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:

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

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

For a function $f$

$$f \ominus B = \bigwedge_{b \in B} f_{-b}$$

$$f \oplus B = \bigvee_{b \in B} f_b$$
• The opening

\[ X \circ B = (X \ominus B) \oplus B \]  

(4)

• Geometric interpretation:

\[ X = \bigcup \{B_p \mid B_p \subseteq X \} \]  

(5)
Notations and conventions

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

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

(6)
Notations and conventions

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

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

(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

$$nB = B \oplus B \oplus \ldots \oplus B$$

($n-1$ dilations)

- 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 $n \times m$

   (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 $n \times m$

   (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 $B$,

$$9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(2B) \oplus \partial(4B)$$  \hspace{1cm} (7)

(usual notation of size)
Logarithmic decomposition

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

\[
9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(2B) \oplus \partial(4B)
\]

(usual notation of size)

Theorem.

\[
B \oplus B = B \oplus \partial(B)
\]
Logarithmic decomposition

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

$$9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(2B) \oplus \partial(4B)$$  \hspace{1cm} (7)

(usual notation of size)

Theorem.

$$B \oplus B = B \oplus \partial(B)$$  \hspace{1cm} (8)

$\Rightarrow$ Other decompositions are possible:

$$9B = B \oplus \partial(B) \oplus \partial(B) \oplus \partial(3B) \oplus \partial(3B)$$  \hspace{1cm} (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]

$$F_Z(z) = P(Z \leq z) = P(\max\{X_1, X_2, \ldots, X_N\} \leq z)$$

$$= P(X_1 \leq z, X_2 \leq z, \ldots, X_N \leq z)$$

$$= P(X_1 \leq z)P(X_2 \leq z)\ldots P(X_N \leq z)$$

$$= (F_X(z))^N$$

and

$$f_Z(z) = \frac{dF_Z(z)}{dz} = \frac{d(F_X(z))^N}{dz} = Nf_X(z)(F_X(z))^{N-1}$$
\[ F_Y(y) = P(Y \leq y) = P(\min\{X_1, X_2, \ldots, X_N\} \leq y) \]
\[ = 1 - P(\min\{X_1, X_2, \ldots, X_N\} > y) \]
\[ = 1 - P(X_1 > y)P(X_2 > y)\ldots P(X_N > y) \]
\[ = 1 - (1 - P(X_1 \leq y))^N \]
\[ = 1 - (1 - F_X(y))^N \]
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:

```
1 0 0 0 1 0
```

- only a single counter is needed
In the case of a binary image:

```
1 0 0 0 1
```

- 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.

\[
\begin{align*}
\text{Input} & \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \\
\text{Output} & \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet
\end{align*}
\]
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 GIIL 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.

\[
\begin{array}{cccccccccccccccc}
\text{Distance} & 0 & 1 & 2 & 3 & 3 & 2 & 1 & 0 & 1 & 2 & 2 & 1 & 0 \\
X & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
X \oplus 5H & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Algorithms based on the geometric interpretation of openings

Propagation algorithm for binary openings with a rectangle

[Diagram of propagation algorithm with images of binary openings progression]
Remark:

\[(X \circ B) \ominus B = X \ominus B\]  \hspace{1cm} (20)

⇒ the opening provides the erosion!
Single pass opening algorithms

Situation (1)

\[ f = f \circ B \]

Situation (2)

\[ f = f \ominus B \]

\[ f \circ B \]

\[ f \ominus B \]
### Computation time

![Computation Time Graph](image)

- **Linear erosion**
- **Linear opening**
- **Linear close-opening**

---

*Algorithms for mathematical morphology*  
M. Van Droogenbroeck
There are methods for the direct computation of linear openings (also applicable to functions)

\[ X \circ (nH \oplus mV) \neq (X \circ nH) \circ mV \]  \hspace{1cm} (21)

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

\[
X \circ (nH \oplus mV) = (((X \ominus nH) \ominus mV) \oplus nH) \oplus mV \\
= (((X \ominus nH) \ominus mV) \oplus mV) \oplus nH \\
= ((X \ominus nH) \circ mV) \oplus nH \hspace{1cm} (22, 23, 24)
\]
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.

<table>
<thead>
<tr>
<th>Original image</th>
<th>HOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 0 0 0 1 0</td>
<td>5 4 3 2 1 0 0 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 0 1 0 0</td>
<td>6 5 4 3 2 1 0 1 0 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 0 0</td>
<td>0 7 6 5 4 3 2 1 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 0</td>
<td>9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 1 0</td>
<td>0 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>0 1 0 0 1 1 1 1 1</td>
<td>0 1 0 0 6 5 4 3 2 1</td>
</tr>
<tr>
<td>1 1 1 0 1 1 1 1 1</td>
<td>3 2 1 0 6 5 4 3 2 1</td>
</tr>
<tr>
<td>1 1 1 1 0 1 1 1 1</td>
<td>4 3 2 1 0 5 4 3 2 1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Original image</th>
<th>HOR</th>
<th>VER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 0 0 0 1 0</td>
<td>5 4 3 2 1 0 0 0 1 0</td>
<td>2 5 5 5 7 0 0 0 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 0 1 0 0</td>
<td>6 5 4 3 2 1 0 1 0 0</td>
<td>1 4 4 4 6 7 0 7 0 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 0 0</td>
<td>0 7 6 5 4 3 2 1 0 0</td>
<td>0 3 3 3 5 6 6 7 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 0 0</td>
<td>9 8 7 6 5 4 3 2 1 0</td>
<td>1 2 2 2 4 5 5 5 5 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 1 0</td>
<td>0 8 7 6 5 4 3 2 1 0</td>
<td>0 2 2 2 4 5 5 5 5 0</td>
</tr>
<tr>
<td>0 1 0 0 1 1 1 1 1 1</td>
<td>0 1 0 0 6 5 4 3 2 1</td>
<td>0 8 0 0 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1 1 1 0 1 1 1 1 1</td>
<td>3 2 1 0 6 5 4 3 2 1</td>
<td>2 2 2 0 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1 1 1 1 0 1 1 1 1</td>
<td>4 3 2 1 0 5 4 3 2 1</td>
<td>1 1 1 1 0 3 3 3 3 3</td>
</tr>
</tbody>
</table>

*Figure 2: A binary object and its corresponding two matrices* \( \text{HOR} \) and \( \text{VER} \).

All rectangles included in \( X \) are mentioned in \( \text{HOR} \) and \( \text{VER} \), but there is some redundancy.
The second phase (shown for $B = 4H \oplus 4V$)
1. Selection of positions where $\text{HOR}[i, j] \geq 4$ and $\text{VER}[i, j] \geq 4$

<table>
<thead>
<tr>
<th>Original image</th>
<th>HOR</th>
<th>VER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 0 0 0 1 0</td>
<td>5 4 3 2 1 0 0 0 1 0</td>
<td>2 5 5 5 7 0 0 0 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 0 1 0 0</td>
<td>6 5 4 3 2 1 0 1 0 0</td>
<td>1 4 4 6 7 0 7 0 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 0 0</td>
<td>0 7 6 5 4 3 2 1 0 0</td>
<td>0 3 3 5 6 6 7 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 0</td>
<td>9 8 7 6 5 4 3 2 1 0</td>
<td>1 2 2 2 4 5 5 5 5 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 1 0</td>
<td>0 8 7 6 5 4 3 2 1 0</td>
<td>0 2 2 2 4 5 5 5 5 0</td>
</tr>
<tr>
<td>0 1 0 0 1 1 1 1 1 1</td>
<td>0 1 0 0 6 5 4 3 2 1</td>
<td>0 8 0 0 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1 1 1 0 1 1 1 1 1 1</td>
<td>3 2 1 0 6 5 4 3 2 1</td>
<td>2 2 2 0 2 3 3 3 3 3</td>
</tr>
<tr>
<td>1 1 1 1 0 1 1 1 1 1</td>
<td>4 3 2 1 0 5 4 3 2 1</td>
<td>1 1 1 1 0 3 3 3 3 3</td>
</tr>
</tbody>
</table>

2. Fill the first column with run-length values
3. Fill the output image

<table>
<thead>
<tr>
<th>Original image</th>
<th>INTERMEDIATE RESULT</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 0 0 0 1 0</td>
<td>0 4 0 0 0 0 0 0 0 0</td>
<td>0 1 1 1 1 0 0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 0 1 0 0</td>
<td>0 5 0 0 0 0 0 0 0 0</td>
<td>0 1 1 1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 0 0</td>
<td>0 5 0 0 4 0 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1 1 0</td>
<td>0 5 0 0 5 4 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 1 1 1 1 1 1 1 1 0</td>
<td>0 5 0 0 5 4 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 1 0 0 1 1 1 1 1 1</td>
<td>0 0 0 0 5 4 0 0 0 0</td>
<td>0 0 0 0 1 1 1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 0 1 1 1 1 1 1</td>
<td>0 0 0 0 5 4 0 0 0 0</td>
<td>0 0 0 0 1 1 1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 1 0 1 1 1 1 1</td>
<td>0 0 0 0 4 0 0 0 0 0</td>
<td>0 0 0 0 1 1 1 1 1 0</td>
</tr>
</tbody>
</table>
Implementation issues: optimizations

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

\[ H O R \] is transposed before being stored.
Implementation issues: optimizations

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

- \texttt{HOR} is transposed before being stored.

- Avoid some computation redundancy.
  - If \( \texttt{HOR}[i, j] = \texttt{HOR}[i, j - 1] \) then \( \texttt{VER}[i, j] = \texttt{VER}[i, j - 1] \).

All these optimizations were included in the implementations.
Generalization of the 2D scan-based algorithm to label images

\( B = 3H \oplus 3V \)

<table>
<thead>
<tr>
<th>LABEL IMAGE</th>
<th>VER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 2 2 2 2 2 2 2 2 3</td>
<td>3 4 4 1 1 1 1 1 1 1 1 1 8</td>
</tr>
<tr>
<td>1 1 1 1 2 2 2 2 3 3 3 3</td>
<td>2 3 3 4 2 2 2 2 7 7 7 7 8</td>
</tr>
<tr>
<td>1 1 1 1 2 2 2 2 3 3 3 3</td>
<td>2 3 3 4 3 3 3 3 2 7 7 7 8</td>
</tr>
<tr>
<td>4 1 1 1 2 2 2 3 3 3 3 3</td>
<td>5 3 3 4 4 8 8 1 2 7 7 7 8</td>
</tr>
<tr>
<td>4 4 4 1 1 2 2 2 3 3 3 3</td>
<td>4 4 4 1 1 1 1 1 1 7 7 7 8</td>
</tr>
<tr>
<td>4 4 4 4 4 2 2 2 3 3 3 3</td>
<td>2 2 2 2 2 4 4 4 3 7 7 7 8</td>
</tr>
<tr>
<td>4 4 4 4 4 2 2 2 3 3 3 3</td>
<td>2 2 2 2 2 4 4 4 3 7 7 7 8</td>
</tr>
<tr>
<td>4 4 4 4 2 2 2 3 3 3 3</td>
<td>3 3 3 3 1 4 4 4 3 7 7 7 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOR</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 2 1 8 7 6 5 4 3 2 1 1</td>
<td>1 1 1 0 2 2 2 2 0 0 0 0 0</td>
</tr>
<tr>
<td>4 3 2 1 5 4 3 2 1 3 2 1</td>
<td>1 1 1 1 2 2 2 2 0 3 3 3 3</td>
</tr>
<tr>
<td>4 3 2 1 4 3 2 1 4 3 2 1</td>
<td>1 1 1 1 2 2 2 2 0 3 3 3 3</td>
</tr>
<tr>
<td>1 3 2 1 3 2 1 5 4 3 2 1</td>
<td>0 1 1 1 2 2 2 0 0 3 3 3 3</td>
</tr>
<tr>
<td>3 2 1 2 1 4 3 2 1 3 2 1</td>
<td>4 4 4 0 0 2 2 2 0 3 3 3 3</td>
</tr>
<tr>
<td>5 4 3 2 1 3 2 1 4 3 2 1</td>
<td>4 4 4 0 0 2 2 2 3 3 3 3</td>
</tr>
<tr>
<td>5 4 3 2 1 3 2 1 4 3 2 1</td>
<td>4 4 4 0 0 2 2 2 3 3 3 3</td>
</tr>
<tr>
<td>4 3 2 1 4 3 2 1 4 3 2 1</td>
<td>4 4 4 0 0 2 2 2 3 3 3 3</td>
</tr>
</tbody>
</table>
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

Basically, \( \text{VER} \) is replaced by a list.

21 non-redundant rectangles.
Problem of multiple intersections

Two solutions:

1. Avoid rewriting pixels in \textit{OUT}

2. Stores all the intersections between elements of \textit{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

1. Generic 2D scan-based
2. Size-specific 2D scan-based
3. List of rectangles
4. List of rectangles (area specific)
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
Comparison of algorithms that do not depend on the size of the structuring element

<table>
<thead>
<tr>
<th></th>
<th>Histogram</th>
<th>van Herk</th>
<th>Scan-based</th>
<th>List-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program complexity</td>
<td>Very low</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Relative computation speed</td>
<td>Fast</td>
<td>Fast</td>
<td>Very fast</td>
<td>Slow</td>
</tr>
<tr>
<td>Allow multiple computations</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dependent on the content</td>
<td>Slightly</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Granulometries