Digital Image Processing

COSC 6380/4393

Lecture – 10

Sept. 21st, 2023

Pranav Mantini

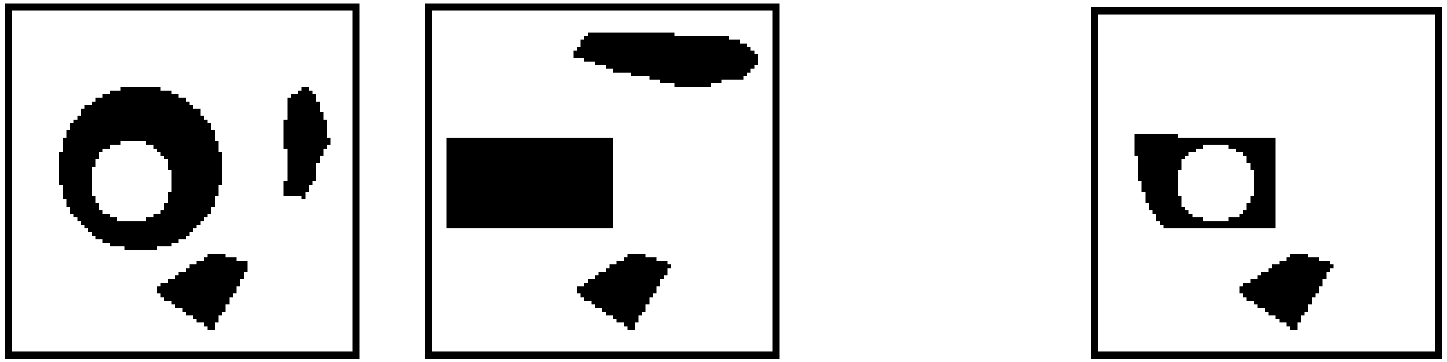
Slides from Dr. Shishir K Shah and Frank (Qingzhong) Liu

Review: THE BASIC LOGICAL OPERATIONS

- We will use only a **few simple** logical operations
- Suppose that X_1 ,..., X_n are **binary variables**
- For example, pixels from one or more binary images
- Here is the notation we will use:
- **Logical Complement**: NOT(X_1) = complement of X_1
- **Logical AND**: AND(X_1, X_2) = $X_1 \wedge X_2$
- **Logical OR**: OR(X1, X2) = X1 $\vee$ X2
- **Binary Majority:** MAJ(X_1 , X_2 , ..., X_n) = 1 if more 1's than 0's = 0 if more 0's than 1's

Review: BINARY AND

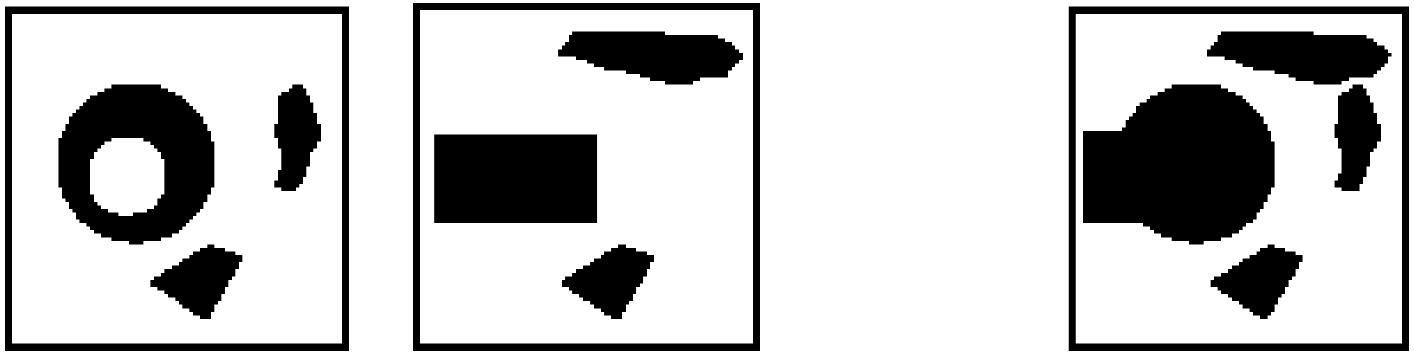
- The AND or intersection of two images:
- $\mathbf{J}_2 = \text{AND}(\mathbf{I}_1, \mathbf{I}_2) = \mathbf{I}_1 \wedge \mathbf{I}_2$ if $J_2(i, j) = \text{AND}[\, I_1(i, j), I_2(i, j)\,]$ for all (i, j)
- Shows the overlap of BLACK regions in I_1 and $\mathbf{I}_2$



$\mathbf{I}_1$ $\mathbf{I}_2$ $\mathbf{J}_2 = \mathbf{I}_1 \wedge \mathbf{I}_2$

Review: BINARY OR

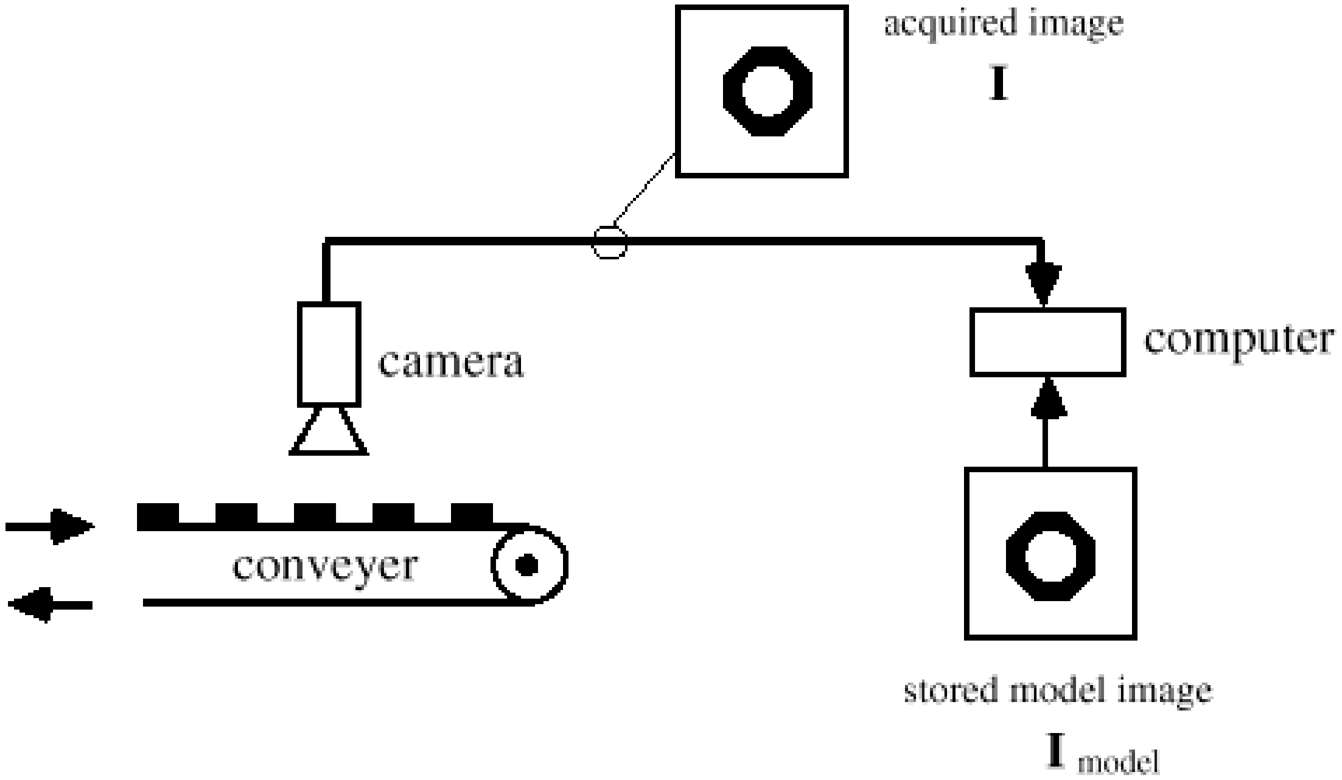
- The OR or union of two images:
- $\mathbf{J}_3 = \mathrm{OR}(\mathbf{I}_1 , \mathbf{I}_2) = \mathbf{I}_1 \vee \mathbf{I}_2$
  if $J_3 (i, j) = \mathrm{OR}[\, I_1 (i, j), I_2 (i, j) \,]$ for all (i, j)
- Shows the **overlap** of the WHITE regions in $\mathbf{I}_1$ and $\mathbf{I}_2$



$\mathbf{I}_1$ $\mathbf{I}_2$ $\mathbf{J}_3 = \mathbf{I}_1 \vee \mathbf{I}_2$

Review: EXAMPLE

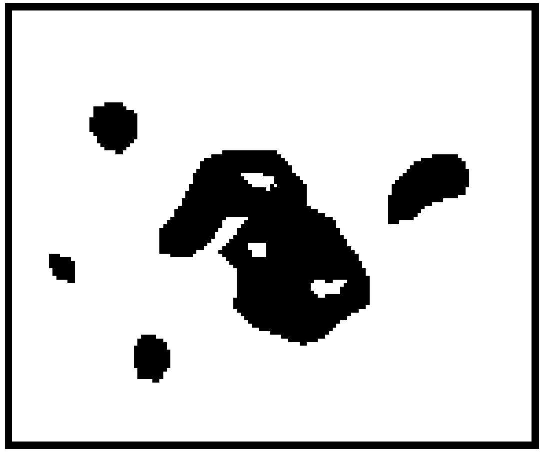
- An assembly-line image inspection system. Similar to many marketed by industry:



- **Objective**: Numerically compare the stored image $\mathbf{I}_{model}$ and the acquired image $\mathbf{I}$

Review: BLOB COLORING

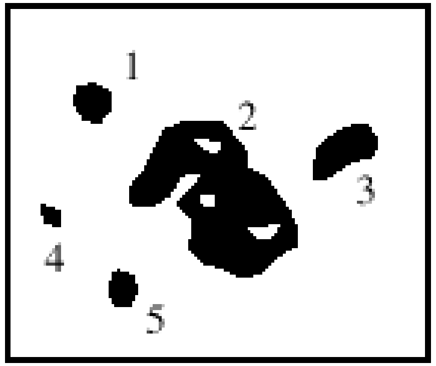
- A simple technique for **region classification** and **correction**
- **Motivation**: Gray-level image thresholding **usually** produces an imperfect binary image:
  - Extraneous blobs or holes due to noise
  - Extraneous blobs from thresholded objects of little interest
  - Nonuniform object/background surface reflectances



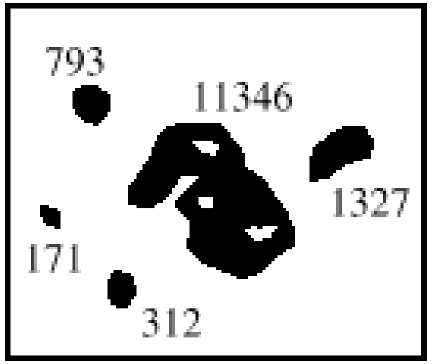
typical thresholded image result

EXAMPLE

- Using blob coloring



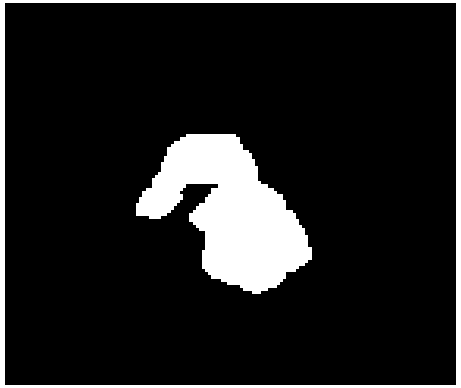
blob coloring result



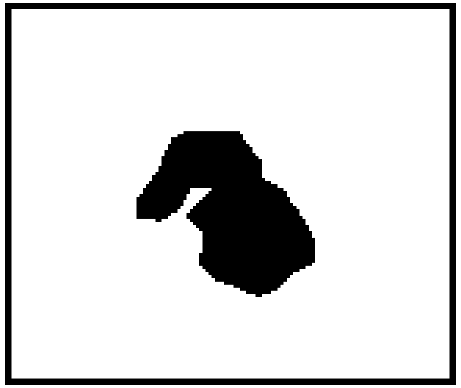
blob counting result

- "Color" of largest blob: **2**

EXAMPLE



minor region removal



complement

- Simple and effective, but doesn’t "cure" everything

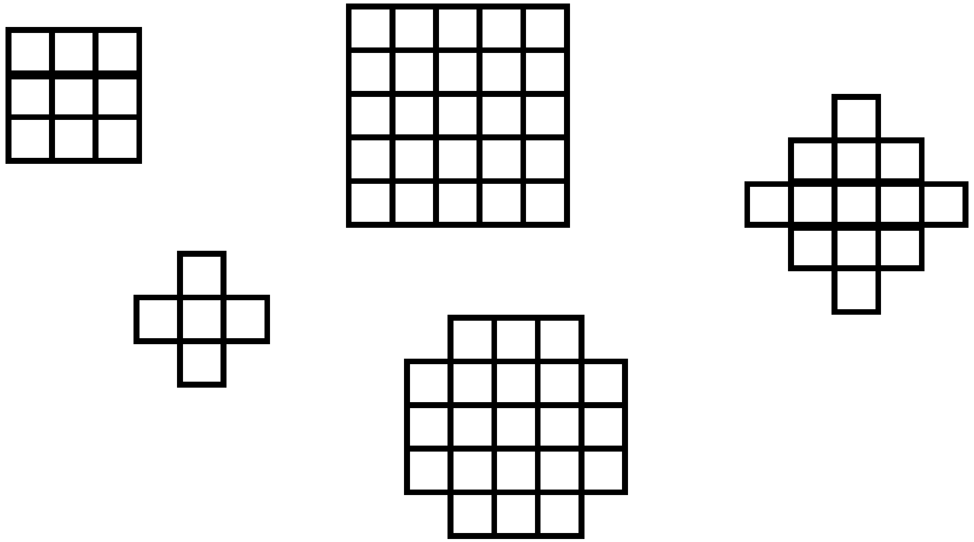
BINARY MORPHOLOGY

- The most powerful class of binary image operators
- A general framework known as **mathematical morphology**

  morphology = shape
- **Morphological operations** affect the **shapes** of **objects** and **regions** in binary images
- All processing is done on a **local basis** - region or blob shapes are affected in a local manner
- Morphological operators
  - Expand (dilate) objects
  - Shrink (erode) objects
  - Smooth object boundaries and eliminate small regions or holes
  - Fill gaps and eliminate 'peninsulas'
- All is accomplished using **local logical operations**

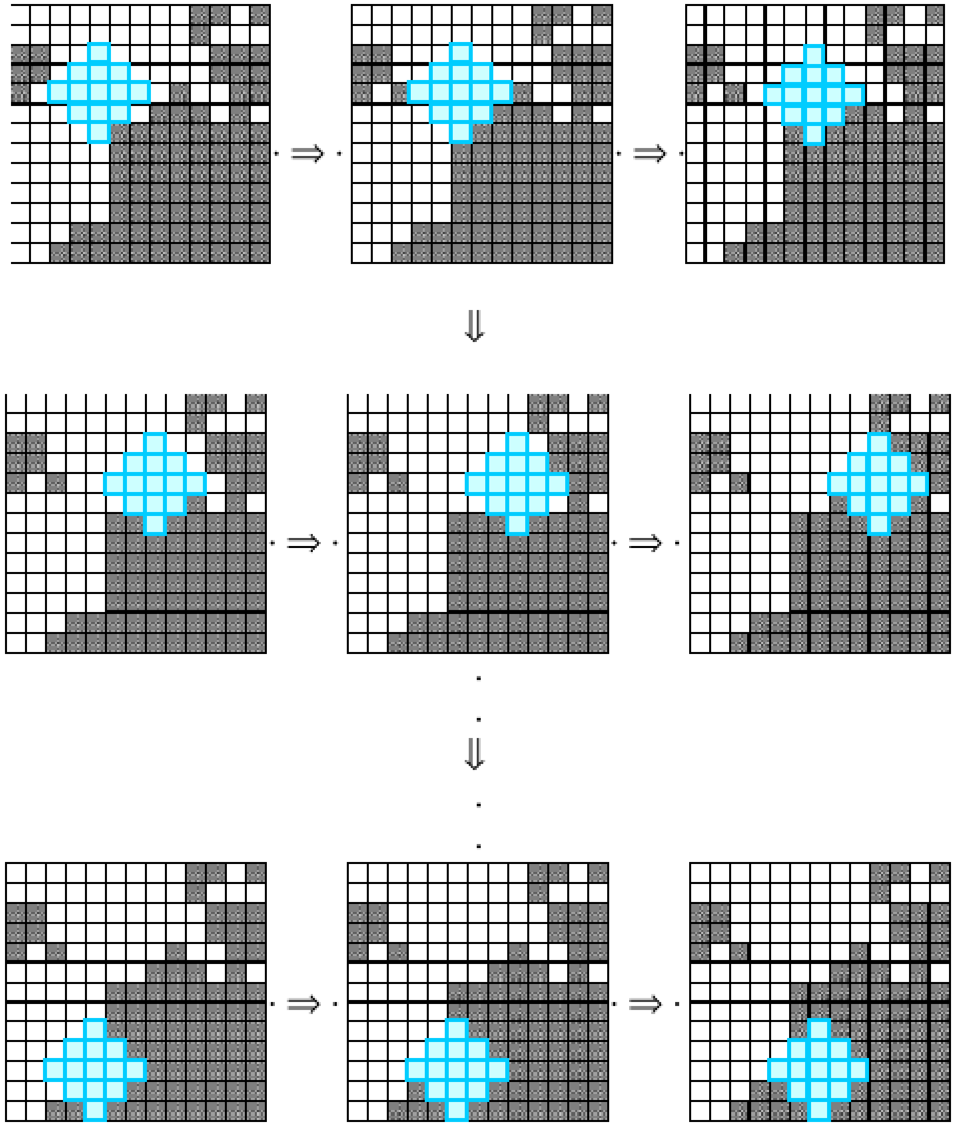
STRUCTURING ELEMENTS OR WINDOWS

- A **structuring element** is a geometric relationship between pixels
- Some examples:



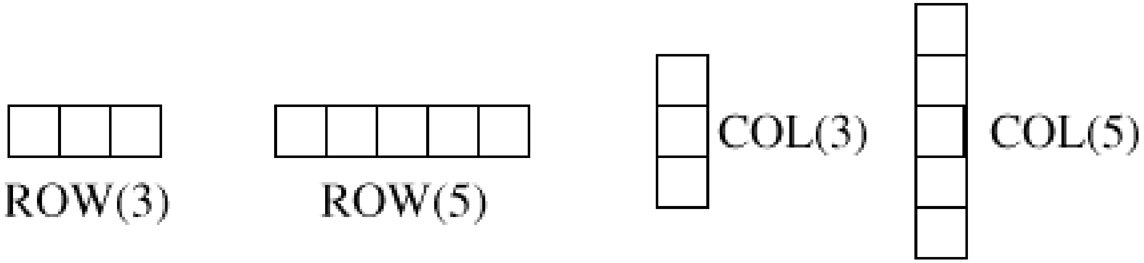
- Morphological operations are defined (conceptually) by moving a structuring element over the image to be modified, in such a way that it is centered over every image pixel at some point

STRUCTURING ELEMENTS



WINDOWING

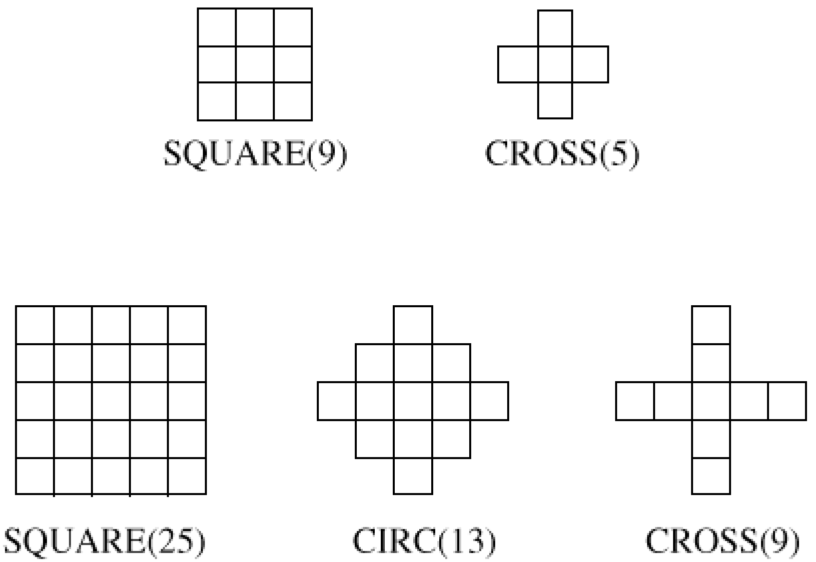
- Some typical windows:



1-D windows ROW(2M+1) and COL(2M+1).

- These operate on rows and columns only
- A window will always cover an **odd number** of pixels **2M+1**:
  - pairs of adjacent pixels, plus the center pixel
- Filtering operations are defined **symmetrically** this way

TWO-DIMENSIONAL WINDOWS



2-D windows SQUARE(2M+1), CROSS(2M+1), CIRC(2M+1)

- Again, 2M+1 denotes the **odd** number of pixels covered by the window
- Can generalize to arbitrary-size windows covering 2M+1 pixels
- These are the **most common** window shapes

Morphological Operations

Structuring element

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

Input binary image

Morphological Operations

Structuring element

Apply OR binary operation

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

Input binary image

Morphological Operations

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

Input binary image

Output/Filtered binary image

Morphological Operations

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

Input binary image

Output/Filtered binary image

Morphological Operations

$$1 \lor 0 \lor 0 = 1$$

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

Input binary image

	1			

Output/Filtered binary image

Morphological Operations

$0 \vee 0 \vee 1 = 1$

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

Input binary image

	1	1		

Output/Filtered binary image

Morphological Operations

$$0 \vee 1 \vee 0 = 1$$

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

Input binary image

	1	1	1	

Output/Filtered binary image

Morphological Operations

$0 \vee 0 \vee 0 = 0$

1	0	0	1	0
0	1	0	0	0
1	1	1	1	0
1	0	1	0	1
0	1	0	1	0

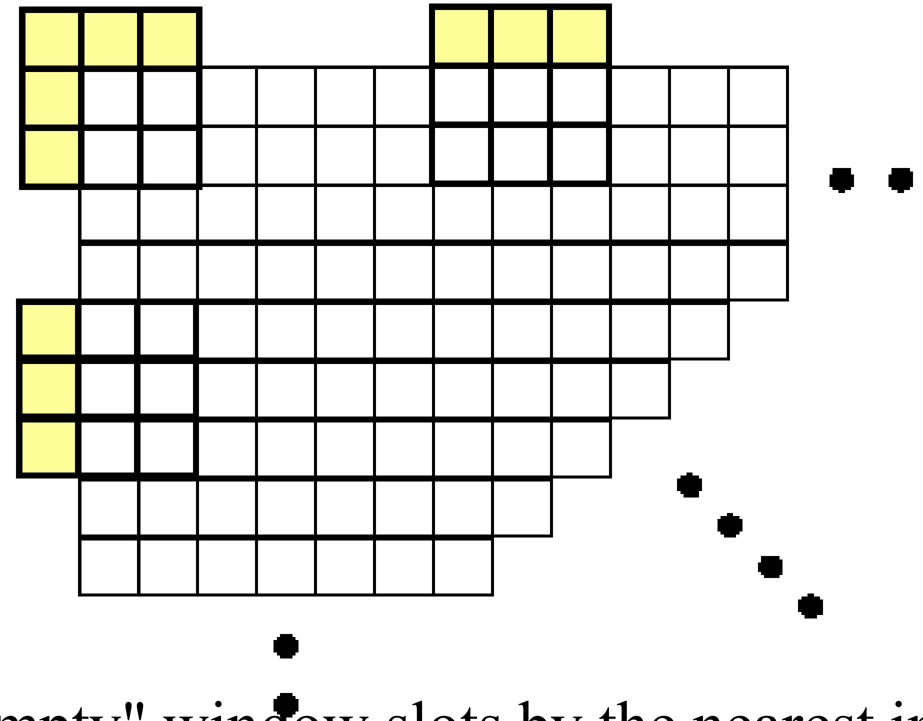
Input binary image

	1	1	1	
	1	1	0	

Output/Filtered binary image

EDGE-OF-IMAGE PROCESSING

- Window overlapping "empty space" :



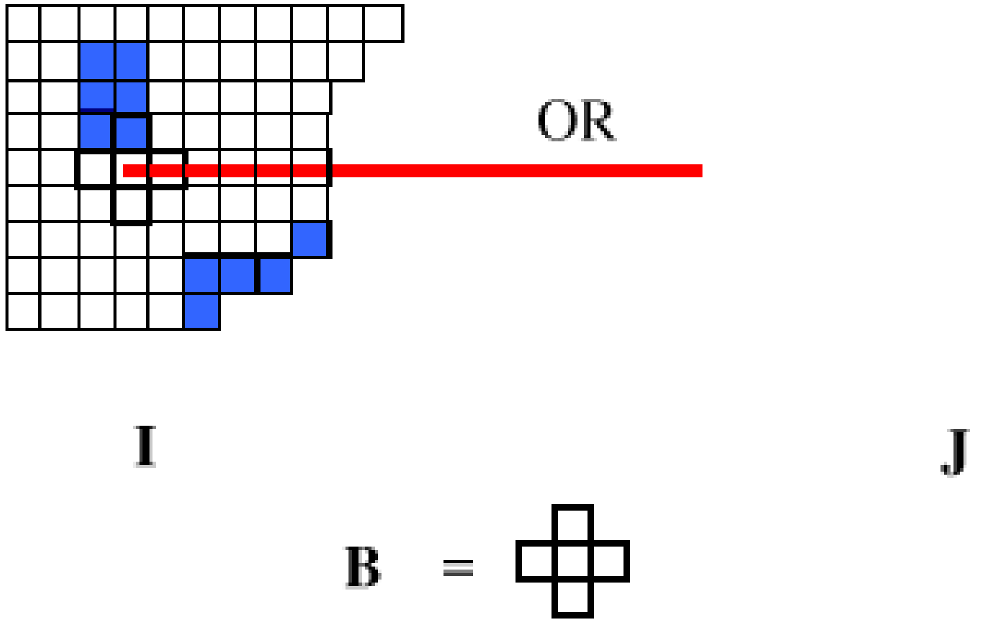
- **Convention**: fill the "empty" window slots by the nearest image pixel. This is called **replication**

DILATION, EROSION AND MEDIAN (MAJORITY)

- DILATION: Given a window **B** and a binary image **I**:
- $\mathbf{J}_1$ = DILATE(**I**, **B**) Apply OR operations within the moving window
- EROSION: Given a window **B** and a binary image **I**:
- $\mathbf{J}_2$ = ERODE(**I**, **B**) Apply AND operation within the moving window
- MEDIAN: Given a window **B** and a binary image **I**:
- $\mathbf{J}_3$ = MEDIAN(**I**, **B**) Apply MAJ operation within the moving window

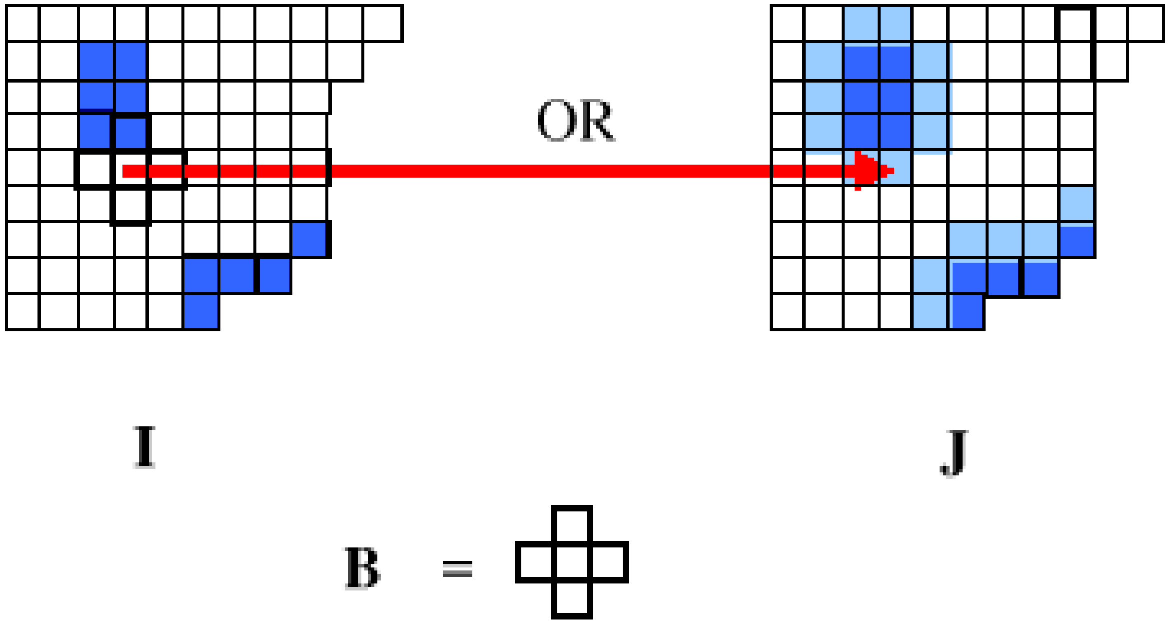
DILATION

- So-called because this operation **increases** the size of BLACK objects in a binary image
- Local Computation: **J** = DILATE(**I**, **B**)



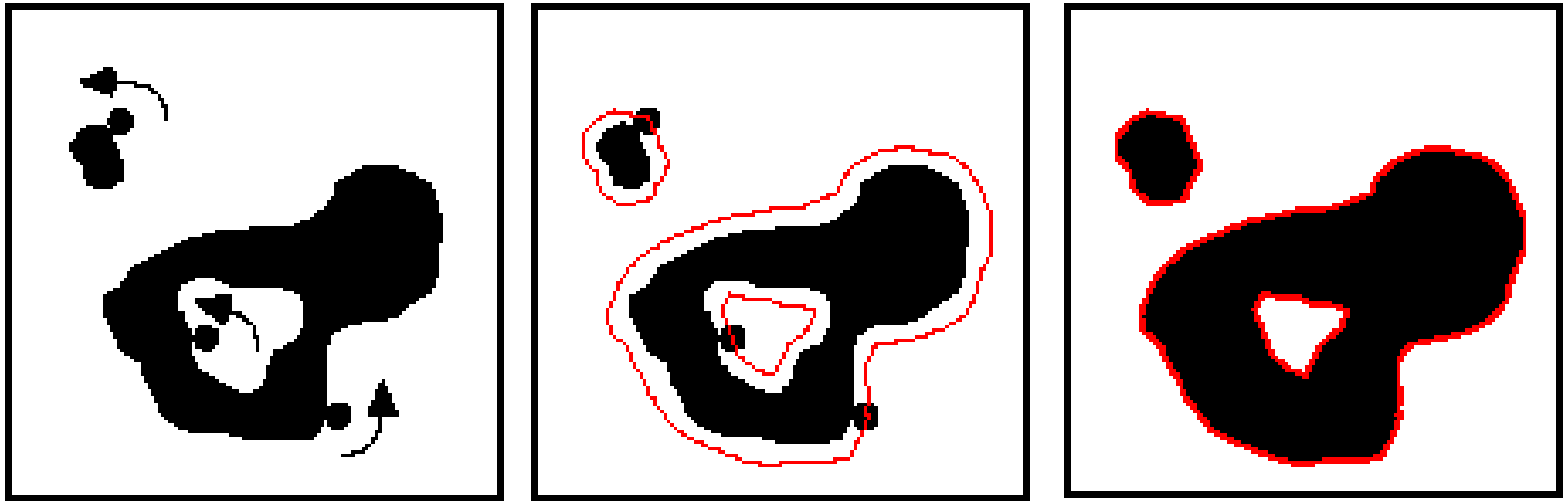
DILATION

- So-called because this operation **increases** the size of BLACK objects in a binary image
- Local Computation: **J** = DILATE(**I**, **B**)



DILATION

- Global Effect:



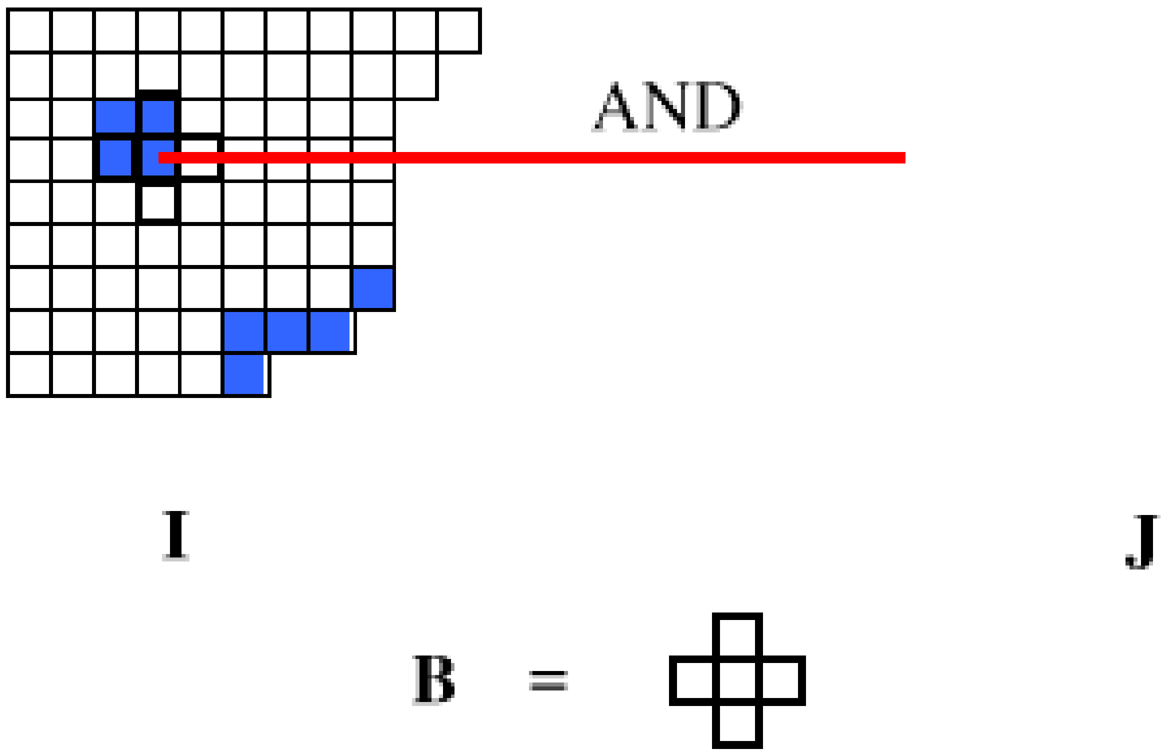
It is useful to think of the structuring element as rolling along all of the boundaries of all BLACK objects in the image.

The center point of the structuring element traces out a set of paths.

That form the boundaries of the dilated image.

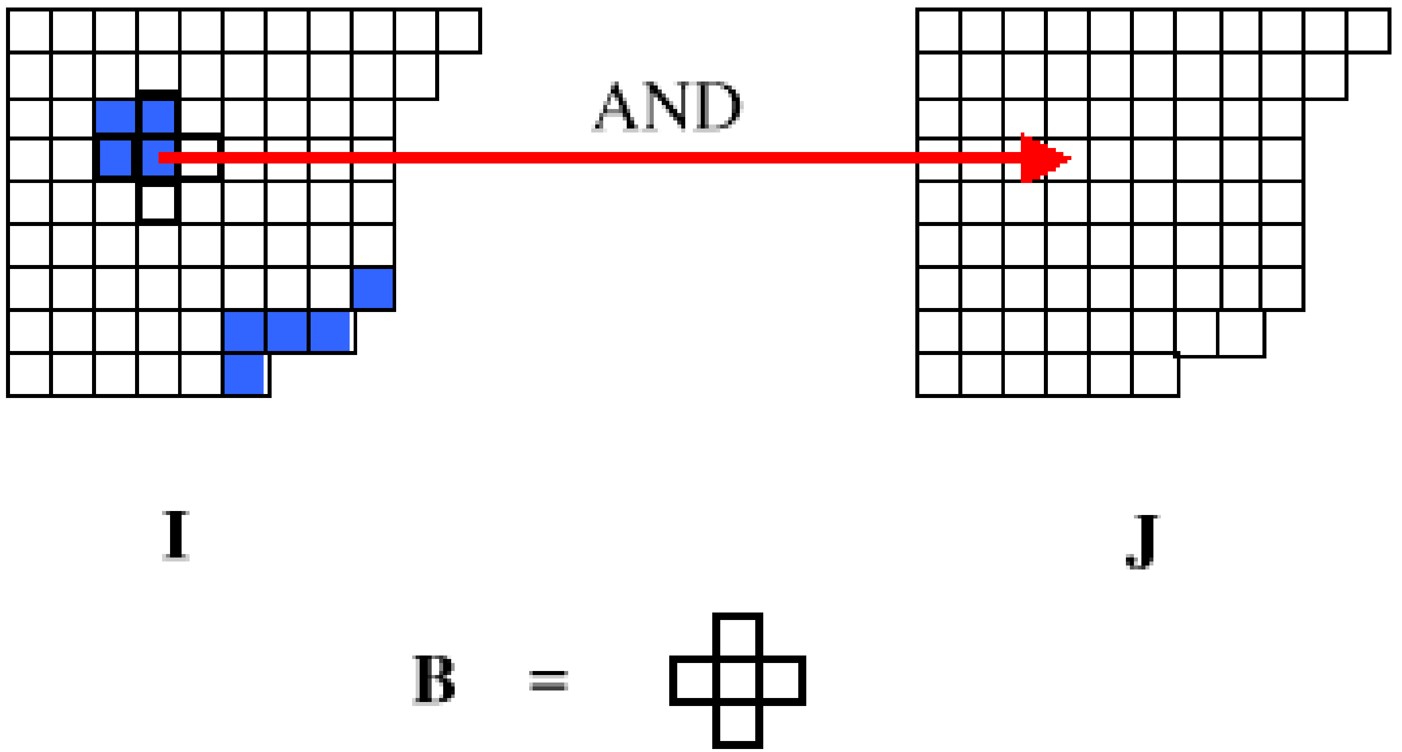
EROSION

- So-called because this operation **decreases** the size of BLACK objects in a binary image
- Local Computation: **J** = ERODE(**I**, **B**)



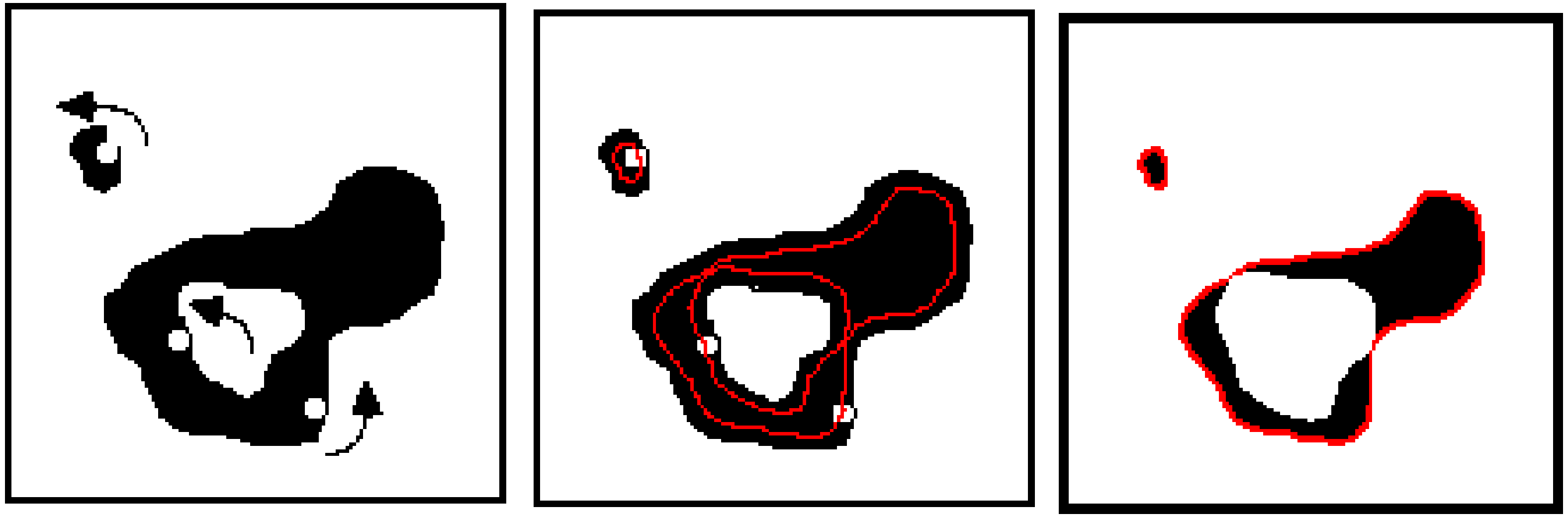
EROSION

- So-called because this operation **decreases** the size of BLACK objects in a binary image
- Local Computation: **J** = ERODE(**I**, **B**)



EROSION

- Global Effect:



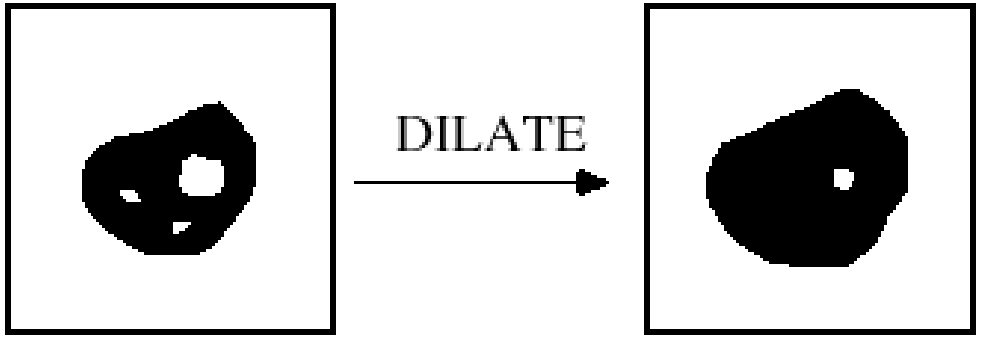
It is useful to think of the structuring element as rolling inside of the boundaries of all BLACK objects in the image.

The center point of the structuring element traces out a set of paths.

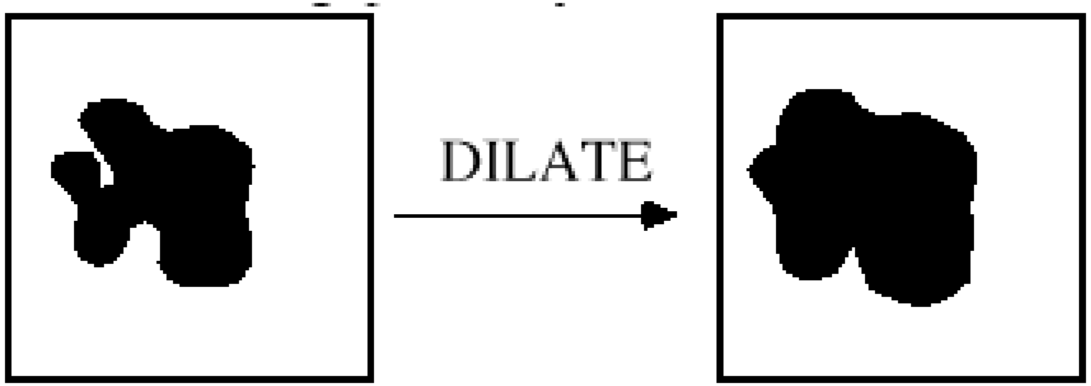
That form the boundaries of the eroded image.

QUALITATIVE PROPERTIES OF DILATION

- Dilation removes object holes of too-small size:

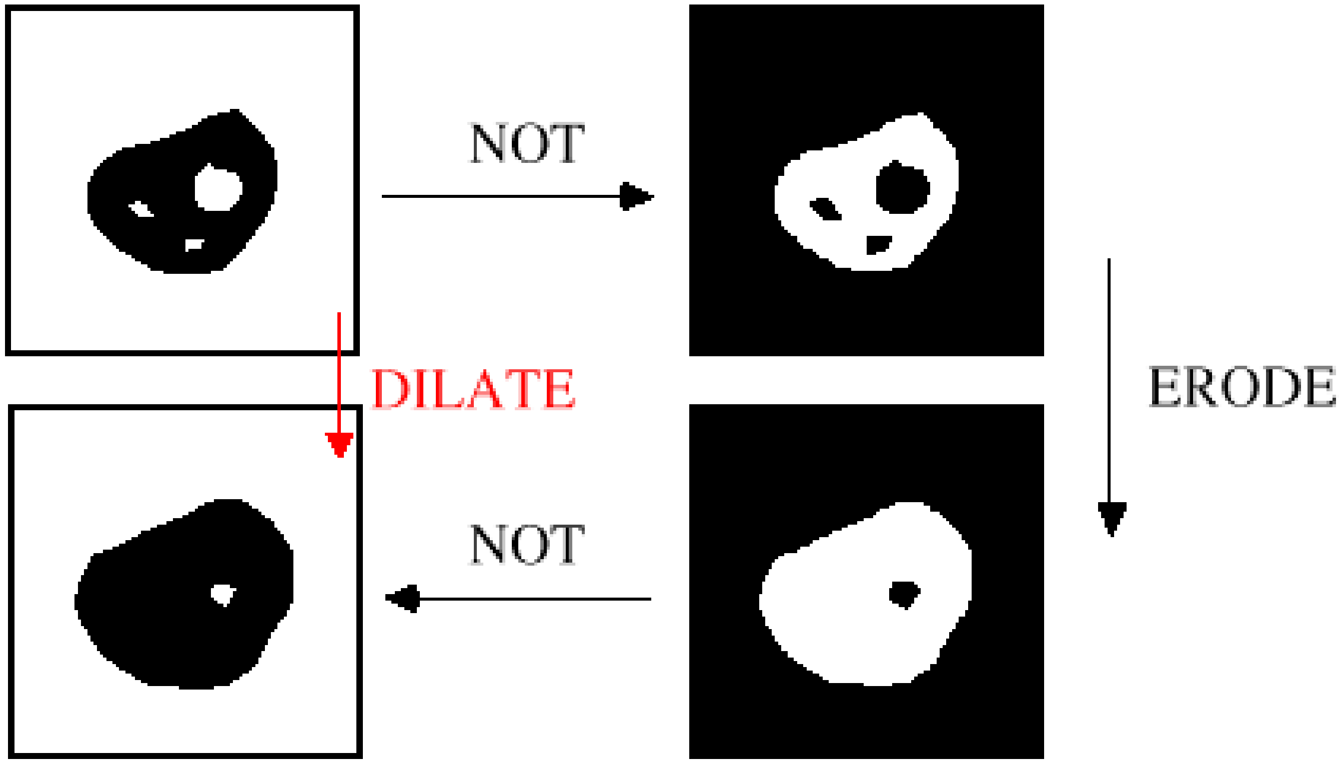


- Dilation also removes gaps or bays of too-narrow width:



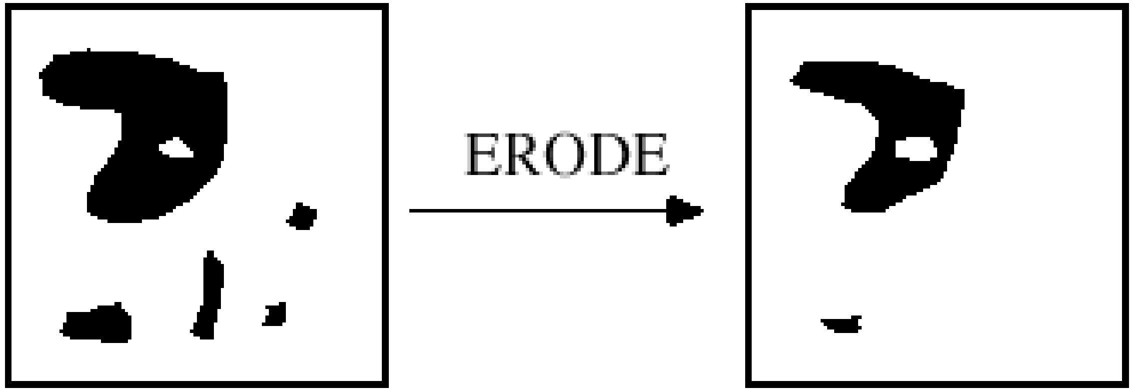
QUALITATIVE PROPERTIES OF DILATION

- Dilation of the BLACK part of an image is the same as erosion of the WHITE part!

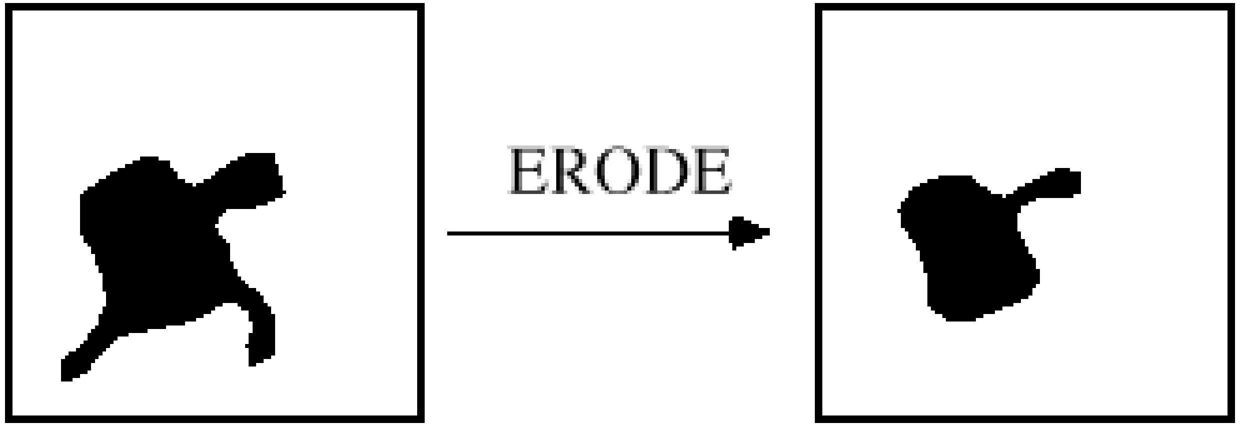


QUALITATIVE PROPERTIES OF EROSION

- Erosion removes objects of too-small size:

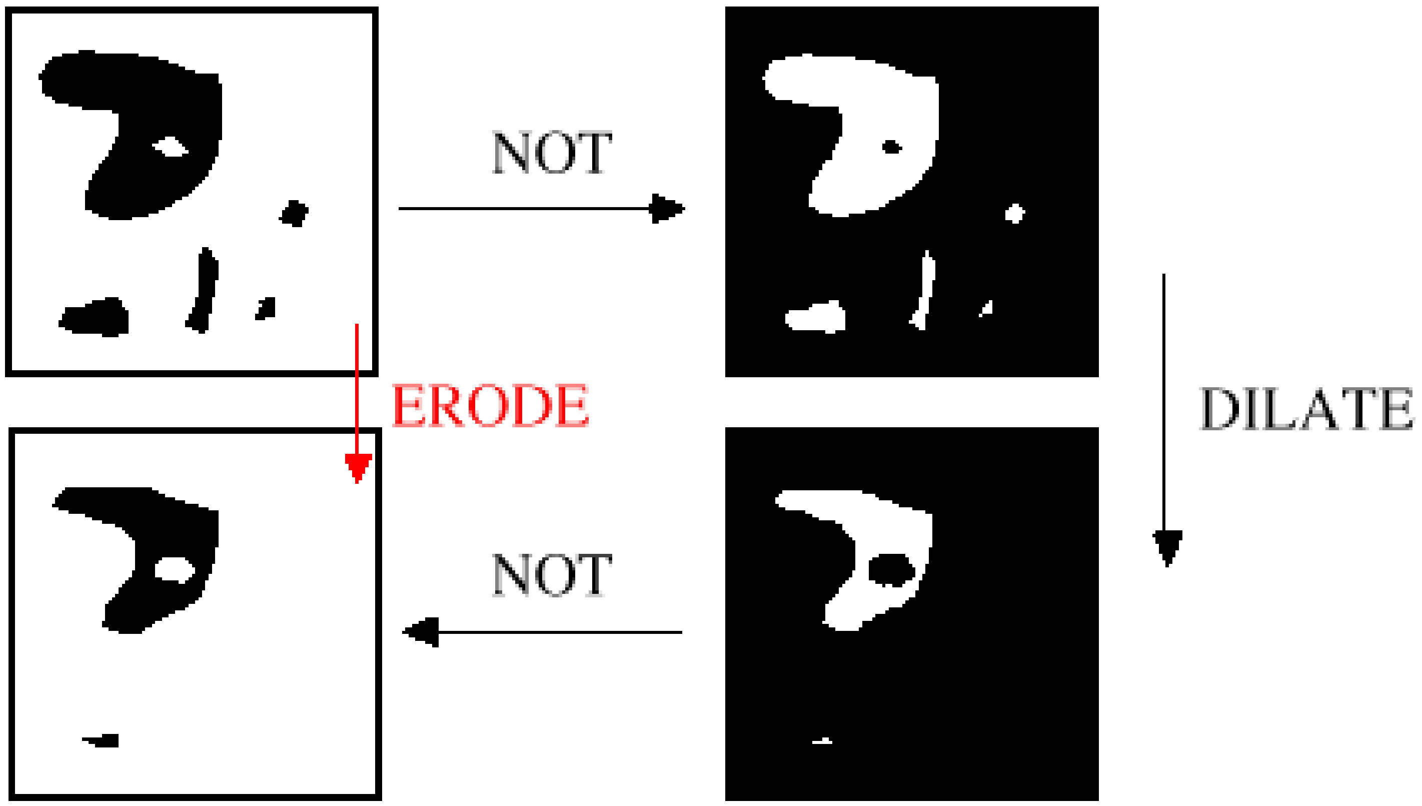


- Erosion also removes peninsulas of too-narrow width:



QUALITATIVE PROPERTIES OF EROSION

- Erosion of the BLACK part of an image is the same as dilation of the WHITE part!

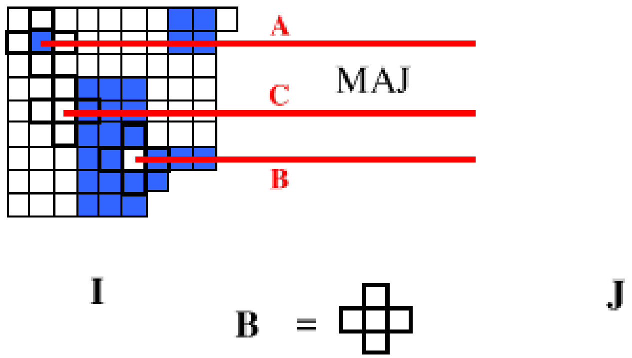


RELATING EROSION AND DILATION

- Erosion and dilation are actually the same operation - they are just **dual** operations with respect to **complementation**
- Erosion and dilation are only **approximate** inverses of one another
- Dilating an eroded image rarely yields the original image
- In particular, dilation cannot
  - Recreate peninsulas eliminated by erosion
  - Recreate small objects eliminated by erosion
- Eroding a dilated image rarely yields the original image
- In particular, erosion cannot
  - Unfill holes filled by dilation
  - Recreate gaps or bays filled by dilation

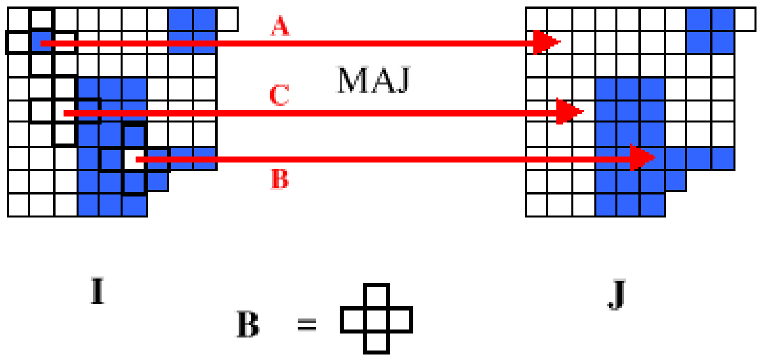
MEDIAN

- Actually **majority**. A special case of the gray-level **median filter**
- Possesses qualitative attributes of both dilation and erosion, but does not generally change the **size** of objects or background
- Local Computation: **J** = MEDIAN(**I**, **B**)



MEDIAN

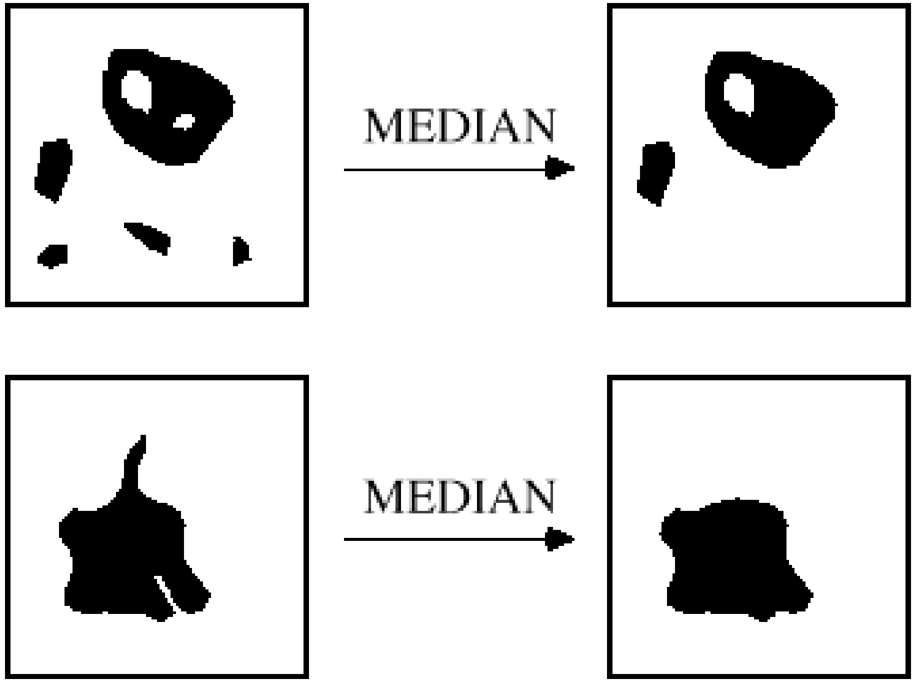
- Actually **majority**. A special case of the gray-level **median filter**
- Possesses qualitative attributes of both dilation and erosion, but does not generally change the **size** of objects or background
- Local Computation: **J** = MEDIAN(**I**, **B**)



- The median removed the small **object** **A** and the small **hole** **B**, but did not change the boundary (**size**) of the larger region **C**

QUALITATIVE PROPERTIES OF MEDIAN

- Median removes both **objects** and **holes** of **too-small** size, as well as both **gaps (bays)** and **peninsulas** of **too-narrow** width



QUALITATIVE PROPERTIES OF MEDIAN

- Note that median does not generally change the **size** of objects (although it does alter them)
- Median **is its own dual**, since

  MEDIAN [NOT(**I**)] = NOT [MEDIAN(**I**)]
- Thus, the median is a **shape smoother**. It is a **filter**
- We can define other shape smoothers as well.

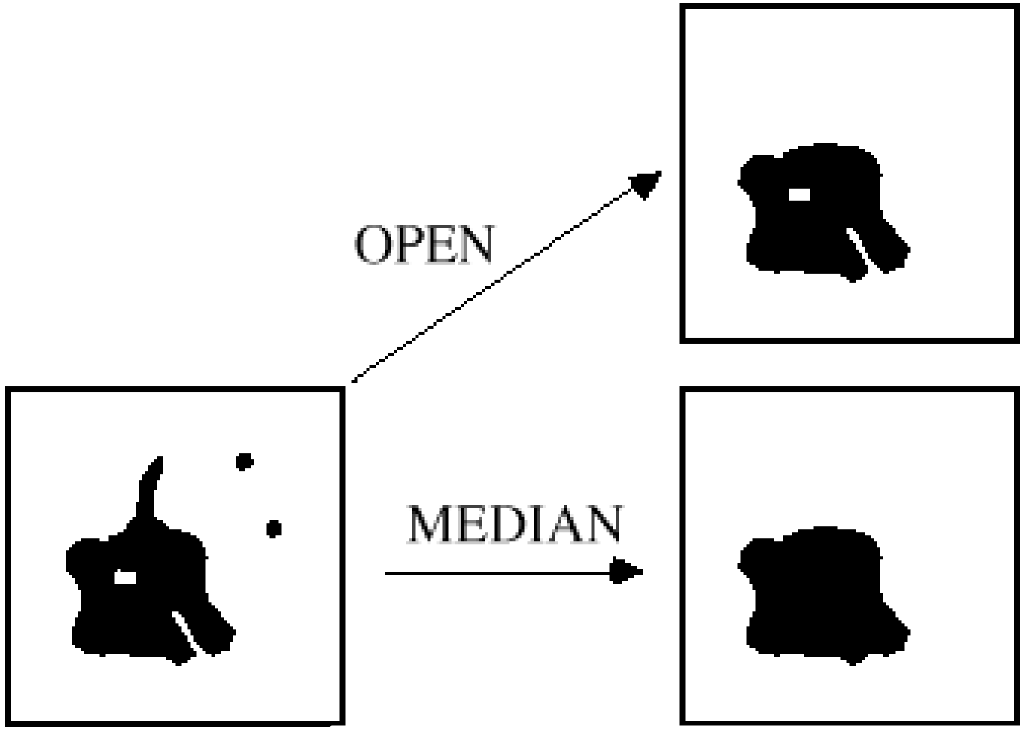
OPENing

- We can define **new** morphological operations by performing the basic ones in sequence
- Given an image **I** and window **B**, define

  OPEN(**I**, **B**) = DILATE [ERODE(**I**, **B**), **B**]
- In other words,

  OPEN = erosion (by **B**) followed by dilation (by **B**)

EXAMPLES



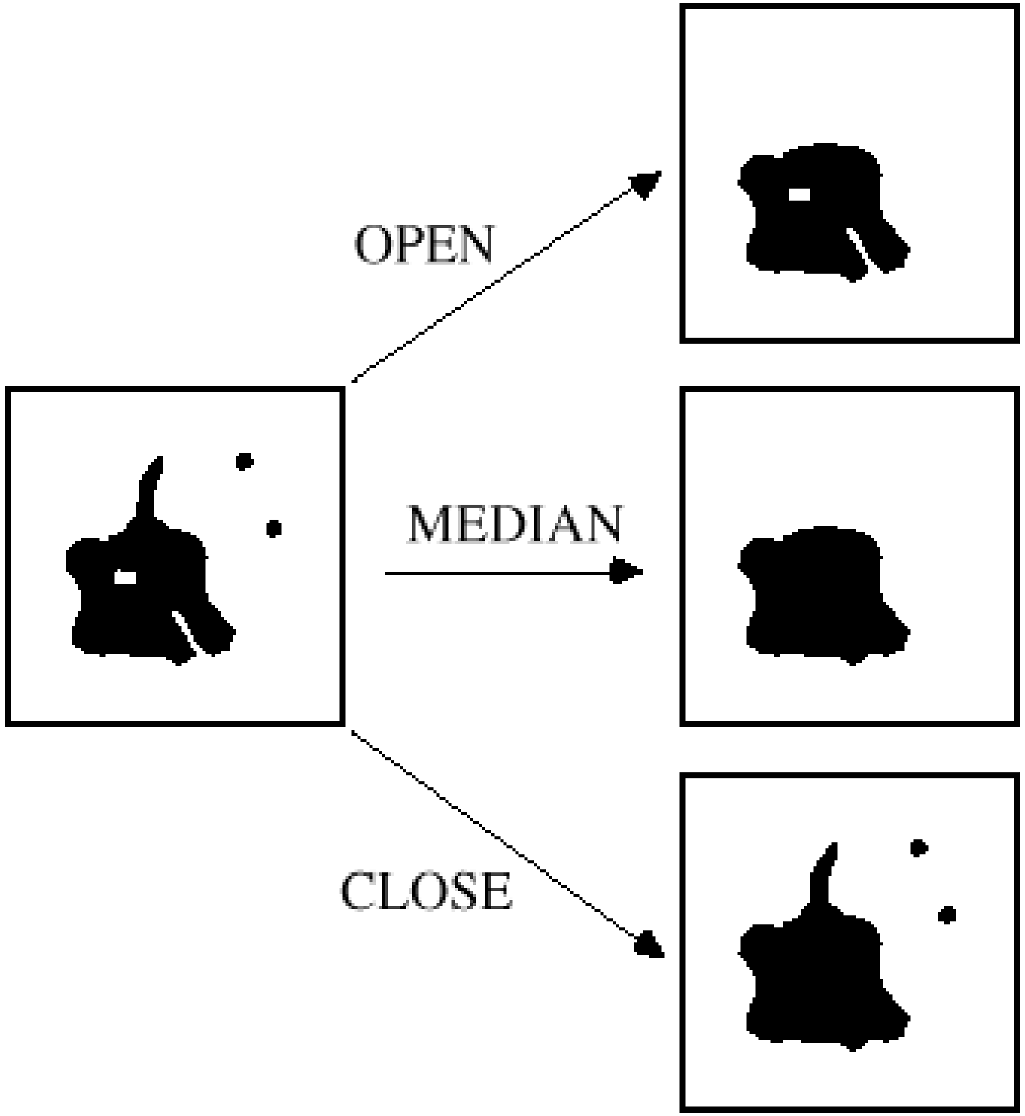
OPENing and CLOSing

- We can define **new** morphological operations by performing the basic ones in sequence
- Given an image **I** and window **B**, define

  OPEN(**I**, **B**) = DILATE [ERODE(**I**, **B**), **B**]

  CLOSE(**I**, **B**) = ERODE [DILATE(**I**, **B**), **B**]
- In other words,
- OPEN = erosion (by **B**) followed by dilation (by **B**)
- CLOSE = dilation (by **B**) followed by erosion (by **B**)

EXAMPLES



OPENing and CLOSing

- OPEN and CLOSE are very similar to MEDIAN:
- OPEN **removes too-small objects**/fingers (more effectively than MEDIAN), but not holes, gaps, or bays
- CLOSE **removes too-small holes**/gaps (more effectively than MEDIAN) but not objects or peninsulas
- OPEN and CLOSE generally **do not affect object size**
- OPEN and CLOSE are used when too-small BLACK and WHITE objects (respectively) are to be removed
- Thus OPEN and CLOSE are more specialized **smoothers**

Open-Close and Close-Open

- Very effective smoothers can be obtained by sequencing the OPEN and CLOSE operators:
- For an image **I** and structuring element **B**, define

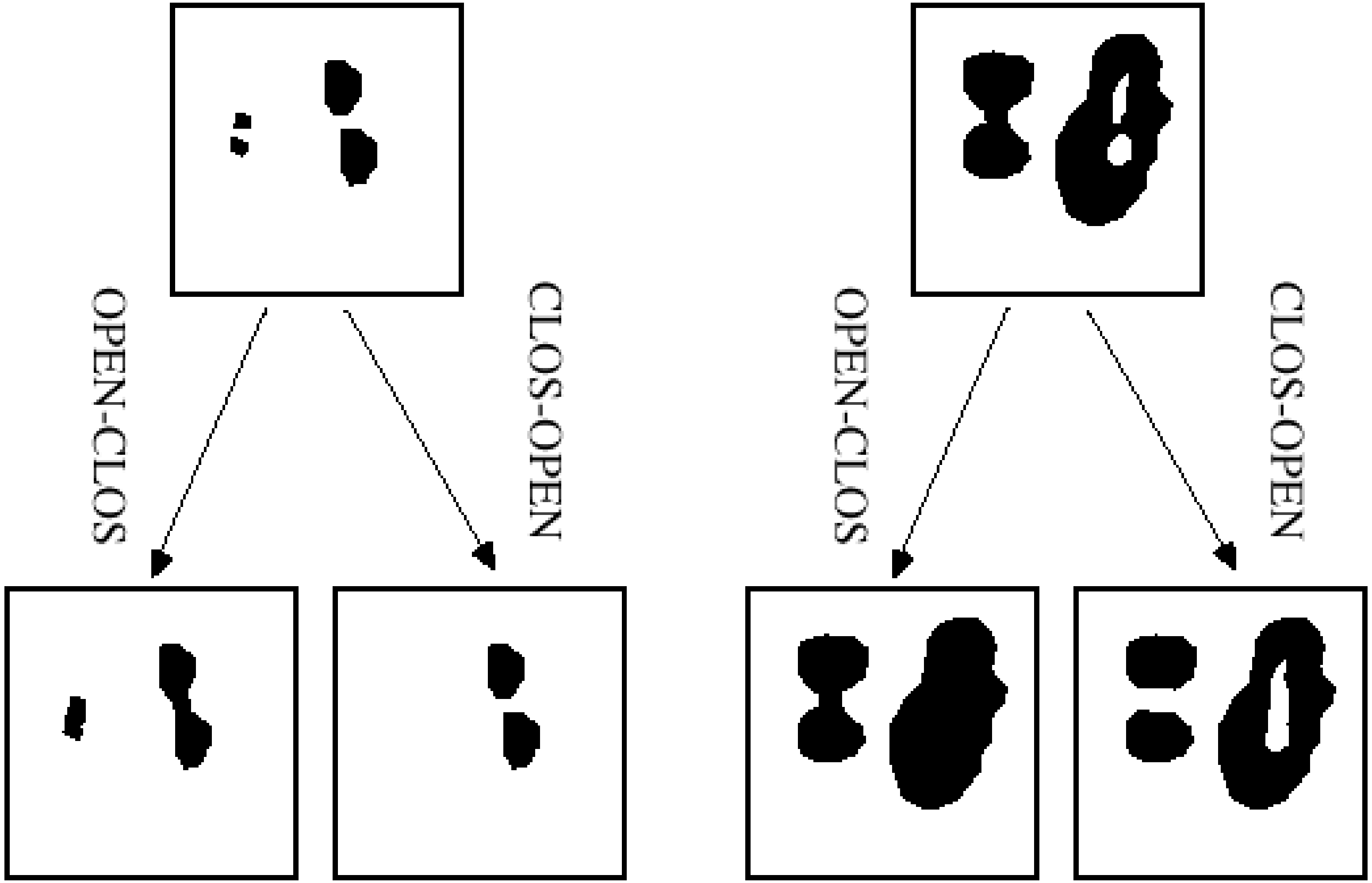
  OPEN-CLOS(**I**, **B**) = OPEN [CLOSE (**I**, **B**), **B**]

  CLOS-OPEN(**I**, **B**) = CLOSE [OPEN (**I**, **B**), **B**]
- These operations are quite similar (not mathematically identical)

Open-Close and Close-Open

- Both remove too-small structures without affecting size much
- Both are similar to the median filter except they smooth **more** (for a given structuring element **B**)
- One notable difference between OPEN-CLOS and CLOS-OPEN:
- OPEN-CLOS tends to **link neighboring objects together**
- CLOS-OPEN tends to **link neighboring holes together**

EXAMPLES

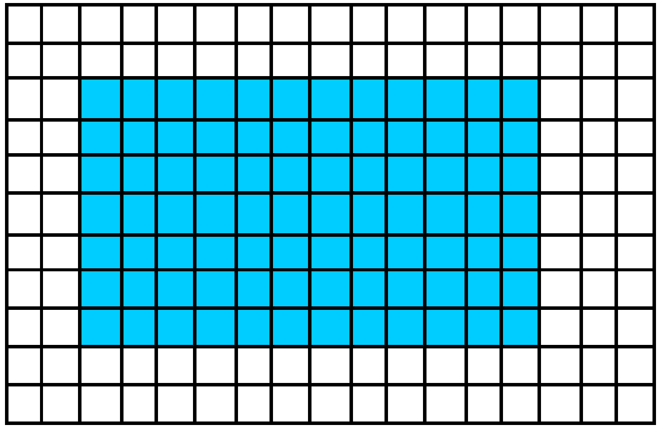


SKELETONIZATION

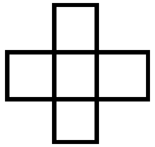
- A way of obtaining an image’s **medial axis** or **skeleton**

EXAMPLE

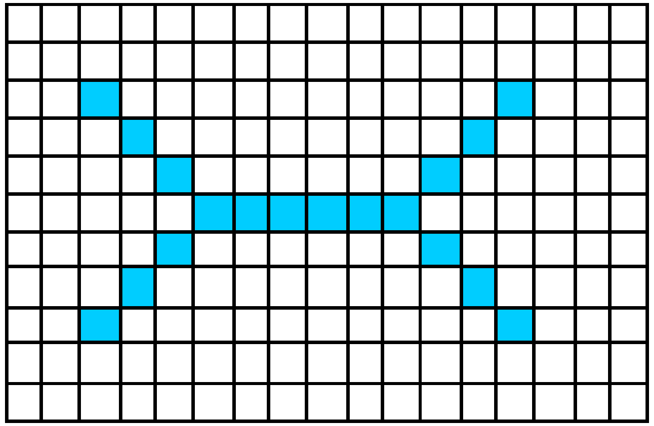
- Image $\mathbf{I}_0$:



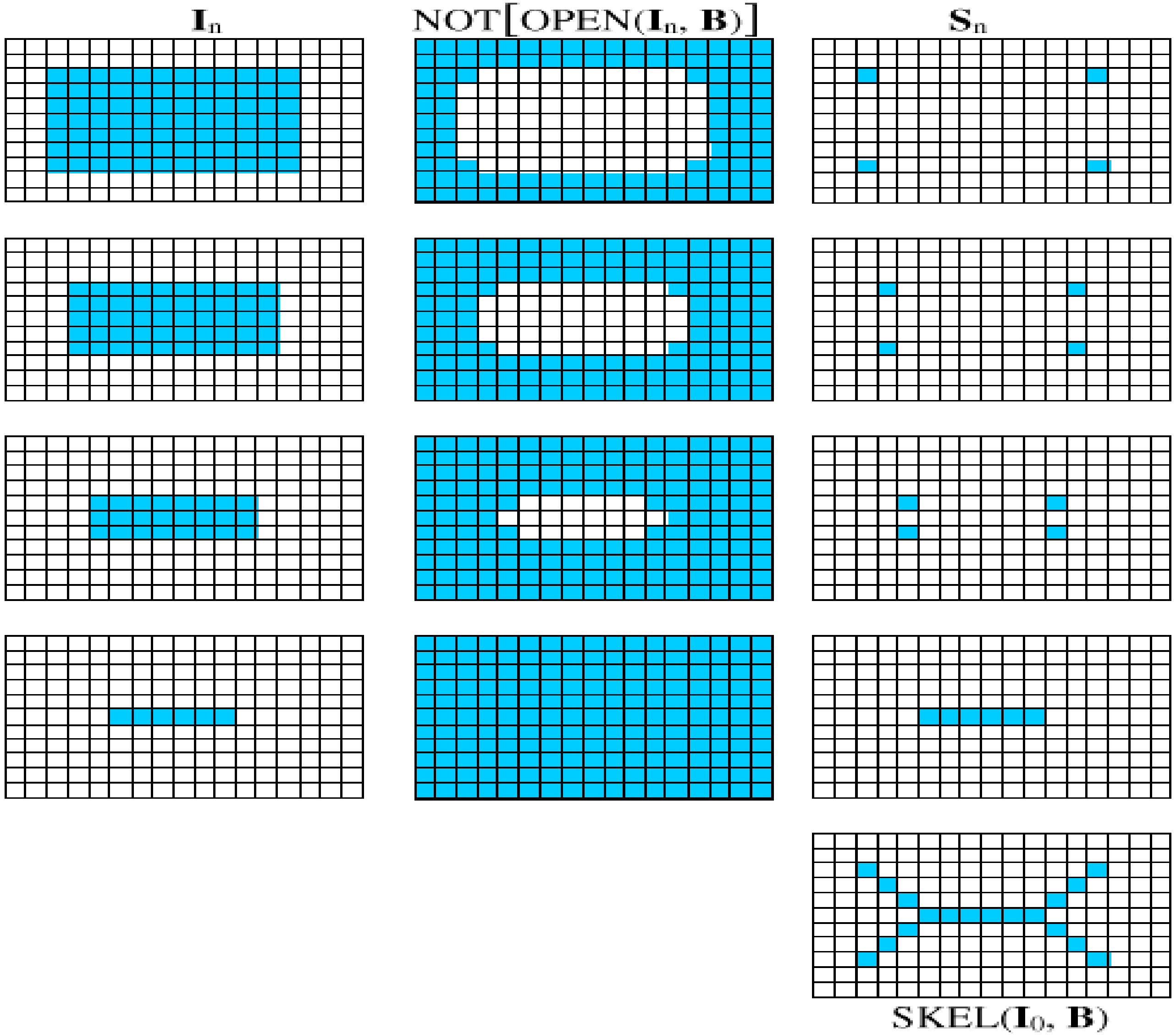
- Structuring Element **B**:



- SKEL($\mathbf{I}_0$, **B**):



THE STEPS

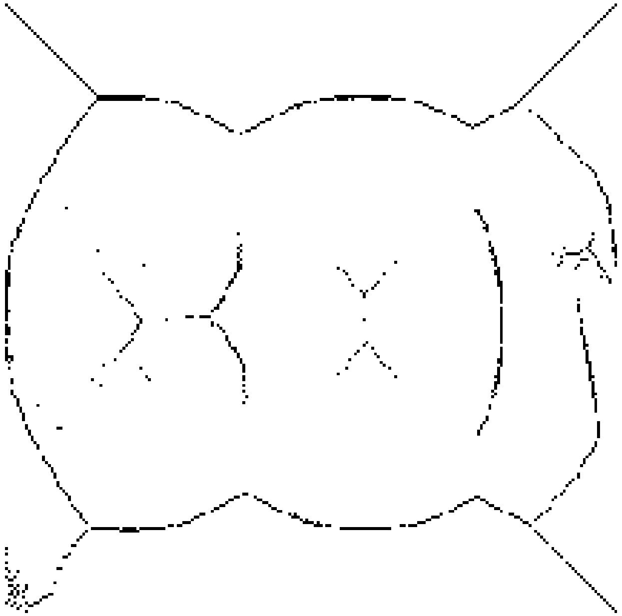

$\mathbf{I}_n$
NOT[OPEN($\mathbf{I}_n$, $\mathbf{B}$)]
$\mathbf{S}_n$
SKEL($\mathbf{I}_0$, $\mathbf{B}$)

SKELETONIZATION

- A way of obtaining an image's **medial axis** or **skeleton**
- Given an image $\mathbf{I}_0$ and window $\mathbf{B}$, the skeleton is SKEL($\mathbf{I}_0$, $\mathbf{B}$)
- Obtaining the skeleton requires a fairly complex iteration:
- Define $\mathbf{I}_n$ = ERODE [· · · ERODE [ERODE($\mathbf{I}_0$, $\mathbf{B}$), $\mathbf{B}$], · · · $\mathbf{B}$] (n consecutive EROSIONS of $\mathbf{I}_0$ by $\mathbf{B}$)
- $N = \max \{ n: \mathbf{I}_n \bullet \phi \}$ ϕ = empty set
- (the largest number of erosions before $\mathbf{I}_n$ "disappears")
- $\mathbf{S}_n = \mathbf{I}_n \wedge \mathrm{NOT}[\mathrm{OPEN}(\mathbf{I}_n, \mathbf{B})]$
- Then $\mathrm{SKEL}(\mathbf{I}_0, \mathbf{B}) = \mathbf{S}_1 \vee \mathbf{S}_2 \vee \cdots \vee \mathbf{S}_N$

EXAMPLE

binary image



skeleton (of background)

APPLICATION EXAMPLE

- **Simple Task: Measuring Cell Area**
- Simple processing steps:
  - (i) Find general cell region by **simple thresholding**
  - (ii) Apply region correction techniques:
    - Blob coloring
    - Minor region removal
    - CLOS-OPEN
  - (iii) Display cell boundary for operator verification
  - (iv) Compute image cell area by counting pixels
  - (v) Compute actual cell area using perspective projection

COMMENTS

- Previous manual measurement techniques required > 1 hour per cell image to analyze
- Algorithm runs in less than a second.
- Published in CRC Press's *Image Analysis in Biology* as the standard for "Automated Area Measurement."

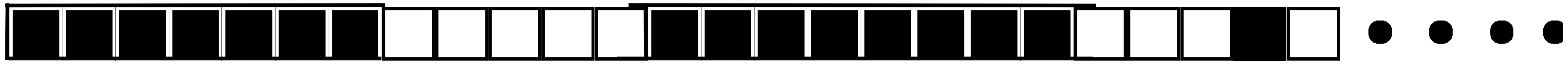
Compression: RUN LENGTH CODING

- The number of bits required to store an N x N binary image is N^2
- This can be significantly reduced in many cases.
- Run-length coding works well if the WHITE and BLACK regions are generally not small.

EXAMPLE

what's
stored:

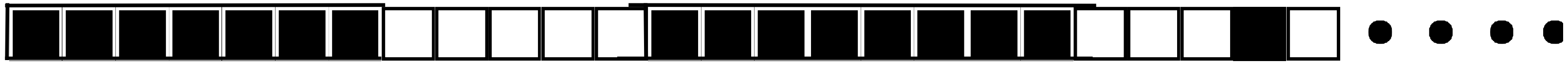
row m



EXAMPLE

what's
stored: '1' 7 5 8 3 1

row m



HOW DOES IT WORK?

- Binary images are stored (or transmitted) on a line-by-line (row-by-row) basis
- For each image row numbered *m*:
  - Store the first pixel value ('0' or '1') in row *m* as a reference
  - Set **run counter** *c* = 1
  - For each pixel in the row:
    - Examine the next pixel to the right
    - If same as current pixel, set *c* = *c* + 1
    - If different from current pixel, **store *c*** and set *c* = 1
    - Continue until end of row is reached
- Each run-length is stored using b bits.

COMMENTS

- Can yield excellent **lossless compressions** on some images.
- This will happen if the image contains lots of runs of 1's and 0's.
- If the image contains only very short runs, then run-length coding can actually **increase** the required storage.

WORST CASE

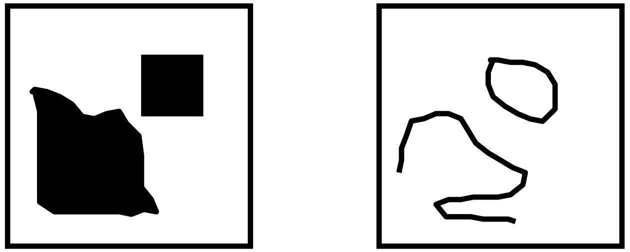
what's
stored: '1'1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
row m 

- In this worst-case example the storage increases **b-fold**!
- Rule of thumb: the average run-length L should satisfy:

$$L > b.$$

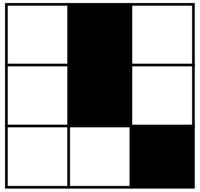
CONTOUR REPRESENTATION & CHAIN CODING

- We can distinguish between two general types of binary image: **region images** and **contour images.**



region image   contour image

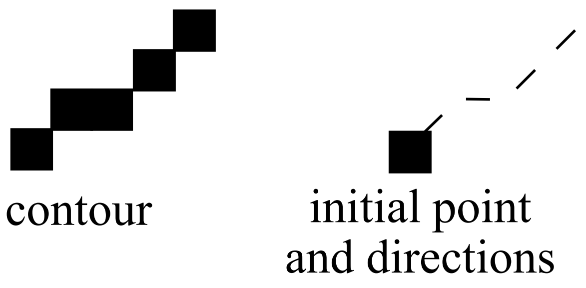
- We will require contour images to be special:
- Each BLACK pixel in a contour image must have **at most** two BLACK 8-neighbors
- a BLACK pixel and its 8-neighbors –



- **Contour images** are composed only of **single-pixel width** contours (straight or curved) and single points.

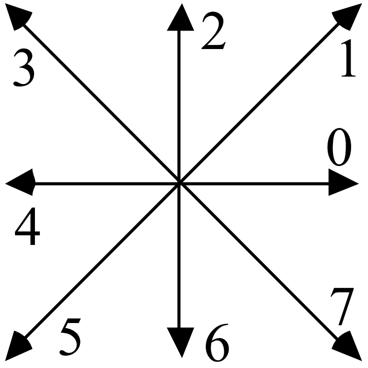
CHAIN CODE

- The chain code is a highly efficient method for **coding contours**
- Observe that if the **initial (i, j) coordinate** of an 8-connected contour is known, then the rest of the contour can be coded by giving the **directions** along which the contour propagates



CHAIN CODE

- We use the following 8-neighbor direction codes:

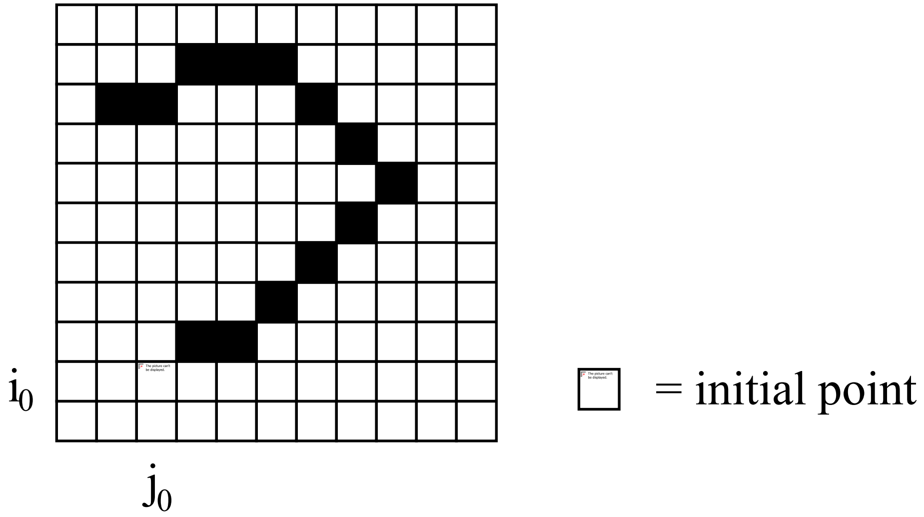


- Since the numbers 0, 1, 2, 3, 4, 5, 6, 7 can be coded by their 3-bit binary equivalents:

  000, 001, 010, 011, 100, 101, 110, 111

  the location of each point on the contour **after** the initial point can be coded by 3 bits.

EXAMPLE



- Its chain code: (after recording the initial coordinate (i0, j0)

1, 0, 1, 1, 1, 1, 3, 3, 3, 4, 4, 5, 4

=

001, 000, 001, 001, 001, 001, 011, 011, 011, 100, 100, 101, 100

COMMENTS

- The **compression** obtained can be quite significant: coding the contour by M-bit coordinates (M = 9 for 512 x 512 images) requires 6 times as much storage
- The technique is effective in many computer vision and pattern recognition applications, e.g. **character recognition**
- For closed contours, the initial coordinate can be chosen arbitrarily. If the contour is **open,** then it is usually an **end point** (one 8-neighbor).