 9 and 10. The erosion form of the algorithm breaks the function into segments whose length matches the size of the structuring element, and computes minima forwards and backwards. A second step combines the results to produce the final result with a fixed cost of three comparisons per pixel. An algorithm with the same complexity but based on circular buffers was developed in a paper by MIYATAKE *et al.* [24]. Several authors have improved VAN HERK's algorithms [25, 26, 27, 28]. In these schemes the number of comparisons is lowered to about 1.5 per pixel. However other overheads are introduced, and the net effect is that these algorithms are very close in performance to VAN HERK's algorithm once implemented.

3.2 Border effects

Up until now we have considered infinite, or at least large, domains for the signal f. In practice, however, we almost always need to deal with limited domains, typically rectangles in two dimensions. When domains are limited edge effects become an important practical consideration. In this section we touch on some of the issues involved. The objectives are threefold: (1) develop a general framework for dealing with border effects, (2) show that caution is needed when an opening is computed as an erosion followed by a dilation, and (3) initiate a further discussion on anchors and border effects.

It is typically the case in image analysis that we can think of the signal or function f as being defined on a domain which is effectively infinite, and we have a translation-invariant operator ψ which is defined on an infinite domain so in principle we know how to compute $\psi(f)$ which also has an infinite domain. But we actually observe

$$f^{(\mathcal{D})} = \mathcal{D}\left(f\right) \tag{8}$$

which is the *restriction* of f to a finite domain $\mathcal{D}$, and we would like, if possible, to compute $\mathcal{D}\left(\psi(f)\right)$. We therefore need to define an operator $\psi^{(\mathcal{D})}$ which operates on functions whose domain is $\mathcal{D}$. Ideally this finite-domain operator would *match* the infinite-domain operator ψ in the sense that

$$\psi^{(\mathcal{D})}(\mathcal{D}\left(f\right)) = \mathcal{D}\left(\psi(f)\right) \tag{9}$$

for all f. We term this property *domain-invariance*.

In general the domain-invariance property is unachievable as values of the function outside the domain affect values of $\psi(f)$ within the domain. However a standard strategy of *extension* can achieve a useful *partial* domain-invariance. This strategy is, for openings and erosions, to assume the function has value $+\infty$ outside the domain. Mathematically, we compute $\psi^{(\mathcal{D})}(f^{(\mathcal{D})})$ as

$$\psi^{(\mathcal{D})}(f^{(\mathcal{D})}) = \mathcal{D}\left(\psi(E^{+\infty}(f^{(\mathcal{D})}))\right) \tag{10}$$

where $E^{+\infty}$ extends the domain of $f^{(\mathcal{D})}$ by giving the value $+\infty$ outside the window. In practice the maximum pixel value is used in place of $+\infty$ as we only require that pixel value used for extension is greater than or equal to all values of $f^{(\mathcal{D})}$.

This strategy achieves partial domain-invariance. Consider a one-dimensional function f observed in a domain $\mathcal{D}$, and let us open this function with a structuring element B which is a segment H of length

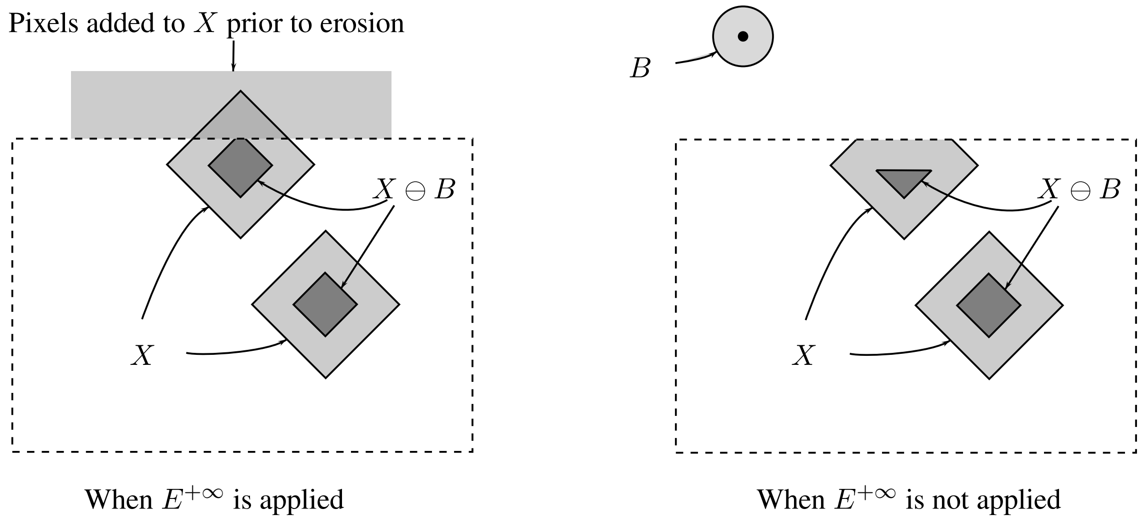

Pixels added to X prior to erosion
B
$X \ominus B$
X
$X \ominus B$
X
When $E^{+\infty}$ is applied
When $E^{+\infty}$ is not applied

Figure 4: Comparison of two strategies for computing $X \ominus B$ at borders.

l. Suppose the actual signal f is non-decreasing in a region of length $2l$, centred on the right-hand edge of $\mathcal{D}$. If ψ is an opening by H, then it can be seen that the domain-invariance property holds within the domain near the right-hand edge. That is $\mathcal{D}\left(\psi(E^{+\infty}(f^{(\mathcal{D})}))\right) = \psi(f)$ in this region. In fact both functions are equal to f in this region. Moreover the same local domain-invariance holds for both non-decreasing and non-increasing edge behaviours, and at both ends of the domain. Similar properties may be obtained in higher dimensions but these are not simple to characterise.

Arbitrary functions are unlikely to meet domain-invariance with respect to morphological operators. Therefore, the strategy to apply $E^{+\infty}$ is commonly adopted, implicitly or explicitly, in computational image analysis, and is usually considered the best strategy overall. Full domain-invariance is not achieved, but as we have noted this can never be achieved. The same strategy also leads to partial domain-invariance for erosion. Figure 4 compares the results of two erosions depending on whether or not $E^{+\infty}$ is applied. For closings and dilations the dual strategy is used; that is, extension with $-\infty$, or zero for non-negative signals.

We now return briefly to openings as cascades. Let $\gamma_B^{(\mathcal{D})}$, $\epsilon_B^{(\mathcal{D})}$ and $\delta_B^{(\mathcal{D})}$ respectively be opening, erosion and dilation operators for finite domain signals $f^{(\mathcal{D})}$ defined via the extension strategy described above. It would be useful if

$$\gamma_B^{(\mathcal{D})}(f) = \delta_B^{(\mathcal{D})}(\epsilon_B^{(\mathcal{D})}(f)) \tag{11}$$

but unfortunately this is *not* the case in one or two dimensions. In other words, although partial domain-invariance is achievable for an opening via the extension strategy, it is not achieved by cascading an extended erosion and an extended dilation. It is shown later in this paper, however, that in the one-dimensional case at least, an extended opening can be computed *directly* –that is, not via an erode-dilate cascade– and that in fact this can be computed more quickly than an erosion!

3.3 Redundancy and memory

Naive implementations of morphological operators fail to be efficient because of *redundancy* of these operators. For example, an erosion by a structuring element whose size is N would require $N-1$ comparisons per pixel should the operator be implemented naively. Trivial implementations are inefficient because they compare neighbouring pixels several times and do not keep intermediate results. Histogram based algorithms, as described in [19, 29], try to keep the information collected in the past in a more useful data structure than an array of values –in this section we refer to this information as *memory*. VAN HERK's algorithm derives from another approach that consists in trying to *propagate* information. In general these techniques are seeking to maximise memory and minimise redundancy. We have found that anchors provide an appropriate theoretical framework for morphological algorithms because they offer insights into the structure of the signal which clarify how much memory is available and how much is needed at any given moment.

3.4 Complexity and performance criteria

Most alternative algorithms have been designed to achieve the lowest number of *comparisons per pixel*. It is clear that at least one comparison per pixel is needed for the computation of an erosion or an opening when the input signal is totally unknown. Some algorithms get near to this bound as we have seen, but other factors than the number of comparisons should be considered as well to measure the overall performance of a software implementation, in particular the number of accesses to memory and how borders of memory blocks filled with data are dealt with: loop checking, sentinels, etc. In this paper we try to bear in mind all aspects of computational complexity, and in the end we compare algorithms empirically –via execution times for standalone C programs running under Linux.

4 Theory of anchors

An important notion used to characterise the behaviour of filters is that of *root* signals. A root signal, sometimes called "fixed point", of an operator is a signal which is invariant to the applications of that operator. That is, $\psi(f) = f$, where f is the function (or *signal*) and ψ is the operator. Root signals for linear operators typically include constant-valued or straight-line signals.

Root signals have been studied in the context of stack filters with a particular emphasis on median filters [30, 31, 32, 33, 34, 35, 36], the underlying question being whether operators drive any input to a root signal. This is called the *convergence property*. As morphological openings are known to be idempotent, i.e. $\psi(\psi(f)) = \psi(f)$, the convergence property is of no particular interest. A comprehensive study on root signals is provided in [37].

To analyse the behaviour of openings and erosions we introduce the concept of *anchors*. We now define this concept.

4.1 Definition of anchors

An anchor is essentially a local version of the root signal notion. An anchor is a single location where a signal f is unaffected by the application of an operator ψ:

Definition 1 *Given a signal f and an operator ψ, a point x in the domain of f is an* anchor *for f with respect to ψ if*

$$\psi(f)(x) = f(x). \tag{12}$$

We define $A(f,\ \psi)$ as the set of anchors for f with respect to ψ. Clearly $A(f,\ \psi)$ is a subset of the domain of f. As will be shown in this paper, anchors play a central role in the computation of erosions and openings.

If ψ is an opening by B, then one can derive from the geometrical interpretation of an opening that the lower bound of a function f is an anchor. If f has a finite domain $\mathcal{D}$ then there is at least one such global minimum, and therefore at least one anchor point. That is,

Theorem 2 *If γ_B is a morphological opening on a finite domain and B is finite, then the set of opening anchors is always non-empty:*

$$A(f,\ \gamma_B) \neq \emptyset. \tag{13}$$

We provide an improved statement on the number of anchors for openings later.

Again based on the geometrical definition of openings (see Figure 2), larger structuring elements are less likely to be pushed high under the envelope of f. Therefore the number of contact points between the structuring element and a function f, like the ones encircled on Figure 2, decreases as the cardinality of B increases. This intuitive result is illustrated in Figure 5 which gives the percentage of opening anchors for two images (*opera*, shown in Figure 1, and a random image) as well as the theoretical lower bound (established later in this paper).

Despite of the existence of opening anchors, there is no similar proposition for erosions; even for finite domains a signal may have no erosion anchors. In some practical cases however, an erosion anchor always exist. Here is an example.

If f is extended outside the domain $\mathcal{D}$ according to our definition ($E^{+\infty}(f^{(\mathcal{D})})$) and if f is lower bounded, then there exists $q \in \mathcal{D}$ such that $f(q) = \bigwedge_{p\in\mathcal{D}} f(p) = \bigwedge_{p\in\mathcal{E}} E^{+\infty}(f^{(\mathcal{D})})$. Let us now assume that B contains the origin ($o \in B$) and therefore $\epsilon_B(f) \leq f$. Since $f(q)$ is a lower bound, $f(q) \leq \epsilon_B(f)(q)$ too. This means that $f(q)$ is an erosion anchor: $\epsilon_B(f)(q) = f(q)$.

On the other hand, if B does not contain the origin, the existence of an erosion anchor can not be guaranteed. For example there is no erosion anchor if f is a one-dimensional V-shaped function and if B is made of two disjoint segments with the origin in the middle. Another example is the one of an increasing function that is eroded by a linear segment whose origin is in the middle. Figure 6 shows a last example of a function whose erosion anchor set is empty.

4.2 Existence of anchors

Although Theorem 2 is a global existence theorem for opening anchors, it assumes the domain of the function f to be finite. We now establish some local existence results for morphological operators. Finiteness of the function domain is no longer required, but we do assume finiteness of the structuring element.

We begin with the erosion operator ϵ_B. By definition

$$\epsilon_B(f)(x) = \bigwedge_{b\in B} f(x+b). \tag{14}$$

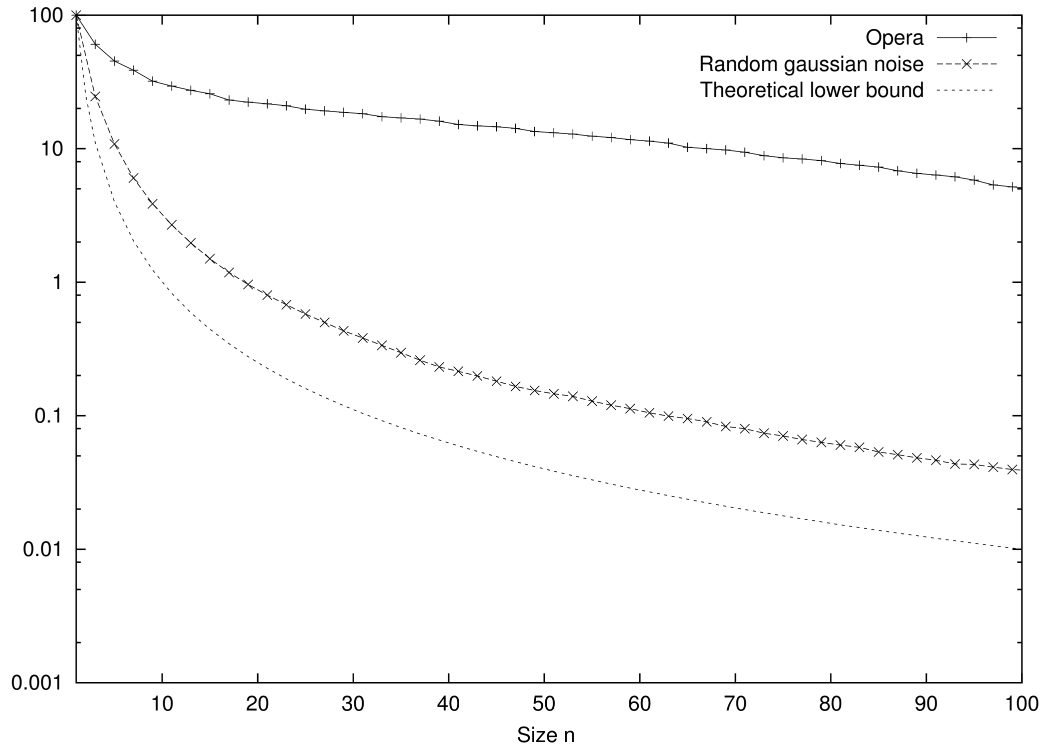


Figure 5: Percentage of opening anchors; B is a $n \times n$ square structuring element.

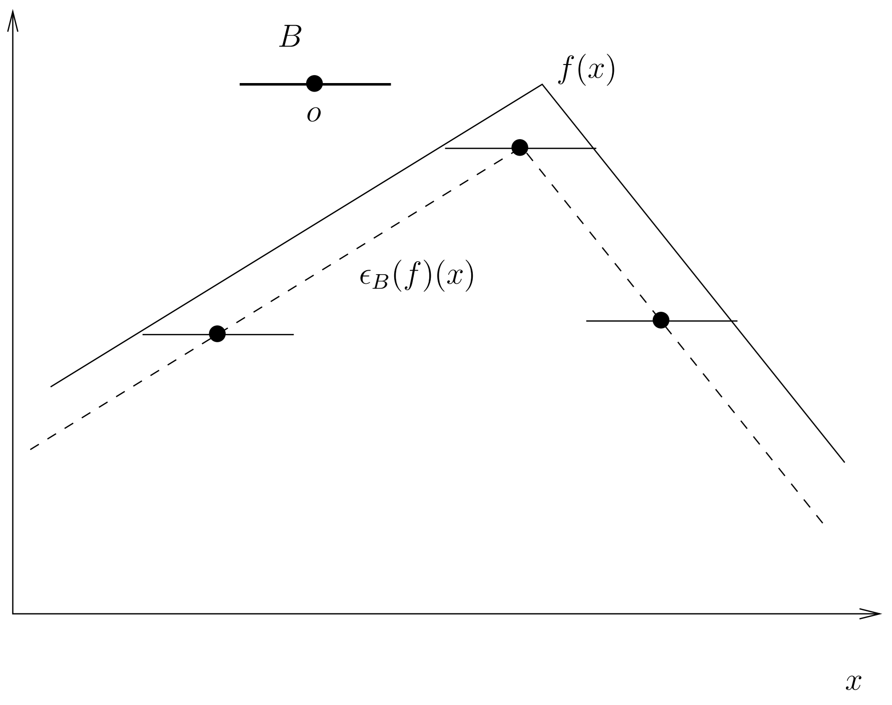


Figure 6: The set of erosion anchors might be empty.

If B is finite then the infimum is achieved at some element b' of B. That is

$$\epsilon_B(f)(x) = f(x + b') \tag{15}$$

for some $b' \in B$. But if $b' \in B$, then $y = x + b' \in B_x$. This leads to

Property 3 *If B is finite and x is any point in $\mathcal{E}$, then*

$$\epsilon_B(f)(x) = f(y) \tag{16}$$

for some $y \in B_x$.

If we consider B_x to define a kind of neighbourhood for x (up to a shift of the origin), then Proposition 3 states that the erosion value at a point x is equal to the function value at some location y in the neighbourhood of x.

For the dilation $\delta_B(f)$ an essentially identical argument leads to

Property 4 *If B is finite and x is any point in $\mathcal{E}$, then*

$$\delta_B(f)(x) = f(y) \tag{17}$$

for some $y \in (\check{B})_x$ where $(\check{B})_x$ denotes the symmetric set $\check{B}$ translated by x.

Note that in the case of a dilation B_x is replaced by $(\check{B})_x$.

Using these propositions we can now prove similar results for openings and closings. We will show that in these cases we need a larger neighbourhood for a point x, namely $(B \oplus \check{B})_x$.

From Equation (6) we have $\gamma_B(f) = \delta_B(\epsilon_B(f))$. Now apply Proposition 4 with f replaced by $\epsilon_B(f)$. Assuming $b_1 \in B$, this yields

$$\gamma_B(f)(x) = \delta_B(\epsilon_B(f))(x) = \epsilon_B(f)(y) \tag{18}$$

for some $y = x - b_1 \in (\check{B})_x$. However from Theorem 3

$$\epsilon_B(f)(y) = f(z) \tag{19}$$

for some $z = y + b_2 \in B_y$, with $b_2 \in B$. It follows that

$$\gamma_B(f)(x) = f(z) \tag{20}$$

with $z = x - b_1 + b_2$ for some $b_1,\, b_2 \in B$. But this implies that $z \in (B \oplus \check{B})_x$ so we have

Theorem 5 *If B is finite and x is any point in $\mathcal{E}$, then*

$$\gamma_B(f)(x) = f(z) \tag{21}$$

for some $z \in (B \oplus \check{B})_x$.

For the closing we have an identical result. Similar results appear in [38] for the discrete case.

There is a fundamental difference in the types of neighbourhood for erosions (Proposition 3) and openings (Theorem 5). We have imposed for B to be finite but not to contain the origin. If B does not contain the origin, the neighbourhood of x for an erosion, i.e. B_x, does not contain x. On the other hand the origin is always an element of $B \oplus \check{B}$ and therefore $x \in (B \oplus \check{B})_x$. This is a supplementary reason why the set of erosion anchors can be empty.

4.3 Opening anchors

We now prove the central theoretical result of this paper, namely that the location z in the foregoing discussion is an opening anchor; that is,

$$\gamma_B(f)(z) = f(z). \tag{22}$$

This establishes the existence, in the neighbourhood $(B \oplus \check{B})_x$ of any point x, of an opening anchor.

To prove that z is an opening anchor we first note that, as constructed, y and z satisfy the following:

$$\gamma_B(f)(x) = \epsilon_B(f)(y) = f(z) \tag{23}$$

with $y \in (\check{B})_x$ and $z \in B_y$. The opening $\gamma_B(f)(z)$ may be written

$$\gamma_B(f)(z) = \bigvee_{s \in (\check{B})_z} \epsilon_B(f)(s). \tag{24}$$

However $z \in B_y$ implies $y \in (\check{B})_z$, and therefore

$$\gamma_B(f)(z) \geq \epsilon_B(f)(y). \tag{25}$$

But $\epsilon_B(f)(y) = f(z)$ so

$$\gamma_B(f)(z) \geq f(z). \tag{26}$$

It is a well-known and fundamental property of openings that they are anti-extensive operators which means that $\gamma_B(f)(z) \leq f(z)$. This proves that $\gamma_B(f)(z) = f(z)$ so z is an opening anchor.

Formally we now have a stronger form of Theorem 5:

Theorem 6 *If B is finite and x is any point in $\mathcal{E}$, then*

$$\gamma_B(f)(x) = \gamma_B(f)(z) = f(z) \tag{27}$$

for some $z \in (B \oplus \check{B})_x$.

A corollary of this important theorem is that all the information needed to compute $\gamma_B(f)$ is contained in its opening anchors. Should an algorithm be able to detect the location of opening anchors and which locations in the neighbourhood they influence in the sense of Theorem 6, it would provide the opening for each x immediately. Unless $f(x)$ has been analysed previously, there is no way to locate anchor points. But with an appropriate scanning order, it is possible to keep some information about f to locate anchor points efficiently. Section 5 describes such an algorithm.

4.4 Links between opening anchors and erosion

The set of erosion anchors may be empty. So we can not rely on erosion anchors to develop an algorithm to compute the erosion. But is there another set which could help computing the erosion?

To answer this question note that $\epsilon_B = \epsilon_B\, \delta_B\, \epsilon_B$ because $(\epsilon_B,\, \delta_B)$ is an adjunction. And since γ_B is defined as $\delta_B\, \epsilon_B$, it follows that $\epsilon_B = \epsilon_B\, \gamma_B$. Therefore, the computation of erosions should be based on opening anchors rather than on erosion anchors.

The following theorem, which extends Proposition 3, establishes a formal link between erosion and opening anchors:

Theorem 7 *If B is finite and x is any point in $\mathcal{E}$, then*

$$\epsilon_B(f)(x) = \gamma_B(f)(y) \tag{28}$$

for some $y \in B_x$. Moreover y is an opening anchor; that is

$$\gamma_B(f)(y) = f(y). \tag{29}$$

The proof is similar to that of Theorem 6. By definition,

$$\epsilon_B(f)(x) = \bigwedge_{y \in B_x} f(y) \tag{30}$$

but as B is finite

$$\epsilon_B(f)(x) = f(y) \tag{31}$$

for some $y \in B_x$. We need to show that this value of y satisfies $\gamma_B(f)(y) = f(y)$. As before, note that $y \in B_x$ implies $x \in (\check{B})_y$. Now

$$\begin{aligned} \gamma_B(f)(y) &= \bigvee_{s \in (\check{B})_y} \epsilon_B(f)(s) && (32) \\ &\geq \epsilon_B(f)(x) && (33) \\ &= f(y). && (34) \end{aligned}$$

As before we use the fact that $\gamma_B(f)(y) \leq f(y)$ for all y. This proves that $\gamma_B(f)(y) = f(y)$ and therefore y is an opening anchor.

4.5 Density of opening anchors

Figure 7 shows empirically how dense opening anchors can be. Theorem 6 leads to a first lower bound: there is at least one opening anchor for each area $(B \oplus \check{B})_x$. But there is a better lower bound.

Intuitively, flat regions contain more anchor points and it is even possible locally for all points to be anchors. In fact Theorem 7 has an interesting side consequence: it provides the following theoretical lower bound for the density of opening anchors:

$$\frac{1}{\#(B)} \tag{35}$$

where $\#(B)$ denotes the cardinality or area of B.

This limit has been plotted on figure 5. It is reachable only if $\mathcal{E}$ can be tiled by translations of B. Where such tiling is not possible, for example, when B is a disk, this bound is conservative.

In addition to a lower bound, what Theorem 7 states is that all the information needed to compute an erosion is contained in the relative positions of x and y, and in the set of opening anchors. This had already been stated, rather informally, in [39]. Interestingly, GIL and KIMMEL have proposed an algorithm for openings in [28] that first performs a modified max-filter algorithm based on VAN HERK's and then feed the results into a min-filter type algorithm. They claim that asymptotically computing the opening filter is not more expensive than computing just the max-filter. Our conclusion is similar although we have started with computing the opening. Note that we do not pretend that opening anchors have to be known to compute the erosion, but if an algorithm for openings gather this information, it will provide the erosion at no additional computation cost.

4.6 Algorithmic properties of anchors

Effective algorithms for erosions and openings intrinsically rely on the ability to find new opening anchors. In VAN HERK's implementation of erosion, local minima values are propagated to the right and to the left, and finally combined. In HUANG's histogram based implementation, the algorithm keeps track of the local minimum of values contained inside a sliding window and it updates the histogram when this window is shifted to the right or to the bottom of the image. Both implementations are efficient because they memorise some statistics when B is moved from position x to position y. More specifically, they keep some statistics (the infimum in the case of an erosion) inside of $B_x \cap B_y$ for computing the infimum over B_y.

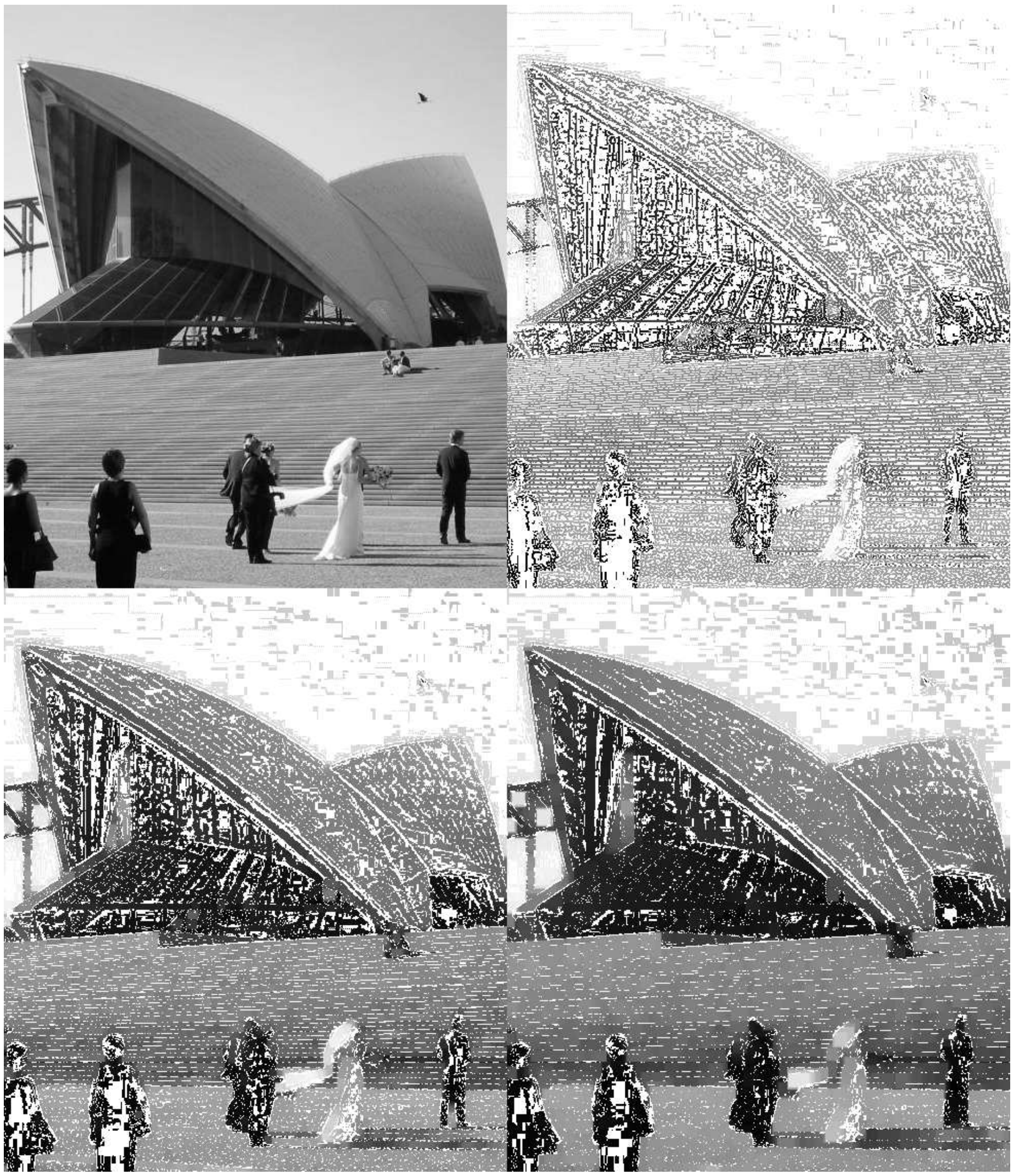


Figure 7: Original and openings with a squared structuring element B (of size 3×3, 11×11, and 21×21 respectively) where opening anchors have been over-written in white.

It appears that any new infimum of $B_y \backslash B_x$ is the value of the erosion $\epsilon_B(f)(y)$, and is also a new opening anchor. Indeed, suppose B is finite and C, D are subparts of B defined such that

$$C_y = B_y \cap B_x \tag{36}$$

and

$$D_y = B_y \backslash B_x. \tag{37}$$

Since $B = C \cup D$,

$$\epsilon_B(f)(y) = \epsilon_{C \cup D}(f)(y) = \epsilon_C(f)(y) \wedge \epsilon_D(f)(y). \tag{38}$$

If a new infimum is found inside D_y, then $\epsilon_C(f)(y) > \epsilon_D(f)(y)$ and consequently $\epsilon_B(f)(y) = \epsilon_D(f)(y)$, which can be seen as if B had been reduced to D. But since Theorem 7 guarantees the existence of an opening anchor inside D_y, there exists some $z \in D_y$ such that $\epsilon_B(f)(y) = \gamma_B(f)(z) = f(z)$.

This is formalised in the following theorem.

Theorem 8 *Suppose B is finite and C, D are subparts of B such that $C_y = B_y \cap B_x$ and $D_y = B_y \backslash B_x$. If*

$$\epsilon_C(f)(y) > \epsilon_D(f)(y) \tag{39}$$

then

$$\epsilon_B(f)(y) = \gamma_B(f)(z) = f(z) \tag{40}$$

for some $z \in D_y$.

The simplest form of this theorem occurs when D is a singleton. In that case, the new infimum (which is inside D) is an opening anchor in addition to be the erosion (for a different location!).

4.6.1 Dealing with borders

If B_x intersects the border (which is defined as the complementary of a domain $\mathcal{D}$ and is denoted $\mathcal{D}^c$), then the infimum of $B_x \backslash \mathcal{D}^c$ is an opening anchor. It results both from Theorem 8 and from our definition of $E^{+\infty}(f^{(\mathcal{D})})(x)$ used to extend functions outside $\mathcal{D}$.

Property 9 *If B is finite and $B_x \backslash \mathcal{D}^c \neq \emptyset$, then*

$$\epsilon_B(E^{+\infty}(f^{(\mathcal{D})}))(x) = \gamma_B(f)(z) = f(z) \tag{41}$$

for some $z \in B_x \backslash \mathcal{D}^c$.

4.6.2 Constrained structuring elements

In practice, the shape of B is not arbitrary. If B is constrained to contain the origin or to be symmetric, we can derive useful properties for implementations.

Suppose for example that $f(x)$ is an erosion anchor and that B contains the origin o. δ_B is then extensive and therefore

$$f(x) = \epsilon_B(f)(x) \leq \delta_B(\epsilon_B(f))(x) = \gamma_B(f)(x). \tag{42}$$

But openings are anti-extensive ($\gamma_B(f) \leq f$) which, combined with the previous equation, implies that $\gamma_B(f)(x) = f(x)$. In other words, *an erosion anchor is always an opening anchor when B contains the origin*!

Theorem 10 *If $o \in B$ and $x \in A(f,\, \epsilon_B)$ then*

$$x \in A(f,\, \gamma_B)\,.$$

Another worthwhile case occurs when B is symmetric up to a translation, i.e. when $B = (\check{B})_p$ for some $p \in \mathcal{E}$. This covers B being a rectangle, a circle, an hexagon, etc. Knowing that openings are invariant to translations of the structuring element [13] yields $\gamma_B = \gamma_{B_p}$. But if $B = (\check{B})_p$ is assumed then $\gamma_B = \gamma_{\check{B}_p} = \gamma_{\check{B}}$. So we have

Theorem 11 *If $\exists p \in \mathcal{E}$ such that $B = (\check{B})_p$, then*

$$A(f,\, \gamma_B) = A(f,\, \gamma_{\check{B}})\,. \tag{43}$$

5 New algorithms for one-dimensional openings and erosions

In previous sections we have defined and developed some results for anchors in a general context. We now consider the special case of one-dimensional (1-D), discrete, equally-spaced data, and develop specific results for this case which will enable us to exploit opening anchors to produce very efficient algorithms for openings and erosions.

Computationally the 1-D case is central for morphological filtering. In the first place 1-D algorithms form the basis for the standard 2-D filters via the separability property. But there are also important direct uses for 1-D morphological operators, notably for detection of locally linear features in 2-D images; see [40] for example.

5.1 Definitions

A discrete 1-D domain may be defined in different ways. Here we imagine the signal f to be defined originally on the whole real line, and then define the discrete-domain function as the restriction of f to the equally-spaced discrete domain

$$\mathcal{D} = \{i(\Delta x) : i \in \mathbb{Z}\} \tag{44}$$

for some fixed $\Delta x > 0$. For convenience we use the notation

$$x_k = x + k(\Delta x) \tag{45}$$

for $x \in \mathcal{D}$. For the value of f at $x \in \mathcal{D}$ we use the notation $f[x]$ to emphasise the discreteness of the restricted function.

Morphological operators in this space will be based on a structuring element H consisting of N consecutive locations in $\mathcal{D}$, with the origin at the first point. That is

$$H = \{0,\, \Delta x,\, 2(\Delta x),\, \ldots,\, (N-1)(\Delta x)\} \tag{46}$$

so that, for $x \in \mathcal{D}$,

$$H_x = \{x,\, x_1,\, x_2,\, \ldots,\, x_{N-1}\}. \tag{47}$$

Note here that there is no need for N to be an odd number as there is for symmetric discrete structuring elements. Also note that the location of the structuring element H with respect to the origin has no effect on the opening by H, and only translates the erosion and dilation by H.

As the structuring element in this case is finite, the infimum and supremum in the definitions (4) and (5) of the dilation and erosion may be replaced by maximum and minimum. Specifically we have, away from the edges of the observation domain,

$$\begin{aligned}
\delta_H(f)[x] &= \max\{f[x_{-N+1}], \cdots, f[x_{-1}], f[x]\} && (48)\\
\epsilon_H(f)[x] &= \min\{f[x], f[x_1], \cdots, f[x_{N-1}]\} && (49)\\
\gamma_H(f) &= \delta_H(\epsilon_H(f)). && (50)
\end{aligned}$$

5.2 Set of local configurations

Suppose x is the location of the last known opening anchor in a left to right row scanning process and y is the location of the last pixel accessed by the algorithm on the right. Essentially, our algorithms will hop from one opening anchor to another opening anchor and compute the opening values in between.

Regardless of implementation issues related to the use of an external memory for intermediate results, we have to consider several cases, drawn in Table 1 and illustrated for $N = 4$. These cases depend on

1. how close x is to y,
2. how the function fluctuates between these two locations, and
3. whether $f[x] < f[y]$ or $f[x] \geq f[y]$.

We now develop a collection of results regarding anchors and the opening function for the 1-D discrete case. These results will enable efficient computations of the opening and erosion operators for a moving window of length N. They will also help us to describe our algorithms.

Configurations 1 and 3. NO LOWER VALUE WITHIN $L < N$ UNITS OF AN OPENING ANCHOR

In the case of Configurations 1 and 3, we are unable as yet to determine the location of any new anchors. The algorithm needs to compare $f[x]$ to values $f[y]$ to the right of y. After further such comparisons, that is, as y is increased, Configuration 1 will result in a configuration either of type 3 or of type 4. Similarly Configuration 3 will result after further comparisons in a configuration of type 4, 5 or 6.

Configuration 2. A DOWNWARD RUN AFTER AN EROSION ANCHOR

We have shown that $\epsilon_H\gamma_H = \epsilon_H$. However there is a tighter relation between $\epsilon_H(f)$ and $\gamma_H(f)$ when $f[x]$ is an erosion anchor. If $f[x]$ is the minimum of all values on the left or on the right, then it is an anchor with respect to the opening γ_H. Formally,

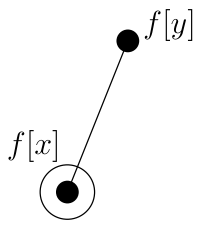
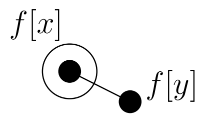
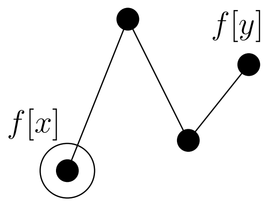
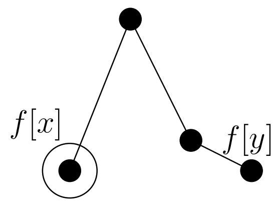
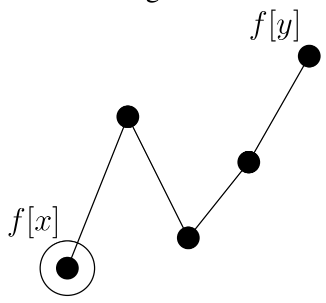
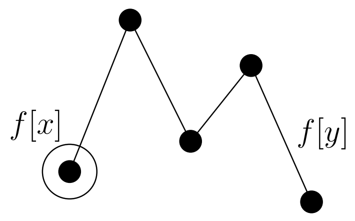
	$f[x] < f[y]$	$f[x] \geq f[y]$
$y = x_1$	Configuration 1 	Configuration 2 
$x < y < x_{N-1}$	Configuration 3 	Configuration 4 
$y = x_N$	Configuration 5 	Configuration 6 

Table 1: Typical neighboring configurations when computing $\gamma_N(f)$ with $N = 4$. The last known opening anchor (i.e. $f[x]$) has been encircled on the drawings.

Property 12 *If*

$$f[x] = \epsilon_H(f)[x] \text{ or } f[x] = \epsilon_{\check{H}}(f)[x]$$

then

$$f[x] = \gamma_H(f)[x] \tag{51}$$

These results follow directly from Theorems 10 and 11.

Suppose that all locations of a sequence are samples of a non-increasing function after an anchor with respect to $\epsilon_{\check{H}}$; that is $f[x] \geq f[x_1] \geq \ldots \geq f[x_J]$ for some integer J and $f[x] = \epsilon_{\check{H}}(f)[x]$. Obviously, $f[x_1], \ldots, f[x_J]$ are all anchors with respect to $\epsilon_{\check{H}}$. This means that these values are also opening anchors.

Configurations 4 and 6. A LOWER VALUE WITHIN N UNITS OF AN EROSION ANCHOR

If there is a lower value of the function f to the right of an erosion anchor x, but within N units, then the first such location, x_L with $L \leq N$, is also an erosion anchor and subsequently an opening anchor. Furthermore, values of the opening at all locations between x and x_L are equal to $f[x]$. The following proposition states this formally.

Property 13 *Assume*

(1) x *is an anchor of* f *with respect to* $\epsilon_{\check{H}}$*,*

(2) $1 \leq L \leq N$*,*

(3) $f[x_L] \leq f[x]$*, and*

(4) $f[x_j] \geq f[x]$ *for* $1 \leq j \leq L-1$.

Then

(5) x_L *is an anchor of* f *with respect to* $\epsilon_{\check{H}}$*, and*

(6) $\gamma_H(f)[x_j] = f[x]$ *for* $1 \leq j \leq L-1$.

Proof
Assumption (1) implies that $f[x_j] \geq f[x]$ for $1-N \leq j \leq -1$, and this together with Assumption (4) gives the stronger statement,

$$f[x_j] \geq f[x] \text{ for } 1-N \leq j \leq L-1. \tag{52}$$

With Assumptions (2) and (3) this implies that $\epsilon_{\check{H}}(f)[x_L] = f[x_L]$ which establishes Conclusion (5). Equation (52) also implies that $\epsilon_{\check{H}}(f)[x_j] = f[x]$ for $1 \leq j \leq L-1$. Moreover the definition of $\epsilon_{\check{H}}$ implies that $\epsilon_{\check{H}}(f)[x_j] \leq f[x_L]$ for $L+1 \leq j \leq L+N-1$.
As $\exists p$ such that $H = (\check{H})_p$, $\gamma_H = \gamma_{\check{H}}$. Now for any integer $1 \leq j \leq L-1$,

$$\begin{aligned}
\gamma_H(f)[x_j] &= \gamma_{\check{H}}(f)[x_j] && (53)\\
&= \delta_{\check{H}}(\epsilon_{\check{H}}(f))[x_j] && (54)\\
&= \max_{j \leq k \leq j+N-1} \epsilon_{\check{H}}(f)[x_k] && (55)\\
&= \max\{T_1, T_2\} && (56)
\end{aligned}$$

where $T_1 = \max_{j \le k \le L-1} \epsilon_{\check{H}}(f)[x_k] = f[x]$ and $T_2 = \max_{L \le k \le j+N-1} \epsilon_{\check{H}}(f)[x_k] = f[x_L] \le f[x]$, and therefore $\gamma_H(f)[x_j] = f[x]$ which is Conclusion (6). ■

The following table summarises conclusions that can be drawn under the assumptions of Proposition 13.

j	$f[x_j]$	$\epsilon_{\check{H}}(f)[x_j]$	$\gamma_H(f)[x_j]$
$1-N \le j \le -1$	$\ge f[x]$	$-$	$-$
$j=0$	$= f[x]$	$= f[x]$	$= f[x]$
$1 \le j \le L-1$	$\ge f[x]$	$= f[x]$	$= f[x]$
$j = L$	$= f[x_L]$	$= f[x_L]$	$= f[x_L]$
$L+1 \le j \le L+N-1$	$-$	$\le f[x_L]$	$-$

Configuration 5. NO LOWER VALUE WITHIN N UNITS OF AN OPENING ANCHOR

A simple application of Theorem 7 tells us that there exists at least one opening anchor inside H_{x_1} and that this anchor is equal to $\epsilon_H(f)[x_1]$. That is

Property 14 *There exists $y \in H_{x_1}$ such that*

$$y \in A(f,\, \gamma_H) \;\; \textit{and} \;\; \epsilon_H(f)[x_1] = f[y]. \tag{57}$$

The algorithm will have to determine an opening anchor, by looking for the minimum of $f[x_1], \ldots, f[x_N]$, in this most unfavourable case (Configuration 5) which occurs when the algorithm does not find an opening anchor just by comparing $f[x]$ to $f[y]$ with $y \in H_{x_1}$. Once a new opening anchor has been identified, the algorithm applies Proposition 13 from right to left to determine values for $\gamma_H(f)[x_1], \ldots, \gamma_H(f)[y]$, and then returns again to one of the other configurations in Table 1.

There are no configurations other than those shown in Table 1. We take this to be evident and do not provide a formal proof.

Table 2 illustrates how the algorithm handles these configurations. The solid line on the drawings shows the opening.

5.2.1 Description of the algorithm for the opening

Appendix A provides simplified C-like pseudo code of our opening algorithm. For clarity, we have used the same notation as in the text and refer to configurations of Table 1 in the code. In addition we have included the code related to the computation of the histogram which is used for finding the minimum of N consecutive values.

Our algorithm will proceed from left to right, so we begin at $x = 0$. Let f, g be a line of the input and output buffers respectively.

$f[0]$ is an anchor with respect to the opening according to Theorem 8. So,

$$g[0] = f[0]. \tag{58}$$

Computationally this means that the first output value $g[0]$ is obtained simply by copying the first input value $f[0]$. More generally all opening anchors could directly be copied into the output buffer.

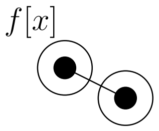
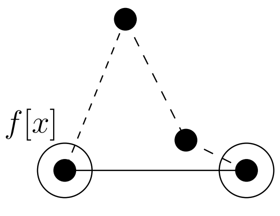
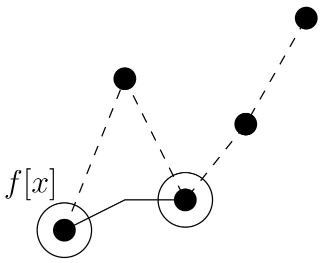
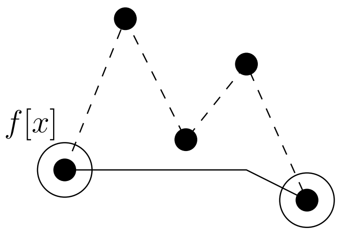
	$f[x] < f[y]$	$f[x] \geq f[y]$
$y = x_1$	Configuration 1	Configuration 2 
$x < y < x_{N-1}$	Configuration 3	Configuration 4 
$y = x_N$	Configuration 5 	Configuration 6 

Table 2: Typical neighboring configurations after computing $\gamma_N(f)$ with $N = 4$. Opening anchors have been encircled.

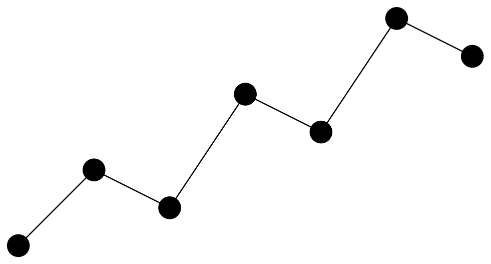


Figure 8: Particular configuration that requires the computation of a minimum every one out of two locations.

Even better it appears that our opening algorithm (not the erosion algorithm!) is capable to work *in situ* so that we can start by copying $f[.]$ into $g[.]$ and skip all opening anchors.

After a global copy, the next step looks for downward runs after an opening anchor. Then the algorithm compares values of $[x+1, x+N]$ to $g[x]$. If there is a value $g[y]$ smaller or equal to $g[x]$, the algorithm has found a new opening anchor. Subsequently it copies $g[x]$ to $g[x+1]$, $\ldots$, $g[y-1]$ (see Proposition 13), and re-enters the loop at label `startLine` .

Configurations 1, 2, 3, 4 and 6 do not require any kind of additional memory. Configuration 5, as shown in Table 1, is the only remaining configuration we need to discuss. Proposition 14 guarantees the existence of an opening anchor, which is the lowest value of $g[x+1]$, $\ldots$, $g[x+N]$. Since this minimum is unknown, the algorithm has to reread all values in order to find the minimum.

Note that on average, the minimum is expected to be in the middle for independent identically distributed input values, but Figure 8 shows an unfortunate case where the minimum has to be found for every one out of two locations.

5.3 Algorithm for erosions

As said previously, we rely on the knowledge of opening anchors to compute erosions. More specifically, as $\epsilon_B(f) = \epsilon_B(\gamma_B(f))$, it is equivalent to process f or $\gamma_B(f)$. In order to implement the erosion instead of the opening, we just have to review the 6 configurations. Again Configurations 1 and 3 are ignored because they end in one of the other remaining configurations. Configurations 2, 4 and 6 are treated as for the opening except that the minimum is allocated to locations delayed by $N/2$ with respect to y. The same principle yields for Configuration 5 in the sense that values are allocated with a spatial delay of $N/2$ pixels. Therefore it is not possible to work *in situ*, and a small increase in computation time is observed.

6 Performance

We have experienced that with the anchors based approach, computation efficiency mainly depends on how Configurations 1, 3 and 5 are implemented. In other words, performance relies on the speed to locate the next minimum of an increasing function.

A histogram was used for the purpose of finding the minimum (initialised to $f[y]$) but only for Configuration 5. It is important not to compute the histogram unless Configuration 5 is reached, and to drop the histogram when a new anchor is found on the right. Hereafter, our algorithms are referred to as `ErosionByAnchors` and `OpeningByAnchors`.

6.1 Implementation

The implementation has the following characteristics:

- All algorithms have been implemented in the C language and it uses pointer arithmetic.
- Images are defined on 8 bits greyscale images which implies that histogram indexes range between 0 and 255. Note that for larger greyscale ranges, it would be more efficient to use a list or a queue structure, like in the MAXLIST algorithm for example (see [21] for a description of the MAXLIST algorithm), than a histogram.
- Our algorithm for openings works *in situ*. Therefore the input buffer is copied into the output buffer (using the C `memcpy` function) prior to any calculation. This technique reduces the number of accesses to memory since opening anchors do not have to be accessed individually in the output buffer.

6.2 Computation times

For the sake of comparison, we have implemented algorithms described by other authors as well. We have chosen to implement the algorithms proposed by VAN HERK [23] and GIL-KIMMEL [28] for erosions. All algorithms were compiled with `gcc` and the `-O2` optimisation flag. Computation times were measured with a profiler named `gprof` which calculates the amount of time spent in each routine. The run-time figures given by `gprof` are based on a sampling process, so they are subject to statistical inaccuracy. In our experiments the sampling period was $0,01$ second and the total run-time was about $1,5$ second for 4000×5000 images. Since an opening uses on average a continuous 80% portion of the total running time and since we draw the time spent in the opening routine plus the time spent in routines called by the opening routine, the maximum sampling inaccuracy error is twice $0,01$ second, less than 2% thus. To reduce it even further we have averaged the results over 10 independent runs.

Figures 9 and 10 show the measured computation times for 20 millions pixel large images filled with the original opera image or with random noise.

The number of occurrences of each configuration for the same two images, expressed as percentages, are drawn in Figure 11 and Figure 12 respectively.

Our experiments show that:

- In terms of computation speed, algorithms for erosions and openings by anchors always perform significantly better than VAN HERK's algorithm (except for $N = 3$ in the case of random noise) and than GIL-KIMMEL's algorithm. We observe a 30% gain when $N = 10$.
- The comparison of Figures 9 and 10 confirms that our algorithms are content-dependent. Moreover they are more effective for a natural image than for a random image. This is related to the number of occurrences of Configuration 2 that drops dramatically for a random image (compare

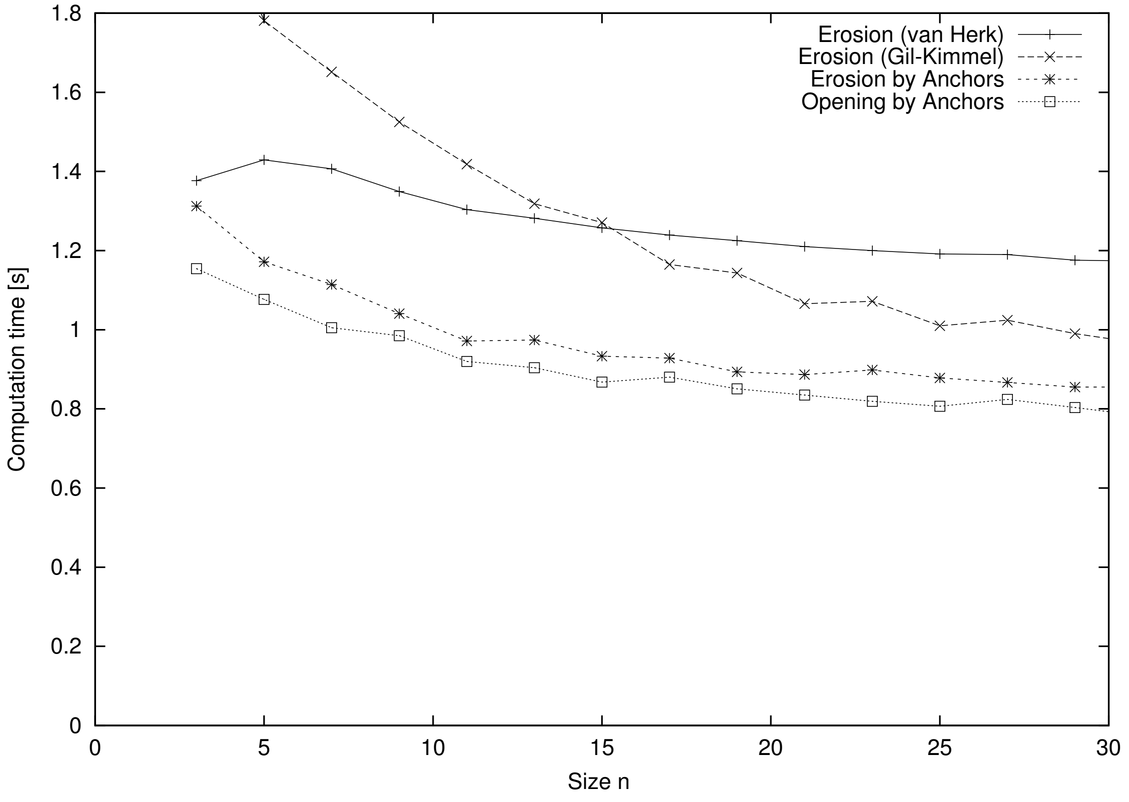


Figure 9: Computation times on a natural image (opera).

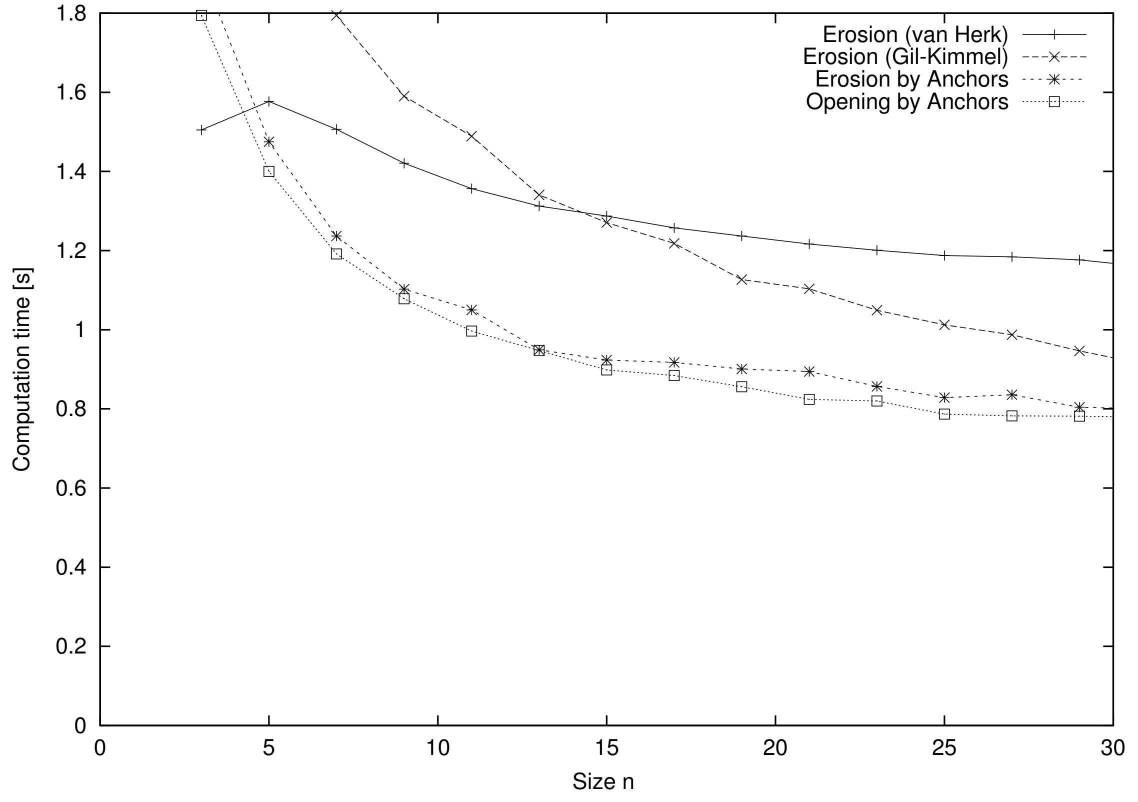


Figure 10: Computation times on a random image.

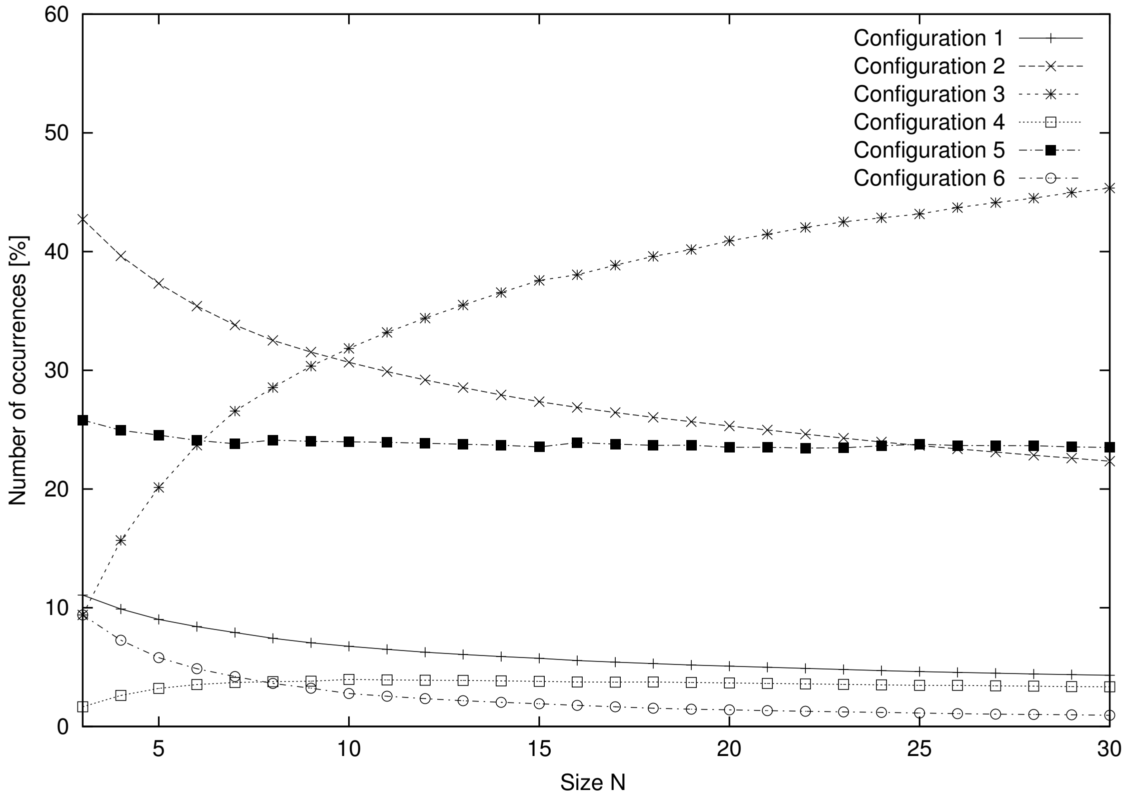


Figure 11: Number of occurrences for each configuration for a natural image (opera).

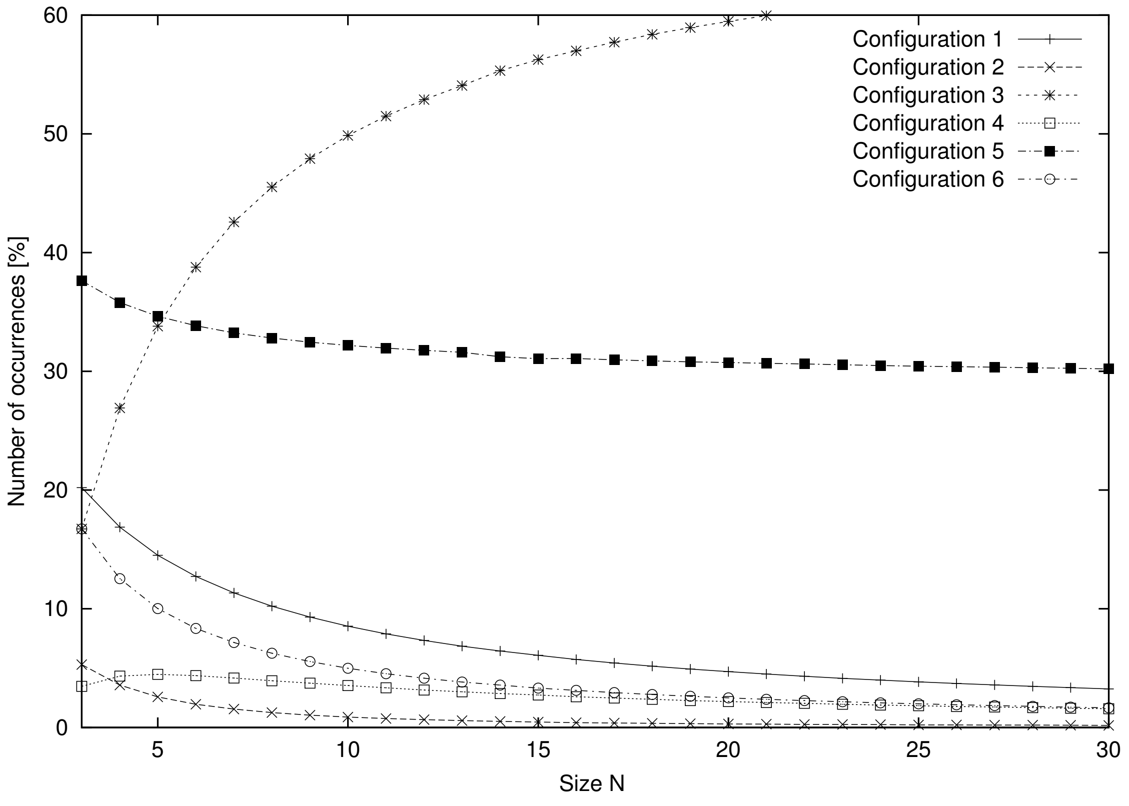


Figure 12: Number of occurrences for each configuration for a random image.

Figures 11 and 12). Indeed, a decreasing slope is handled very effectively as a new anchor is found at the cost of only a single comparison. However our algorithms are to some extent less effective on an increasing function. If the steps of an increasing function are typically less then 5 greyscale, experiments have shown that our performance is at least as good as for a random image. Computation times for increasing functions with larger steps (around 10) are not as good, and similar to the times of the erosion algorithm of GIL-KIMMEL. However a function with even larger steps is processed efficiently in our algorithm as Configurations 2, 4 and 6 become significantly more probable when steps are larger. On an increasing slope the algorithm relies on the use of a histogram to compute the minimum over H_x. Other methods based on queues or lists could possibly further improve the performances of our algorithms.

- Computation times are higher for small structuring elements. There are two explanations for this:

    1. Computation times are higher for small structuring elements because of the inefficiency of histograms to track a minimum on small windows. Indeed when $N = 3$, two comparisons suffice to find the minimum; achieving the same with a histogram requires more comparisons on average when the signal is random.
    2. Figure 5 and equation (35) illustrate that opening anchors are less numerous when the size of B increases. Interestingly computation times decrease as well for increasing B. It may seem counter-intuitive that lower densities lead to reduced computation times. But the point is that we do not know the opening anchor before we have computed $\gamma_B(f)$! There is thus a balance between the difficulty to find opening anchors and the algorithmic usage of found anchors. Apparently this balance favours low anchor densities.

6.3 Note on granulometries

Considering the computation time spent in finding opening anchors, it is worth analysing if the computation of granulometries, i.e. openings ordered by a size parameter, could benefit from the existence of opening anchors. In [41], VINCENT developed the concept of *opening trees* with respect to one-dimensional structuring elements. Our approach is similar and we now show that anchor sets are ordered for granulometries.

Suppose B contains A and B is A-open, i.e. $A \subseteq B$ and $B \circ A = B$, then [9]

$$\gamma_B(f) \leq \gamma_A(f). \tag{59}$$

If x is an anchor with respect to γ_B, we have $\gamma_B(f)(x) = f(x)$. But since openings are anti-extensive

$$\gamma_A(f)(x) \leq f(x) \tag{60}$$

and therefore

$$\gamma_A(f)(x) \leq f(x) = \gamma_B(f)(x). \tag{61}$$

Combined with equation (59), this provides

$$\gamma_A(f)(x) = f(x) = \gamma_B(f)(x) \tag{62}$$

so that x is an anchor for γ_A as well. Formally, we have the following theorem:

Theorem 15 *For any function f, if $A \subseteq B$, $B \circ A = B$, and A, B are both finite, then*

$$A(f,\ \gamma_B) \subseteq A(f,\ \gamma_A) \tag{63}$$

This theorem is essential for granulometries. It tells us that if we order a family of morphological openings, anchor sets will be ordered (reversely) as well.

7 Conclusions

In this paper we have introduced the concept of anchors for operators on functions, and showed the anchor locations for the morphological opening operator, in particular, have significance both theoretically and algorithmically. In the one-dimensional setting we have developed an algorithm based on opening anchors which computes an erosion approximately 30% faster than the best current methods. Moreover a modified algorithm computes the opening even faster. We believe that this algorithm, besides being the fastest currently available, is interesting theoretically as it is based on the novel analytical concept of the opening anchor.

References

[1] G. Matheron, *Random sets and integral geometry*, Wiley, New York, 1975.

[2] P. Maragos, "Pattern spectrum and multiscale shape representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 701–716, July 1989.

[3] S. Beucher and C. Lantuéjoul, "Use of watersheds in contour detection," in *International Workshop on Image Processing*, Rennes, September 1979, pp. 2.1–2.12, CCETT/IRISA.

[4] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, June 1991.

[5] P. Salembier and J. Serra, "Flat zones filtering, connected operators, and filters by reconstruction," *IEEE Transactions on Image Processing*, vol. 4, no. 8, pp. 1153–1160, August 1995.

[6] J. Serra, *Image analysis and mathematical morphology*, Academic Press, London, 1982.

[7] H. Heijmans and C. Ronse, "The algebraic basis of mathematical morphology: I. Dilations and erosions," *Computer Vision, Graphics, and Image Processing*, vol. 50, pp. 245–295, 1990.

[8] C. Ronse and H. Heijmans, "The algebraic basis of mathematical morphology: II. Openings and closings," *Computer Vision, Graphics, and Image Processing: Image Understanding*, vol. 54, no. 1, pp. 74–97, 1991.

[9] H. Heijmans, *Morphological image operators*, Advances in Electronics and Electron Physics. Academic Press, 1994.

[10] L. Vincent, "Morphological area openings and closings for greyscale images," in *Proc. Shape in Picture '92, NATO Workshop*, Driebergen, The Netherlands, September 1992, Springer-Verlag.

[11] C. Ronse and H. Heijmans, “A lattice-theoretical framework for annular filters in morphological image processing,” *Applicable Analysis in Engineering, Communication, and Computing*, vol. 9, no. 1, pp. 45–89, 1998.

[12] E. J. Breen and R. Jones, “Attribute openings, thinnings, and granulometries,” *Computer Vision and Image Understanding*, vol. 64, no. 3, pp. 377–389, 1996.

[13] R. Haralick, S. Sternberg, and X. Zhuang, “Image analysis using mathematical morphology,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp. 532–550, July 1987.

[14] J. Pecht, “Speeding up successive Minkowski operations,” *Pattern Recognition Letters*, vol. 3, no. 2, pp. 113–117, 1985.

[15] R. van den Boomgaard, *Mathematical morphology: Extensions towards computer vision*, Ph.D. thesis, Amsterdam University, March 1992.

[16] M. Van Droogenbroeck, *Traitement d’images numériques au moyen d’algorithmes utilisant la morphologie mathématique et la notion d’objet : application au codage*, Ph.D. thesis, Catholic University of Louvain, May 1994.

[17] D. Coltuc and I. Pitas, “On fast running max-min filtering,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 8, pp. 660–663, August 1997.

[18] B. Chaudhuri, “An efficient algorithm for running window pel gray level ranking in 2-D images,” *Pattern Recognition Letters*, vol. 11, no. 2, pp. 77–80, February 1990.

[19] T. Huang, G. Yang, and G. Tang, “A fast two-dimensional median filtering algorithm,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 27, no. 1, pp. 13–18, February 1979.

[20] I. Pitas, “Fast algorithms for running ordering and max/min calculations,” *IEEE Transactions on Circuits and Systems*, vol. 36, no. 6, pp. 795–804, June 1989.

[21] S. Douglas, “Running max/min calculation using a pruned ordered list,” *IEEE Transactions on Signal Processing*, vol. 44, no. 11, pp. 2872–2877, November 1996.

[22] M. Brookes, “Algorithms for max and min filters with improved worst-case performance,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 9, pp. 930–935, September 2000.

[23] M. van Herk, “A fast algorithm for local minimum and maximum filters on rectangular and octogonal kernels,” *Pattern Recognition Letters*, vol. 13, no. 7, pp. 517–521, July 1992.

[24] T. Miyatake, M. Ejiri, and H. Matsushima, “A fast algorithm for maximum-minimum image filtering,” *Systems and Computers in Japan*, vol. 27, no. 13, pp. 74–85, 1996.

[25] D. Gevorkian, J. Astola, and S. Atourian, “Improving Gil-Werman algorithm for running min and max filters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 526–529, May 1997.

[26] J. Gil and M. Werman, “Computing 2-D min, median, and max filters,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 5, pp. 504–507, May 1993.

[27] J. Gil and R. Kimmel, "Efficient dilation, erosion, opening and closing algorithms," in *Mathematical Morphology and its Applications to Image and Signal Processing V*, J. Goutsias, L. Vincent, and D. Bloomberg, Eds., Palo-Alto, USA, June 2000, pp. 301–310, Kluwer Academic Publishers.

[28] J. Gil and R. Kimmel, "Efficient dilation, erosion, opening, and closing algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 12, pp. 1606–1617, December 2002.

[29] M. Van Droogenbroeck and H. Talbot, "Fast computation of morphological operations with arbitrary structuring elements," *Pattern Recognition Letters*, vol. 17, no. 14, pp. 1451–1460, 1996.

[30] G. Arce and N. Gallagher, "State description for the root-signal set of median filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 30, no. 6, pp. 894–902, December 1982.

[31] G. Arce and M. McLoughlin, "Theoretical analysis of the max/median filter," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 1, pp. 60–69, January 1987.

[32] J. Astola, P. Heinonen, and Y. Neuvo, "On root structures of median and median-type filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 8, pp. 1199–1201, August 1987.

[33] D. Eberly, H. Longbotham, and J. Aragon, "Complete classification of roots to one-dimensional median and rank-order filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 39, no. 1, pp. 197–200, January 1991.

[34] U. Eckhardt, "Root images of median filters," *Journal of Mathematical Imaging and Vision*, vol. 19, no. 1, pp. 63–70, July 2003.

[35] J. Fitch, E. Coyle, and N. Gallagher, "Threshold decomposition of multidimensional ranked order operations," *IEEE Transactions on Circuits and Systems*, vol. 32, pp. 445–450, May 1985.

[36] N. Gallagher and G. Wise, "A theoretical analysis of the properties of median filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 29, no. 6, pp. 1136–1141, December 1981.

[37] P. Maragos and R. Schafer, "Morphological filters—Part II: Their relations to median, order-statistic, and stack filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 8, pp. 1170–1184, August 1987.

[38] J. Barrera and G. P. Salas, "Set operations on closed intervals and their applications to the automatic programming of morphological machines," *Electronic Imaging*, vol. 5, no. 3, pp. 335–352, July 1996.

[39] M. Van Droogenbroeck, "On the implementation of morphological operations," in *Mathematical morphology and its applications to image processing*, J. Serra and P. Soille, Eds., pp. 241–248. Kluwer Academic Publishers, Dordrecht, 1994.

[40] P. Soille and H. Talbot, "Directional morphological filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1313–1329, November 2001.

[41] L. Vincent, “Granulometries and opening trees,” *Fundamenta Informaticae*, vol. 41, no. 1–2, pp. 57–90, 2000.

A Opening algorithm in C-like pseudo code

```
// f[.] is the input signal
// histo[.] stores the histogram

// Copy entire f[.] into g[.]


// First point (=x) and last point (=rightborder) are opening anchors

// g[x] is an opening anchor
// Initialisation
startLine:          // A label
y ← x+1;

// Function values on the right of x which are
// non-increasing are all opening anchors
while ( (y<rightBorder) && (g[y] ≤ g[x]) )
        { x ← x+1; y ← y+1; }      /* Configuration 2 */

// Now we have g[y] > g[x]    /* Configuration 1 */
// Analysis of window [x, x+N[
y ← y+1;
while ( y<x+N )
        {
        if (g[y] ≤ g[x])  // A new minimum has been found
                        /* Configuration 4 */
                {
                // Fills the interval [x+1, y[
                minimum ← g[x];
                x ← x+1;
                while (x < y) { g[x] ← minimum; x ← x+1; }
                goto startLine;
                }
        y ← y+1;          /* Configuration 3 */
        }
```

```
// No new minimum has been found and y=x+N
if (g[y] ≤g[x]) /* Configuration 6 */
        {
        minimum ← g[x];
        x ← x+1;
        while (x < y) { g[x] ← minimum;  x ← x+1; }
        goto startLine;
        }
// Computing the histogram has become unavoidable
/* Configuration 5 */
// Resets the histogram and computes the histogram over [x+1, y]
x ← x+1;
memset(histo, 0, ...); // Fills the histogram with 0
aux ← x;
while (aux ≤y) { histo[g[aux]] ←histo[g[aux]]+1 ; aux←aux+1; }
// Finds and allocates the minimum
minimum ← g[x-1]+1;
while (histo[minimum] ≤0) { minimum ←minimum+1; }
g[x] ←minimum;

// Moves [x,y] to the right, updates the histogram, and
// finds the minimum
while (y < rightBorder)
        {
        y ← y+1;
        if (g[y] ≤ minimum)  // A new minimum has been found
                              /* Configuration 6 */
                {
                x ← x+1;
                while (x < y) { g[x] ←minimum;  x ← x+1; }
                goto startLine;
                }
        // Updates the histogram and recomputes the minimum
        /* Configuration 5 */
        histo[g[x]] ← histo[g[x]]-1;
        histo[g[y]] ← histo[g[y]]+1;
        while (histo[minimum ]≤0) { minimum ←minimum+1; }
        x ← x+1;
        histo[g[x]] ← histo[g[x]]-1;
        g[x] ←minimum;
        histo[minimum] ← histo[minimum]+1 ;
        }

finishLine:
...
```