Algorithms for mathematical morphology

M. Van Droogenbroeck

October 8, 2002

Outline

- Introduction
- Review of existing algorithms and discussion
- Description of two recent algorithms
- Comparison
- Granulometries

Introduction

- Do we really need new algorithms for openings?
- Are existing algorithms not fast enough?

Introduction

- Do we really need new algorithms for openings?
- Are existing algorithms not fast enough?
- Applications
  - Industrial applications
  - Label images
- Requirements
  - Hardware or software?
  - Memory requirements?
  - Is speed the ultimate goal?
  - Should the implementation be reusable?

Definitions and interpretations

- The erosion $X \ominus B$ and dilation $X \oplus B$ of a set X are defined as:

$$X \ominus B = \bigcap_{b \in B} X_{-b} \tag{1}$$

$$X \oplus B = \bigcup_{b \in B} X_b \tag{2}$$

For a function f

$$f \ominus B = \bigwedge_{b \in B} f_{-b} \qquad f \oplus B = \bigvee_{b \in B} f_b \tag{3}$$

- The opening

$$X \circ B = (X \ominus B) \oplus B \tag{4}$$

- Geometric interpretation:

$$X = \bigcup \{B_p \,|\, B_p \subseteq X\} \tag{5}$$

Notations and conventions

- Size of a structuring element: a set B of size n, denoted nB, is usually defined as

$$nB = \underbrace{B \oplus B \oplus \ldots \oplus B}_{n-1 \text{ dilations}} \tag{6}$$

Notations and conventions

- Size of a structuring element: a set B of size n, denoted nB, is usually defined as

$$nB = \underbrace{B \oplus B \oplus \ldots \oplus B}_{n-1 \text{ dilations}} \tag{6}$$

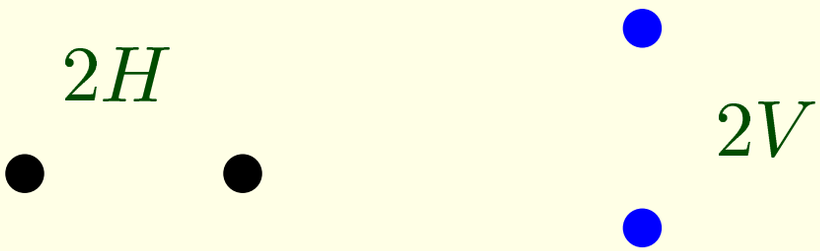
- Here we adopt a different convention!
  - nH is a n-pixels wide segment.

Notations and conventions

- Size of a structuring element: a set B of size n, denoted nB, is usually defined as

$$nB = \underbrace{B \oplus B \oplus \ldots \oplus B}_{n-1 \text{ dilations}} \tag{6}$$

- Here we adopt a different convention!
  - nH is a n-pixels wide segment.
- B is a square or a rectangle.
- H is an horizontal segment and V is a vertical segment



Review of existing algorithms

Problem statement:

- Suppose $B = nH \oplus mV$, we need to compute $X \circ (nH \oplus mV)$.

Review of existing algorithms

Problem statement:

- Suppose $B = nH \oplus mV$, we need to compute $X \circ (nH \oplus mV)$.

Classification of different algorithms?

- Elementary implementations
- Histogram based
- VAN HERK's algorithm
- Algorithms dedicated to binary images
- Distance based algorithms
- Algorithms based on the geometric interpretation of openings

Elementary implementations

1. Trivial implementation

   (a) direct application of the definition $\Rightarrow$ complexity proportional to $n \times m$

Elementary implementations

1. Trivial implementation

   (a) direct application of the definition $\Rightarrow$ complexity proportional to $n \times m$

2. Alternative: linear decomposition.

   (a) based on the chain rule $X \ominus (nH \oplus mV) = (X \ominus nH) \ominus mV$
   (b) results in $(((X \ominus nH) \ominus mV) \oplus nH) \oplus mV$.
   (c) complexity proportional to $n + m$.

Elementary implementations

1. Trivial implementation

   (a) direct application of the definition $\Rightarrow$ complexity proportional to $n \times m$

2. Alternative: linear decomposition.

   (a) based on the chain rule $X \ominus (nH \oplus mV) = (X \ominus nH) \ominus mV$
   (b) results in $(((X \ominus nH) \ominus mV) \oplus nH) \oplus mV$.
   (c) complexity proportional to $n + m$.

3. Other alternative: logarithmic decomposition

   (a) Proposed by PECHT (1985) and VAN DEN BOOMGAARD (1992)
   (b) The logarithmic decomposition suppresses the redundancy inherent to recursive dilations with a unique convex structuring element.

Logarithmic decomposition

For example, if $\partial(B)$ denotes the border of B,

$$9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(2B) \oplus \partial(4B) \tag{7}$$

(usual notation of size)

Logarithmic decomposition

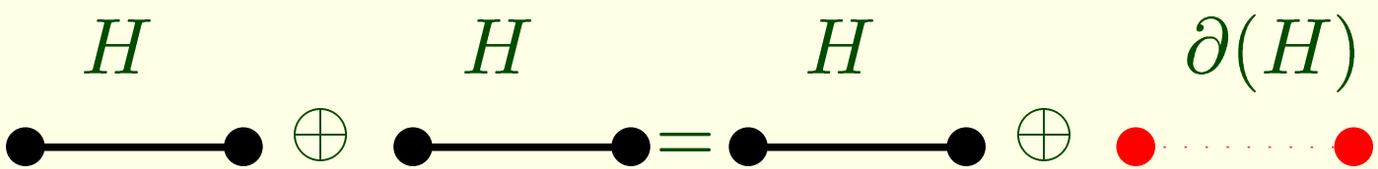
For example, if $\partial(B)$ denotes the border of B,

$$9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(2B) \oplus \partial(4B) \tag{7}$$

(usual notation of size)

Theorem.

$$B \oplus B = B \oplus \partial(B) \tag{8}$$



Logarithmic decomposition

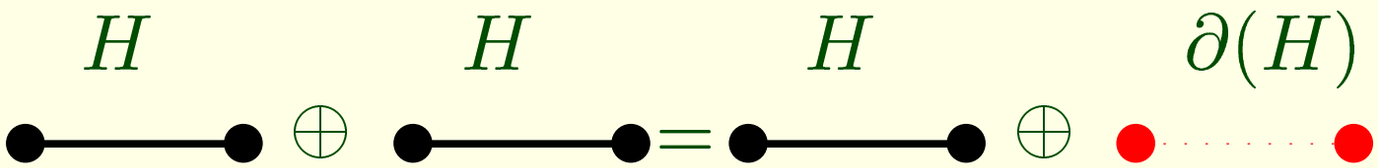
For example, if $\partial(B)$ denotes the border of B,

$$9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(2B) \oplus \partial(4B) \tag{7}$$

(usual notation of size)

Theorem.

$$B \oplus B = B \oplus \partial(B) \tag{8}$$



$\Rightarrow$ Other decompositions are possible:

$$9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(3B) \oplus \partial(3B) \tag{9}$$

Histogram based algorithms (I)

Original idea of HUANG et al. [1979]:

- a local histogram is computed in a sliding window.

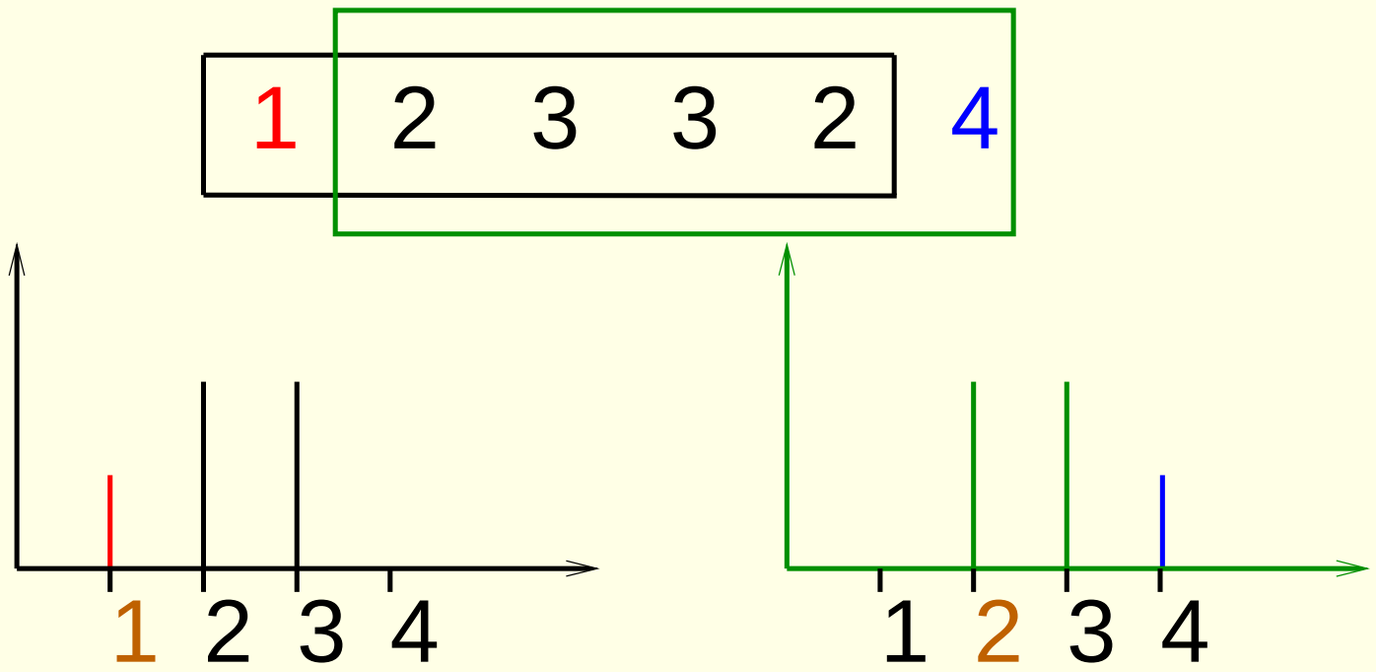
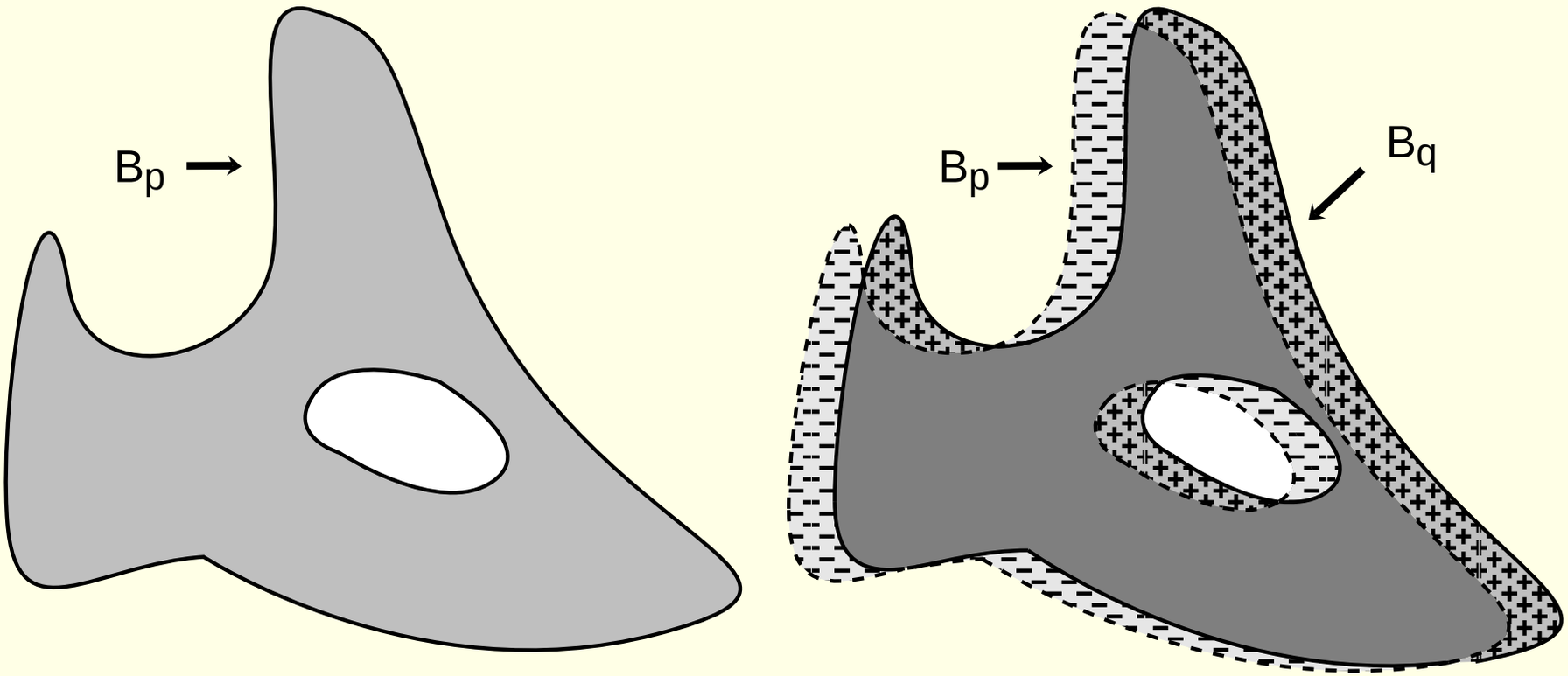


Figure 1: Illustration in the case of an erosion.

2D structuring elements



To be removed

To be added

Statistical properties of an erosion/dilation

Paper by STEVENSON (1996)

Notations

Let $X_1, X_2, \ldots, X_N$ be random variables with a same distribution.

$f_X(x)$ is the PDF and $F_X(x) = P(X \leq x)$

We define: $Y = \min\{X_1, X_2, \ldots, X_N\}$ et $Z = \max\{X_1, X_2, \ldots, X_N\}$.

Then **[Dilation]**

$$\begin{aligned} F_Z(z) = P(Z \leq z) &= P(\max\{X_1, X_2, \ldots, X_N\} \leq z) && (10)\\ &= P(X_1 \leq z, X_2 \leq z, \ldots, X_N \leq z) && (11)\\ &= P(X_1 \leq z)P(X_2 \leq z) \ldots P(X_N \leq z) && (12)\\ &= (F_X(z))^N && (13) \end{aligned}$$

and

$$f_Z(z) = \frac{dF_Z(z)}{dz} = \frac{d\,(F_X(z))^N}{dz} = N f_X(z)\,(F_X(z))^{N-1} \qquad (14)$$

[Erosion]

$$
\begin{aligned}
F_Y(y) = P(Y \leq y) &= P(\min\{X_1, X_2, \ldots, X_N\} \leq y) && (15)\\
&= 1 - P(\min\{X_1, X_2, \ldots, X_N\} > y) && (16)\\
&= 1 - P(X_1 > y)P(X_2 > y) \ldots P(X_N > y) && (17)\\
&= 1 - (1 - P(X_1 \leq y))^N && (18)\\
&= 1 - (1 - F_X(y))^N && (19)
\end{aligned}
$$

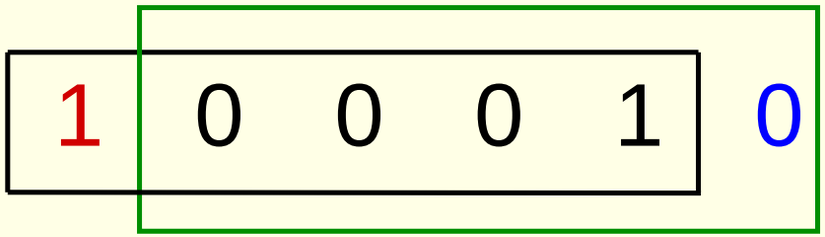
What could we do when a min/max value is lost?

Solution: built a list of local maximums and keep them in memory.

DOUGLAS (1996). Running Max/Min calculation using a pruned ordered list.

Histogram based algorithms (II)

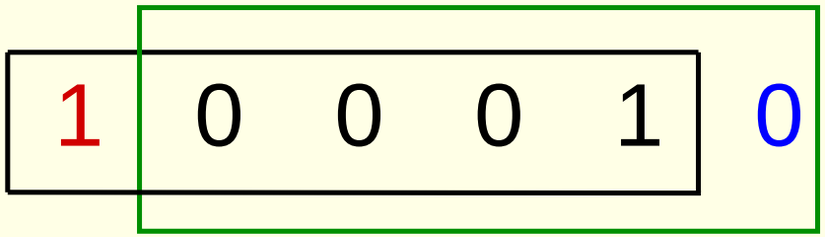
In the case of a binary image:



- only a single counter is needed

Histogram based algorithms (II)

In the case of a binary image:



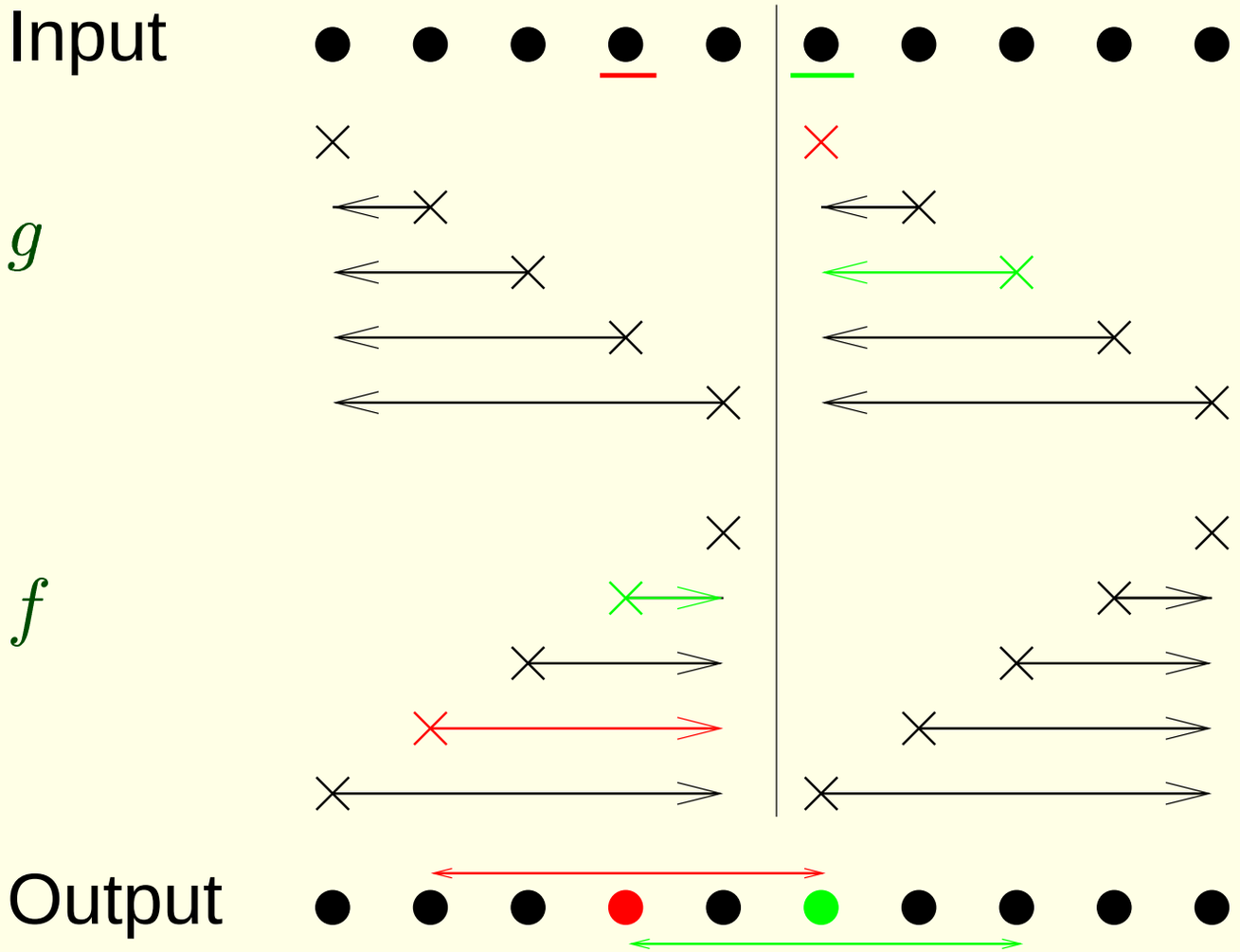
- only a single counter is needed

Properties:

- the computation time does not depend on the size of the structuring element
- the computation time depends on the image content
- applicable to arbitrary structuring elements

VAN HERK's algorithm (I)

VAN HERK (1992) proposed a 1-D algorithm based on separability and a combination of block recursive series which are evaluated forwards and backwards.



VAN HERK's algorithm (II)

Properties:

- 3 comparisons per pixel
- number of operations independent of the structuring element size
- but the structure is dependent of the structuring element size!
- applicable to binary morphology

Further improvements proposed by GIL and KIMMEL (ISMM 2000)

VAN HERK's algorithm will be used for benchmarking

Directional Morphological Filtering

Problem statement: how should we handle an erosion on a Bresenham line?

Answer: papers by JONES, SOILLE and TALBOT (2002)

Algorithms dedicated to binary morphology

- YOUNG et al. (1981) has proposed methods suited for hardware implementations

- VAN VLIET et al. (1988), SCHMITT (1989), and VINCENT (1991) provided methods which analyze the border of both X and B.

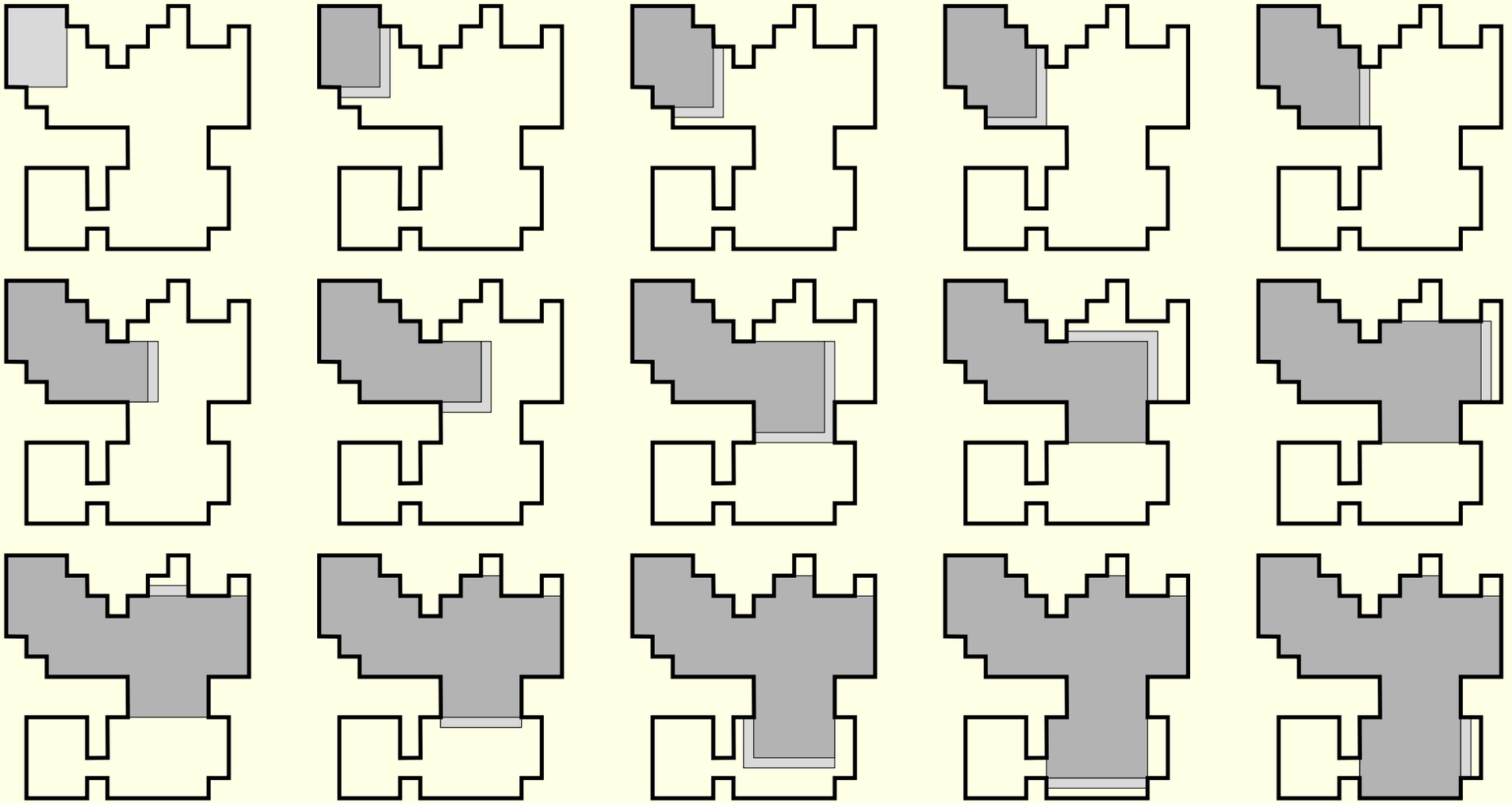
Distance based algorithms

LAY (1987) developed an algorithm based on the distance function.

X	0	1	1	1	1	1	1	0	1	1	1	1	0
Distance	0	1	2	3	3	2	1	0	1	2	2	1	0
$X \ominus 5H$	0	0	0	1	1	0	0	0	0	0	0	0	0

Algorithms based on the geometric interpretation of openings

Propagation algorithm for binary openings with a rectangle

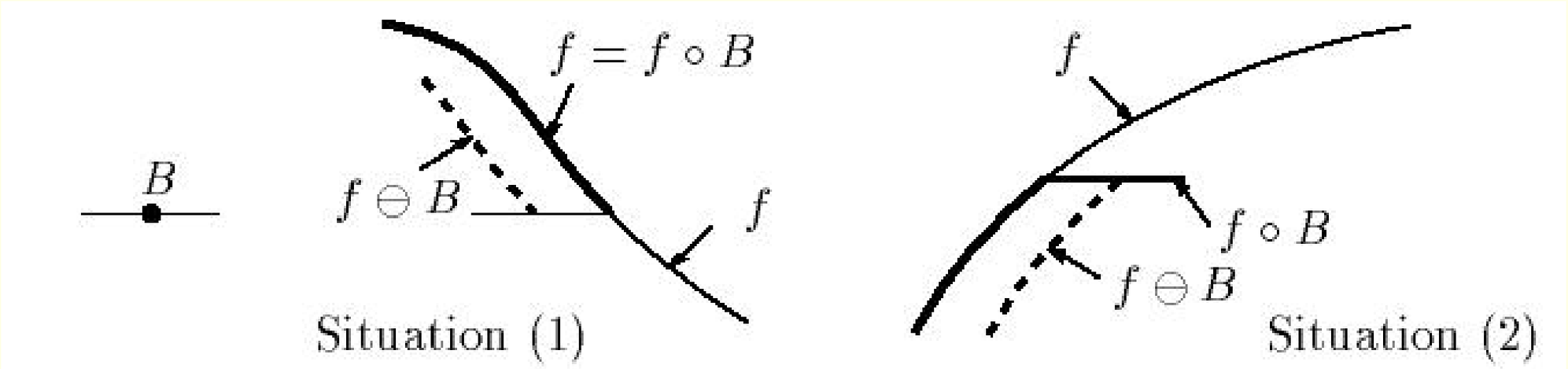


Remark:

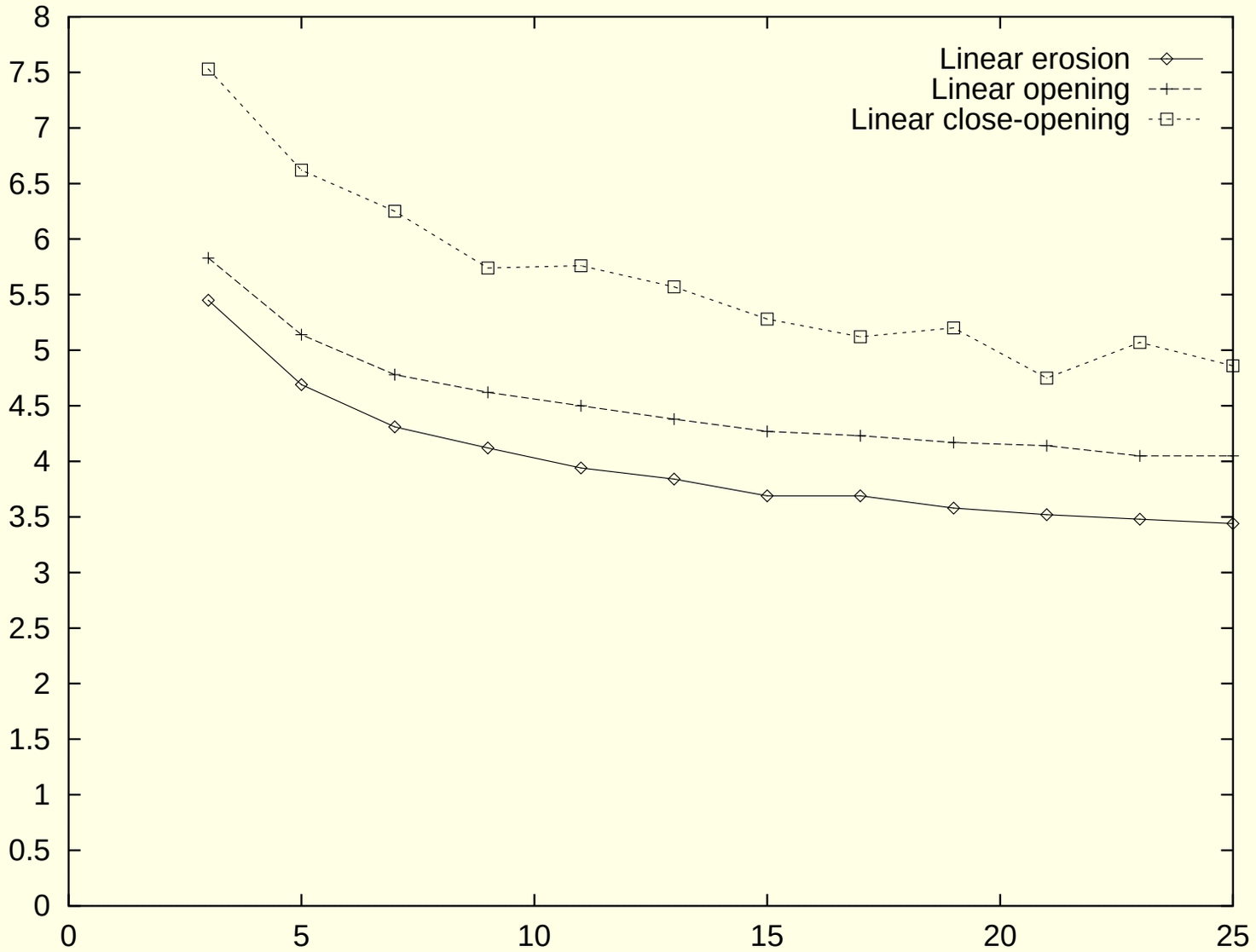
$$(X \circ B) \ominus B = X \ominus B \tag{20}$$

$\Rightarrow$ the opening provides the erosion!

Single pass opening algorithms



Computation time



There are methods for the direct computation of linear openings (also applicable to functions)

$$X \circ (nH \oplus mV) \neq (X \circ nH) \circ mV \tag{21}$$

But direct linear opening algorithms can still help reducing the computation times as

$$\begin{aligned} X \circ (nH \oplus mV) &= (((X \ominus nH) \ominus mV) \oplus nH) \oplus mV && (22) \\ &= (((X \ominus nH) \ominus mV) \oplus mV) \oplus nH && (23) \\ &= ((X \ominus nH) \circ mV) \oplus nH && (24) \end{aligned}$$

First family of algorithms: 2D scan-based algorithms

2 phases:

1. Scan the input image and build two matrices.

2. Fill the output image with thresholded values.

Our algorithms are similar to algorithms that use thresholded distance maps.

Description of the first phase

The first matrix, denoted $HOR[i, j]$ or HOR, is filled with the distance of each pixel contained in an object to the right border of that object.

Original image										HOR									
1	1	1	1	1	0	0	0	1	0	5	4	3	2	1	0	0	0	1	0
1	1	1	1	1	1	0	1	0	0	6	5	4	3	2	1	0	1	0	0
0	1	1	1	1	1	1	1	0	0	0	7	6	5	4	3	2	1	0	0
1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1	0	0	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	1	1	1	0	1	0	0	6	5	4	3	2	1
1	1	1	0	1	1	1	1	1	1	3	2	1	0	6	5	4	3	2	1
1	1	1	1	0	1	1	1	1	1	4	3	2	1	0	5	4	3	2	1

Building the second matrix: program fragment

$VER[i,j]$ is the length of the vertical segment of HOR that has all its value larger or equal to $HOR[i,j]$.

for all i, j **do**
  if $HOR[i,j] \neq 0$ **then**
    $length \leftarrow 1$
    `/* Scan to the top */`
    $k \leftarrow j - 1$
    while $HOR[i,k] \geq HOR[i,j]$ **do**
      $length \leftarrow length + 1$
      $k \leftarrow k - 1$
    end while
    `/* Scan to the bottom */`
    $k \leftarrow j + 1$
    while $HOR[i,k] \geq HOR[i,j]$ **do**
      $length \leftarrow length + 1$
      $k \leftarrow k + 1$
    end while
    $VER[i,j] \leftarrow length$
  else
    $VER[i,j] \leftarrow 0$
  end if

First phase: final result

Original image										HOR										VER									
1	1	1	1	1	0	0	0	1	0	5	4	3	2	1	0	0	0	1	0	2	5	5	5	7	0	0	0	1	0
1	1	1	1	1	1	0	1	0	0	6	5	4	3	2	1	0	1	0	0	1	4	4	4	6	7	0	7	0	0
0	1	1	1	1	1	1	1	0	0	0	7	6	5	4	3	2	1	0	0	0	3	3	3	5	6	6	7	0	0
1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	1	2	2	2	4	5	5	5	5	0
0	1	1	1	1	1	1	1	1	0	0	8	7	6	5	4	3	2	1	0	0	2	2	2	4	5	5	5	5	0
0	1	0	0	1	1	1	1	1	1	0	1	0	0	6	5	4	3	2	1	0	8	0	0	2	3	3	3	3	3
1	1	1	0	1	1	1	1	1	1	3	2	1	0	6	5	4	3	2	1	2	2	2	0	2	3	3	3	3	3
1	1	1	1	0	1	1	1	1	1	4	3	2	1	0	5	4	3	2	1	1	1	1	1	0	3	3	3	3	3

Figure 2: A binary object and its corresponding two matrices HOR and VER.

All rectangles included in X are mentioned in HOR and VER

but

there is some redundancy

The second phase (shown for $B = 4H \oplus 4V$)

1. Selection of positions where $HOR[i,j] \geq 4$ and $VER[i,j] \geq 4$

Original image										HOR										VER									
1	1	1	1	1	0	0	0	1	0	5	4	3	2	1	0	0	0	1	0	2	5	5	5	7	0	0	0	1	0
1	1	1	1	1	1	0	1	0	0	6	5	4	3	2	1	0	1	0	0	1	4	4	4	6	7	0	7	0	0
0	1	1	1	1	1	1	1	0	0	0	7	6	5	4	3	2	1	0	0	0	3	3	3	5	6	6	7	0	0
1	1	1	1	1	1	1	1	1	0	9	8	7	6	5	4	3	2	1	0	1	2	2	2	4	5	5	5	5	0
0	1	1	1	1	1	1	1	1	0	0	8	7	6	5	4	3	2	1	0	0	2	2	2	4	5	5	5	5	0
0	1	0	0	1	1	1	1	1	1	0	1	0	0	6	5	4	3	2	1	0	8	0	0	2	3	3	3	3	3
1	1	1	0	1	1	1	1	1	1	3	2	1	0	6	5	4	3	2	1	2	2	2	0	2	3	3	3	3	3
1	1	1	1	0	1	1	1	1	1	4	3	2	1	0	5	4	3	2	1	1	1	1	1	0	3	3	3	3	3

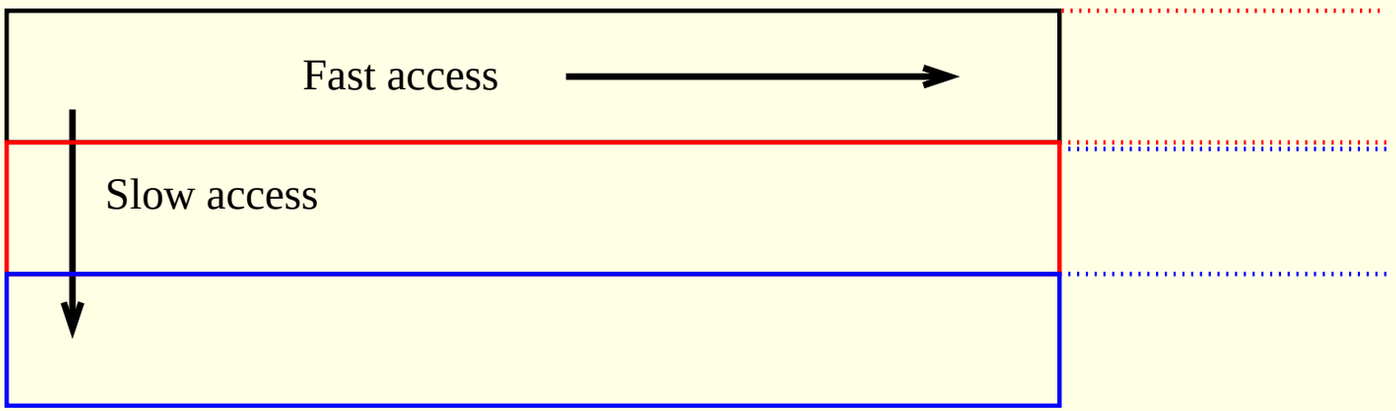
2. Fill the first column with run-length values

3. Fill the output image

Original image										INTERMEDIATE RESULT										OUT									
1	1	1	1	1	0	0	0	1	0	0	4	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0
1	1	1	1	1	1	0	1	0	0	0	5	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0
0	1	1	1	1	1	1	1	0	0	0	5	0	0	4	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	0	0	5	0	0	5	4	0	0	0	0	0	1	1	1	1	1	1	1	1	0
0	1	1	1	1	1	1	1	1	0	0	5	0	0	5	4	0	0	0	0	0	1	1	1	1	1	1	1	1	0
0	1	0	0	1	1	1	1	1	1	0	0	0	0	5	4	0	0	0	0	0	0	0	0	1	1	1	1	1	0
1	1	1	0	1	1	1	1	1	1	0	0	0	0	5	4	0	0	0	0	0	0	0	0	1	1	1	1	1	0
1	1	1	1	0	1	1	1	1	1	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	1	1	1	1	0

Implementation issues: optimizations

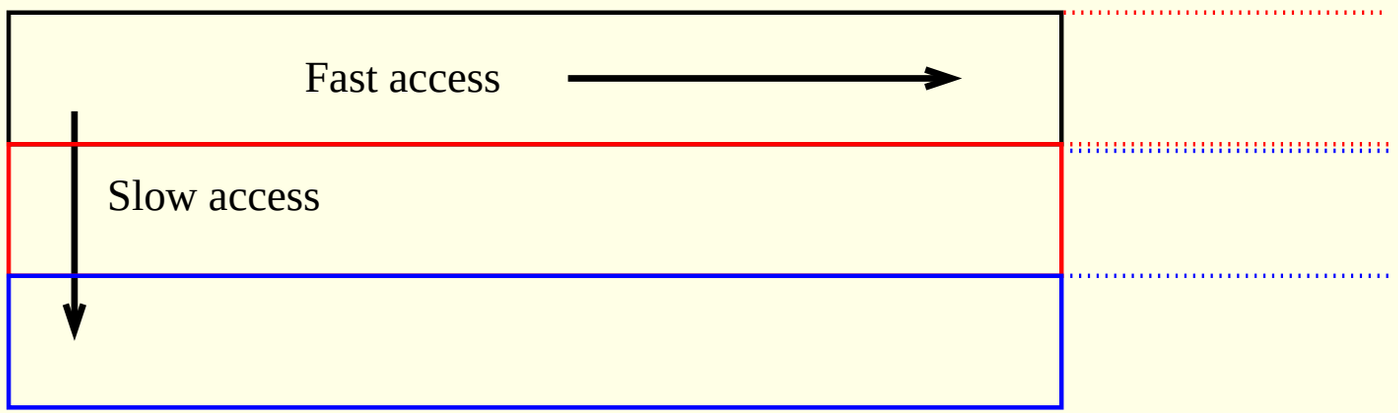
- Better use of hardware and software characteristics.
  - Because of to the linear structure of memory blocks, accesses along a row in a 2-D array are faster than accesses along a column.



  - HOR is transposed before being stored.

Implementation issues: optimizations

- Better use of hardware and software characteristics.
  - Because of to the linear structure of memory blocks, accesses along a row in a 2-D array are faster than accesses along a column.



  - HOR is transposed before being stored.

- Avoid some computation redundancy.
  - If $HOR[i,j] = HOR[i,j-1]$ then $VER[i,j] = VER[i,j-1]$.

All these optimizations were included in the implementations.

Generalization of the 2D scan-based algorithm to label images ($B = 3H \oplus 3V$)

LABEL IMAGE

1	1	1	2	2	2	2	2	2	2	2	3
1	1	1	1	2	2	2	2	2	3	3	3
1	1	1	1	2	2	2	2	3	3	3	3
4	1	1	1	2	2	2	3	3	3	3	3
4	4	4	1	1	2	2	2	2	3	3	3
4	4	4	4	4	2	2	2	3	3	3	3
4	4	4	4	4	2	2	2	3	3	3	3
4	4	4	4	2	2	2	2	3	3	3	3

VER

3	4	4	1	1	1	1	1	1	1	1	8
2	3	3	4	2	2	2	2	2	7	7	8
2	3	3	4	3	3	3	3	2	7	7	8
5	3	3	4	4	8	8	1	2	7	7	8
4	4	4	1	1	1	1	1	1	7	7	8
2	2	2	2	2	4	4	4	3	7	7	8
2	2	2	2	2	4	4	4	3	7	7	8
3	3	3	3	1	4	4	4	3	7	7	8

HOR

3	2	1	8	7	6	5	4	3	2	1	1
4	3	2	1	5	4	3	2	1	3	2	1
4	3	2	1	4	3	2	1	4	3	2	1
1	3	2	1	3	2	1	5	4	3	2	1
3	2	1	2	1	4	3	2	1	3	2	1
5	4	3	2	1	3	2	1	4	3	2	1
5	4	3	2	1	3	2	1	4	3	2	1
4	3	2	1	4	3	2	1	4	3	2	1

OUT

1	1	1	0	2	2	2	2	0	0	0	0
1	1	1	1	2	2	2	2	0	3	3	3
1	1	1	1	2	2	2	2	0	3	3	3
0	1	1	1	2	2	2	0	0	3	3	3
4	4	4	0	0	2	2	2	0	3	3	3
4	4	4	0	0	2	2	2	3	3	3	3
4	4	4	0	0	2	2	2	3	3	3	3
4	4	4	0	0	2	2	2	3	3	3	3

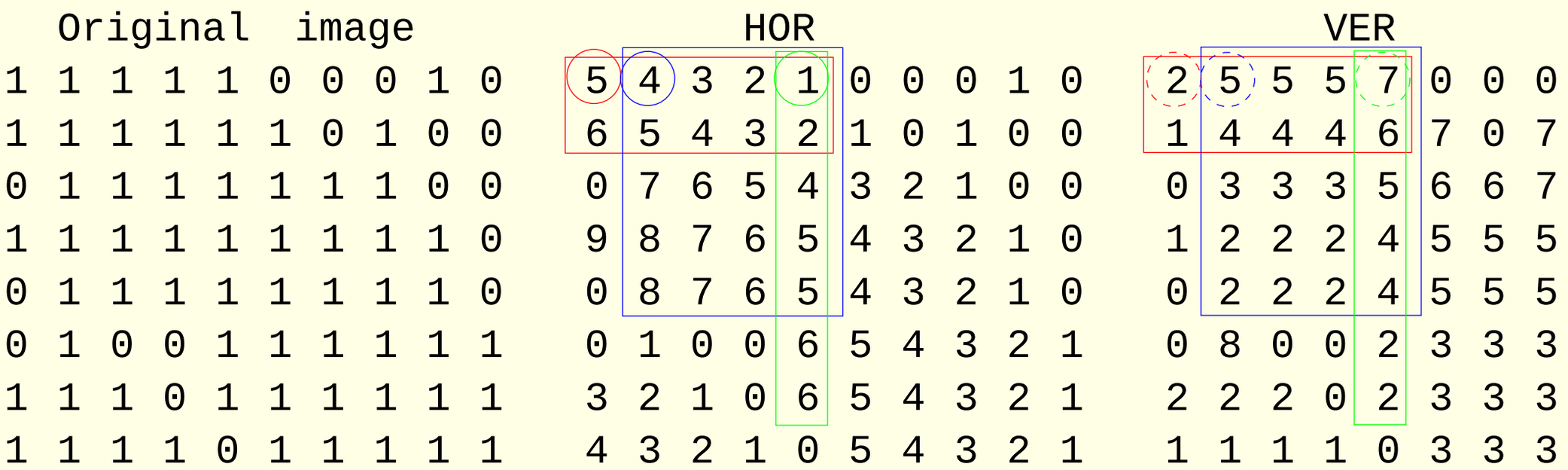
Second family of algorithms: list-based algorithms

Principle: build a list of rectangles $L = R_0, \ldots, R_L$ such that

- $\bigcup R_i = X$
- all rectangles of $\mathcal{L}$ are maximum in size
- the list does not contain any redundant rectangle

1000 lines of C code!

Description

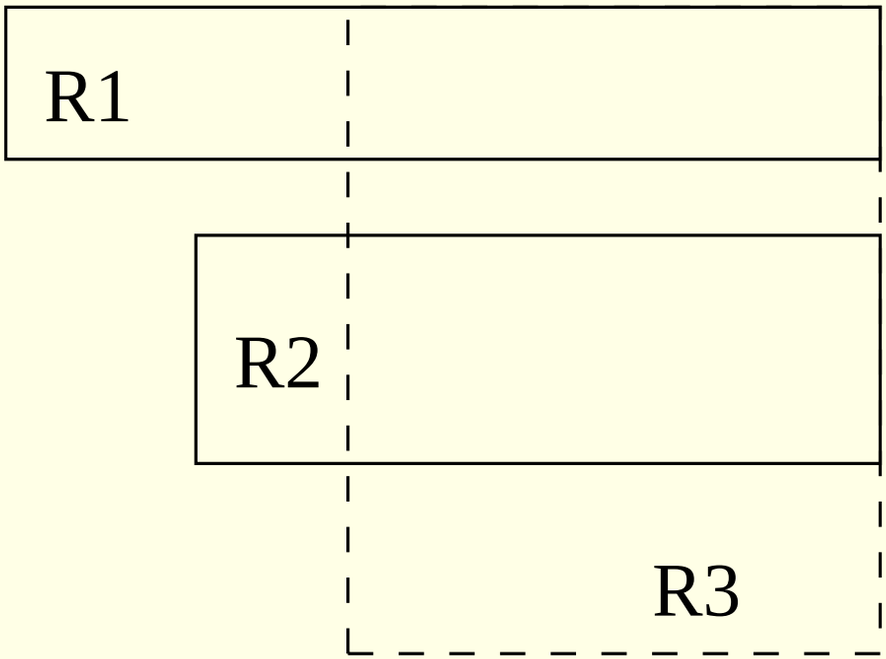


(i,j)=(0,0) Width=5 Height=2
(i,j)=(0,0) Width=4 Height=5
(i,j)=(0,0) Width=1 Height=7

Basically, VER is replaced by a list.

21 non-redundant rectangles.

Problem of multiple intersections



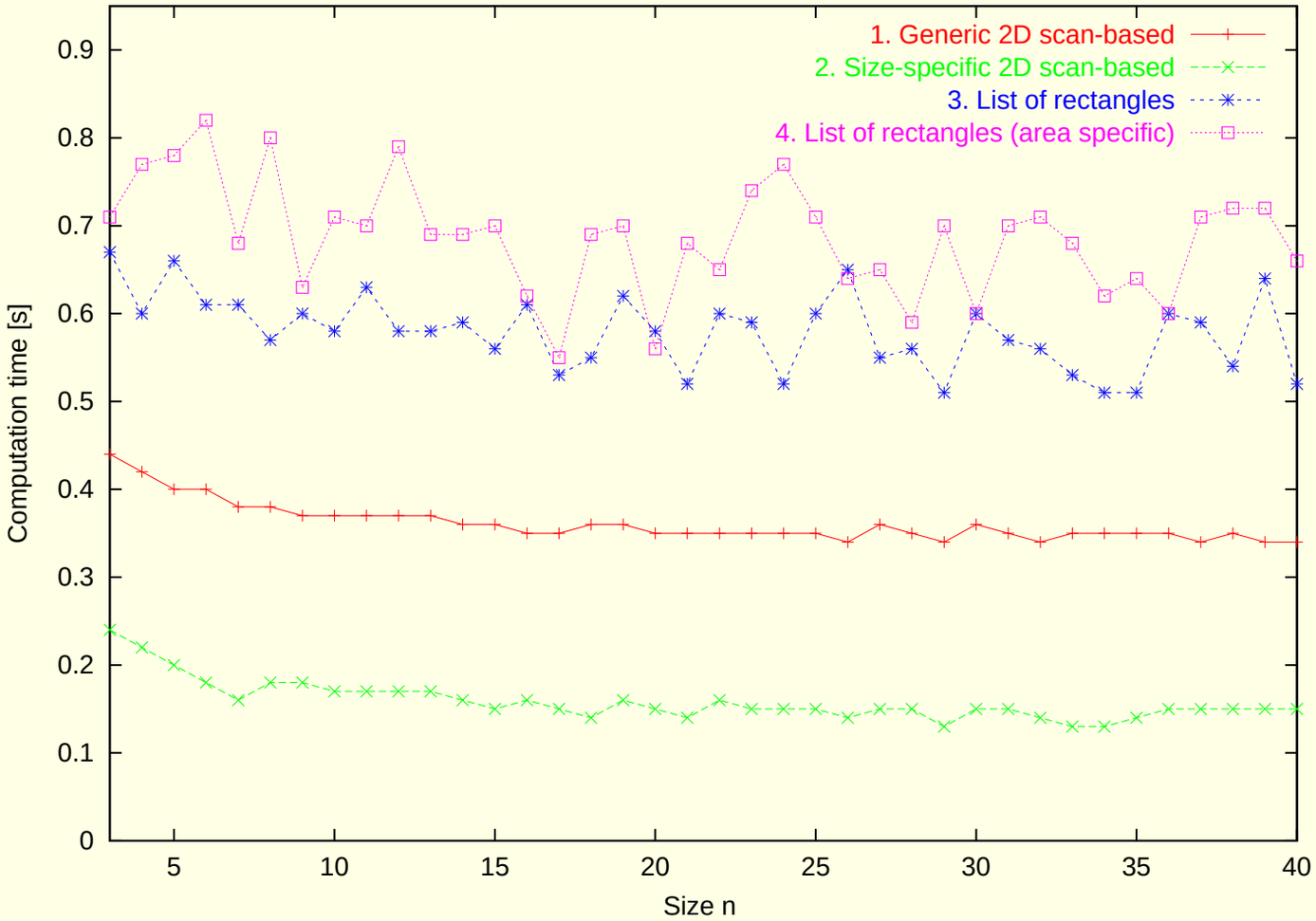
Two solutions:

1. Avoid rewriting pixels in OUT

2. Stores all the intersections between elements of $\mathcal{L}$.

A thorough examination of all the intersections leads to a variant (called area specific) for which no output pixel is written twice.

Computation times of 2D scan and list-based algorithms



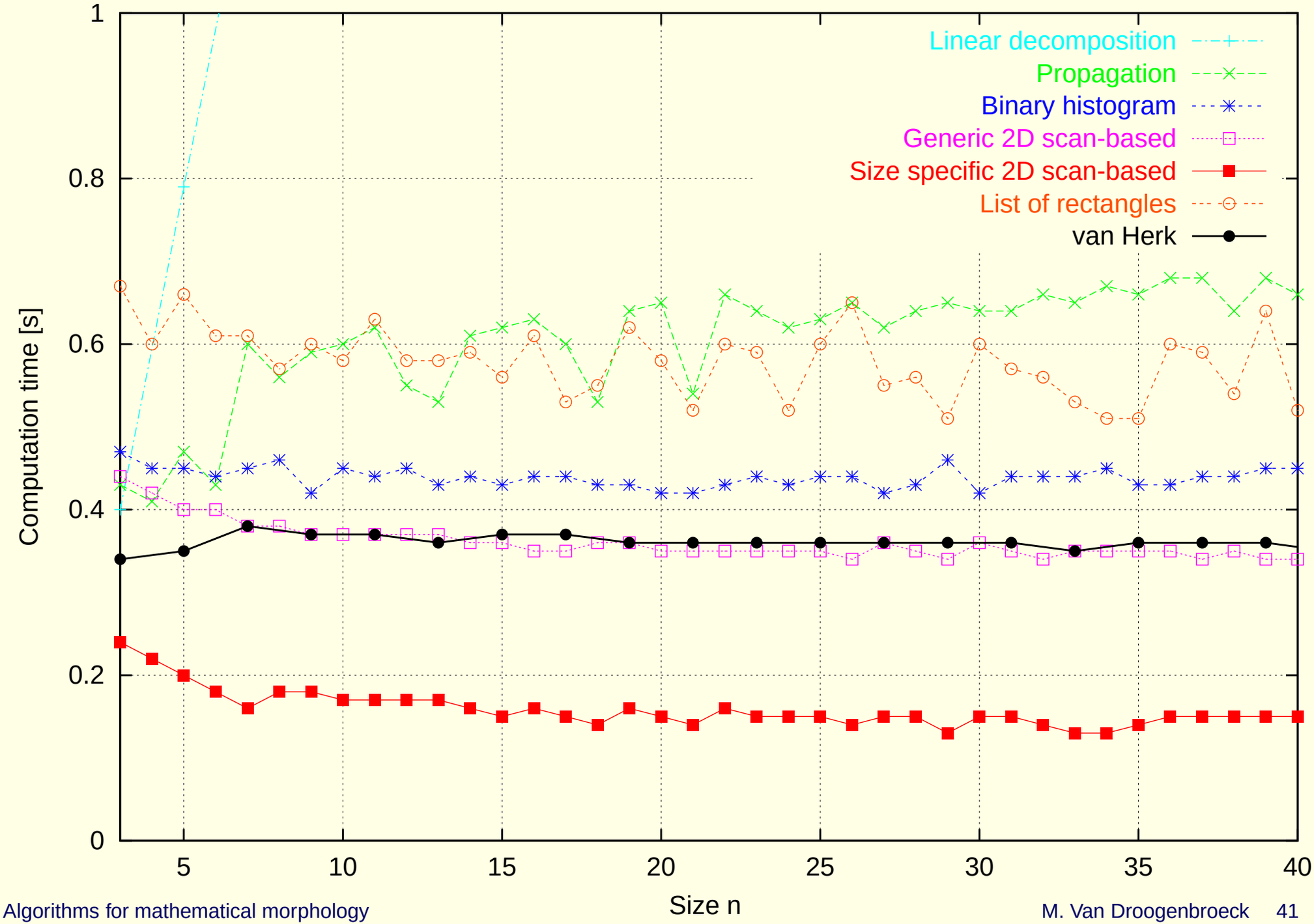
- The respective contributions of the first and second phase to the total amount of time are about 85%-15%.

Comparison with other algorithms

- All algorithms have been reimplemented and optimized for the case of binary images.

- For the particular case of VAN HERK's algorithm (often used as a benchmark), we transposed the intermediate matrix twice, after $X \ominus nH$ and $((X \ominus nH) \ominus mV) \oplus mV$, to achieve a fair comparison with our algorithms.

Computation times of several algorithms for binary openings


Linear decomposition
Propagation
Binary histogram
Generic 2D scan-based
Size specific 2D scan-based
List of rectangles
van Herk
Computation time [s]
Size n
0
0.2
0.4
0.6
0.8
1
5
10
15
20
25
30
35
40

Comparison of algorithms that do not depend on the size of the structuring element

	Histogram	van Herk	Scan-based	List-based
Program complexity	Very low	Low	Medium	High
Relative computation speed	Fast	Fast	Very fast	Slow
Allow multiple computations	No	No	Yes	Yes
Dependent on the content	Slightly	No	Yes	Yes

Granulometries

VINCENT (2000). Granulometries and Opening Trees.