Image Processing

主講人:虞台文

Content

- Smoothing
- Image Morphology
- Some Applications of Image Morphology
- Flood Fill
- Resize
- Image Pyramids
- Threshold

Image Processing

Smoothing

Smoothing

- Smoothing is also called blurring
- Why smoothing?
  - reduce noise and camera artifact
  - reduce the resolution of an image in a principled way

Smoothing

```
void cvSmooth(
  const CvArr* src,
  CvArr* dst,
  int smoothtype=CV_GAUSSIAN,
  int param1=3,
  int param2=0,
  double param3=0,
  double param4=0
);
```

Dependingsmo

smoothtype:

- CV_BLUR_NO_SCALE
- CV_BLUR
- CV_GAUSSIAN
- CV_MEDIAN
- CV_BILATERAL

Smoothing

Smooth type	Name	In place?	Nc	Depth of src	Depth of dst	Brief description
CV_BLUR	Simple blur	Yes	1,3	8u, 32f	8u, 32f	Sum over a `param1×param2` neighborhood with subsequent scaling by 1/(`param1×param2`).
CV_BLUR_NO_SCALE	Simple blur with no scaling	No	1	8u	16s (for 8u source) or 32f (for 32f source)	Sum over a `param1×param2` neighborhood.
CV_MEDIAN	Median blur	No	1,3	8u	8u	Find median over a `param1×param1` square neighborhood.
CV_GAUSSIAN	Gaussian blur	Yes	1,3	8u, 32f	8u (for 8u source) or 32f (for 32f source)	Sum over a `param1×param2` neighborhood.
CV_BILATERAL	Bilateral filter	No	1,3	8u	8u	Apply bilateral 3-by-3 filtering with color `sigma=param1` and a space `sigma=param2`.

Smoothing

Smooth type	Name	In place?	Nc	Depth of src	Depth of dst	Brief description
CV_BLUR	Simple blur	Yes	1,3	8u, 32f	8u, 32f	Sum over a param1×param2 neighborhood with subsequent scaling by 1/(param1×param2).
CV_BLUR_NO_SCALE	Simple blur with no scaling	No	1	8u	16s (for 8u source) or 32f (for 32f source)	Sum over a param1×param2 neighborhood.
CV_MEDIAN	Median blur	No	1,3	8u	8u	Find median over a param1×param1 square neighborhood.
CV_GAUSSIAN	Gaussian blur	Yes	1,3	8u, 32f	8u (for 8u source) or 32f (for 32f source)	Sum over a param1×param2 neighborhood.
CV_BILATERAL	Bilateral filter	No	1,3	8u	8u	Apply bilateral 3-by-3 filtering with color sigma=param1 and a space sigma=param2.

Example cvSmooth

Original

BLUR

param1=7
param2=7

GAUSSIAN

param1=7
param2=7

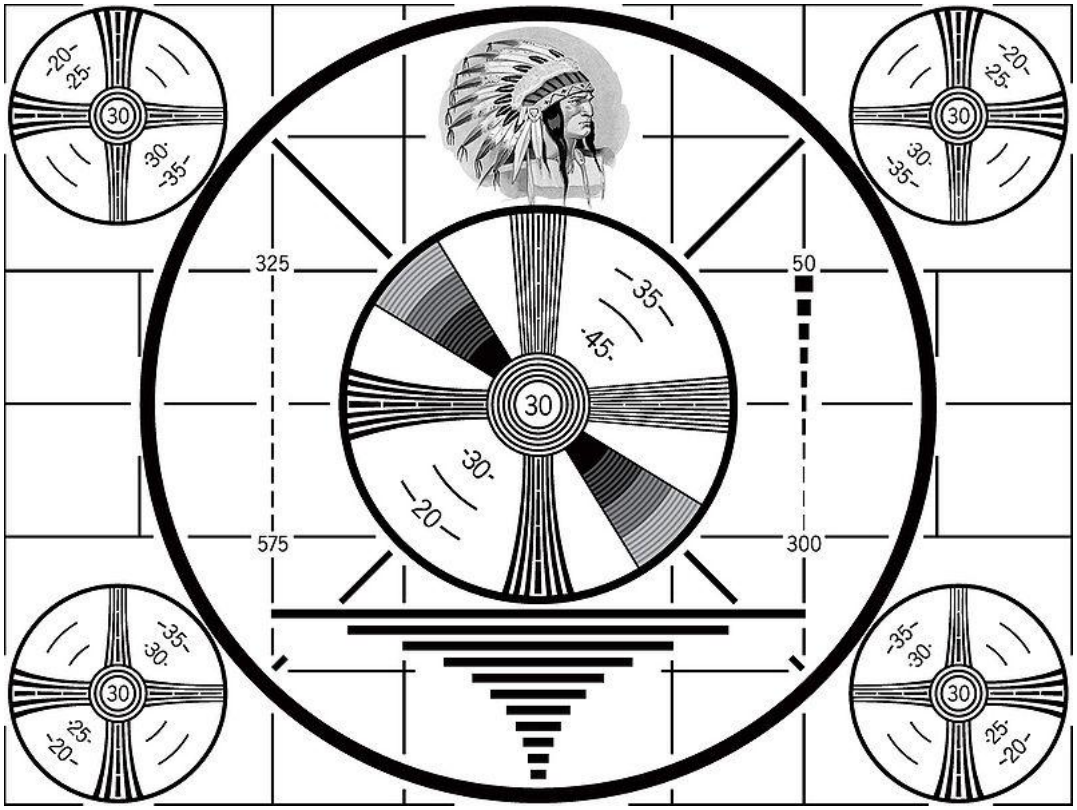
MEDIAN

param1=7

BILATERAL

param1=7
param3=50.0
Param4=3.5

Exercise



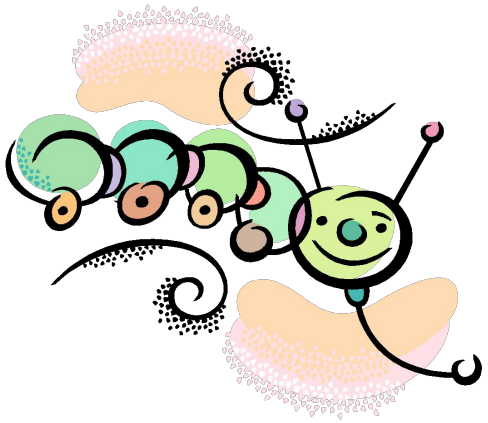
Download
Test Program

Image Processing

Image Morphology

Morphological Operations

- Morphological operations come from the word "morphing" in Biology which means "changing a shape".

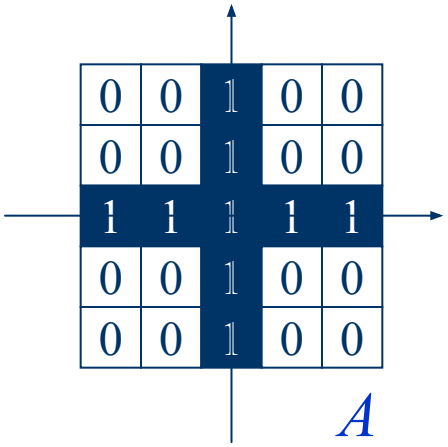


Morphing

Morphological Operations

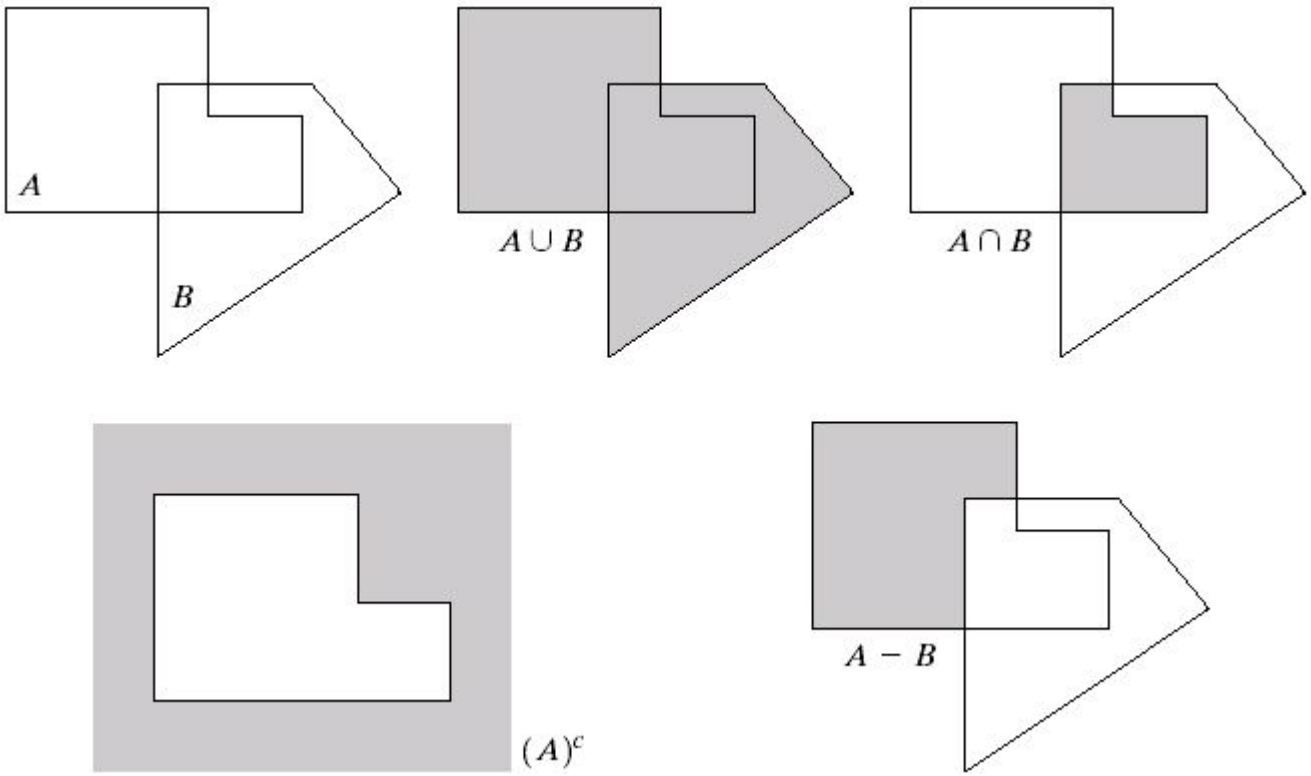
- Originated from set operations
- Can be used to manipulate object shapes such as thinning, thickening, filling, etc.

Binary Image Representation



$$A = \left\{ \begin{matrix} (-2,0),(-1,0),(0,0),(1,0),(2,0), \\ (0,2),(0,1),(0,-1),(0,-2) \end{matrix} \right\}$$

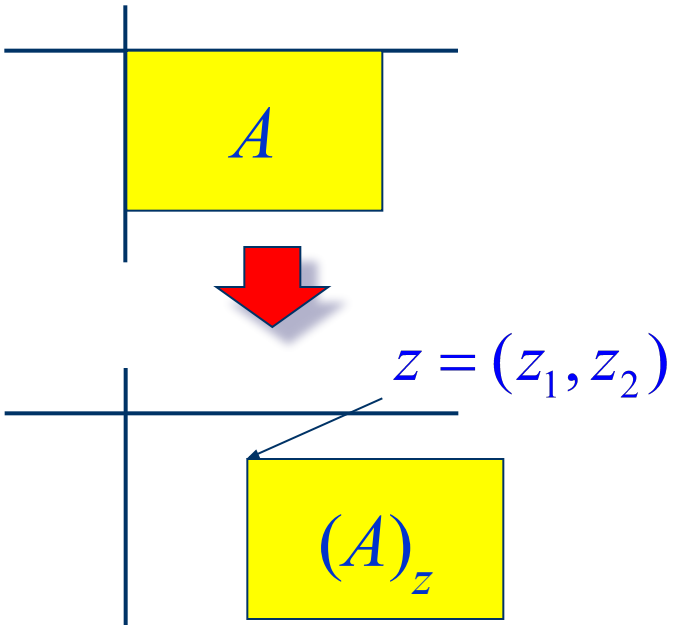
Basic Set Operations



Translation and Reflection Operations

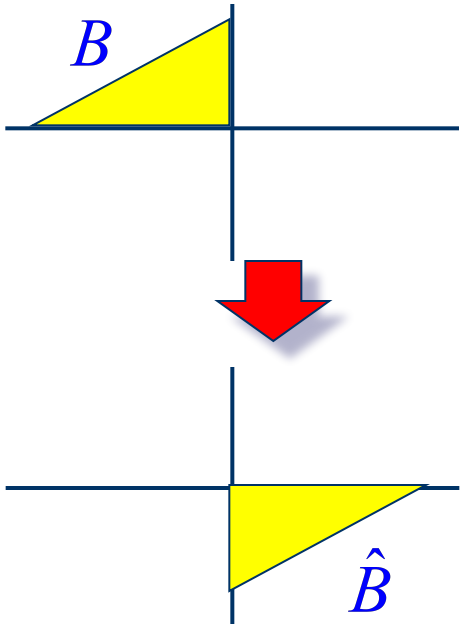
Translation

$$(A)_z = \{c \mid c = a + z, \quad \text{for } a \in A\}$$



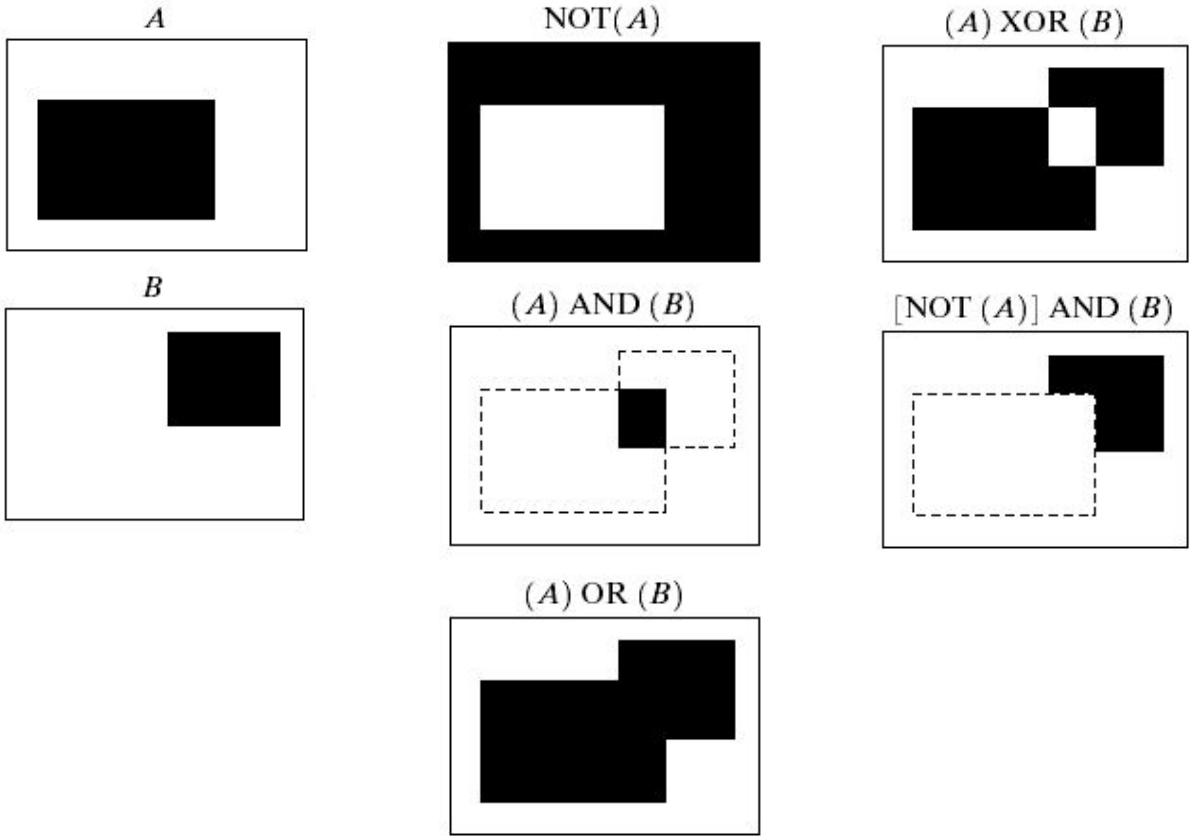
Reflection

$$\hat{B} = \{w \mid w = -b, \quad \text{for } b \in B\}$$



p	q	p AND q (also $p \cdot q$)	p OR q (also $p + q$)	NOT (p) (also $\bar{p}$)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Logical Operations

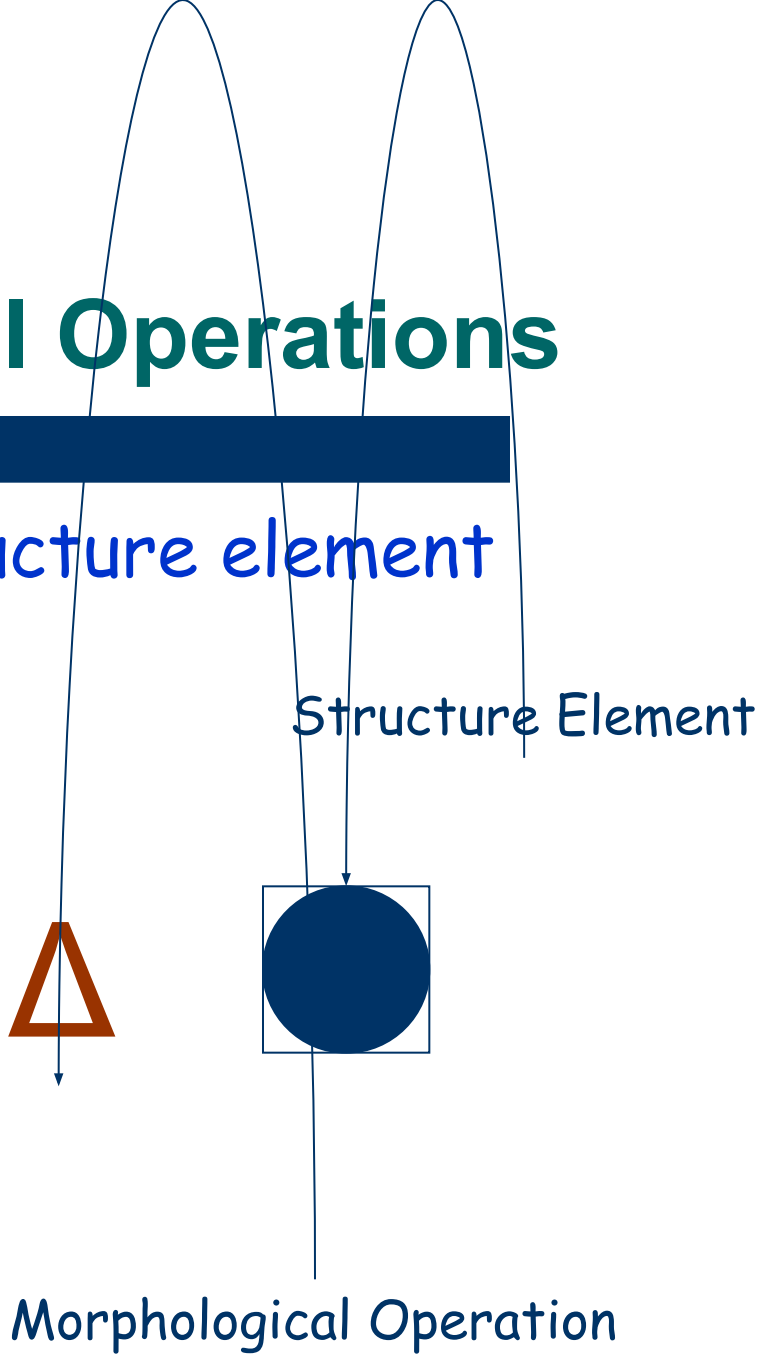


Basic Morphological Operations

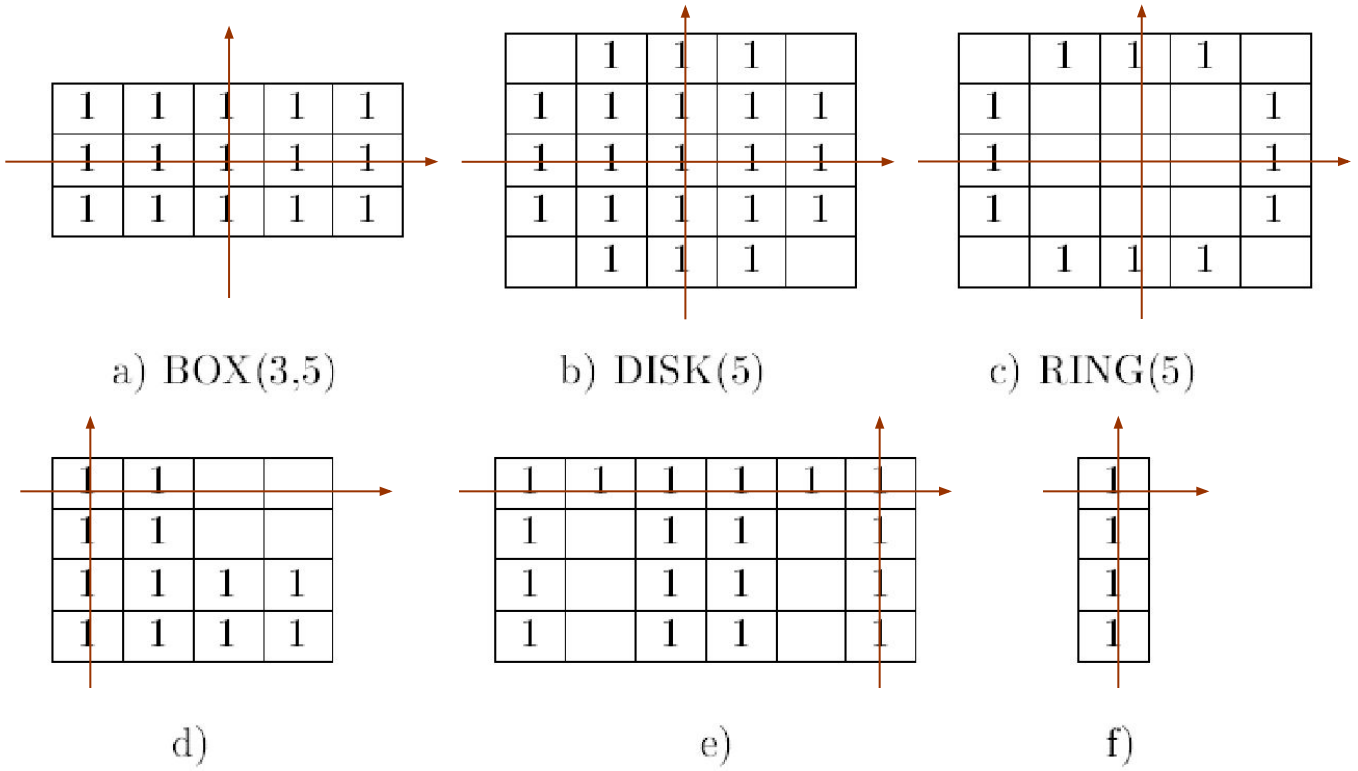
- Dilation
  - Enlarge a region
- Erosion
  - Shrink a region
- Opening
  - Get rid of small portions of region that jut out from the boundary into the background region
- Closing
  - Close up internal holes in a region and eliminate "bays" along the boundary

Basic Morphological Operations

- Defined based on a structure element

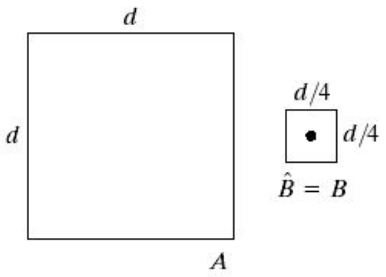


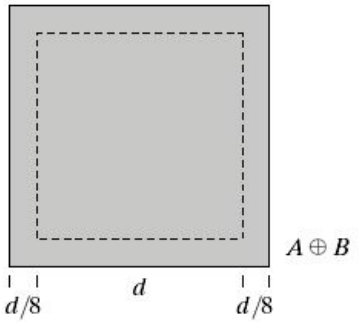
Structure Elements



Dilation $A \oplus B$

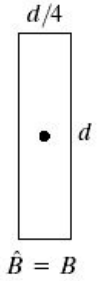
$$A \oplus B = \left\{ z \middle| \left(B \right)_z \cap A \neq \varnothing \right\}$$

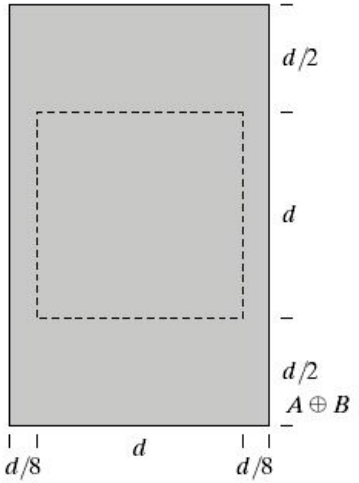




A = Object to be dilated

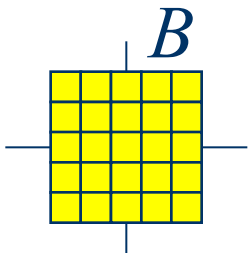
B = Structuring element

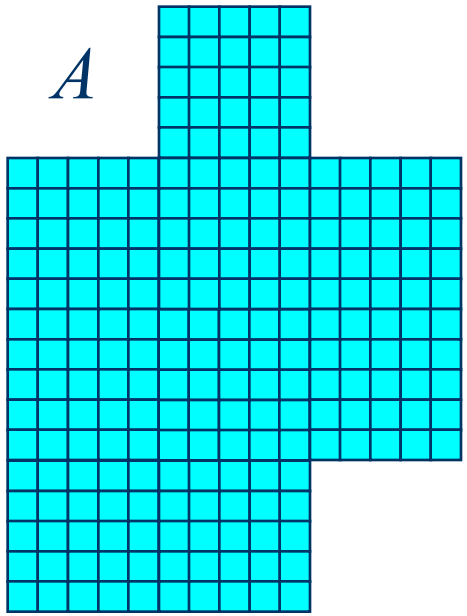


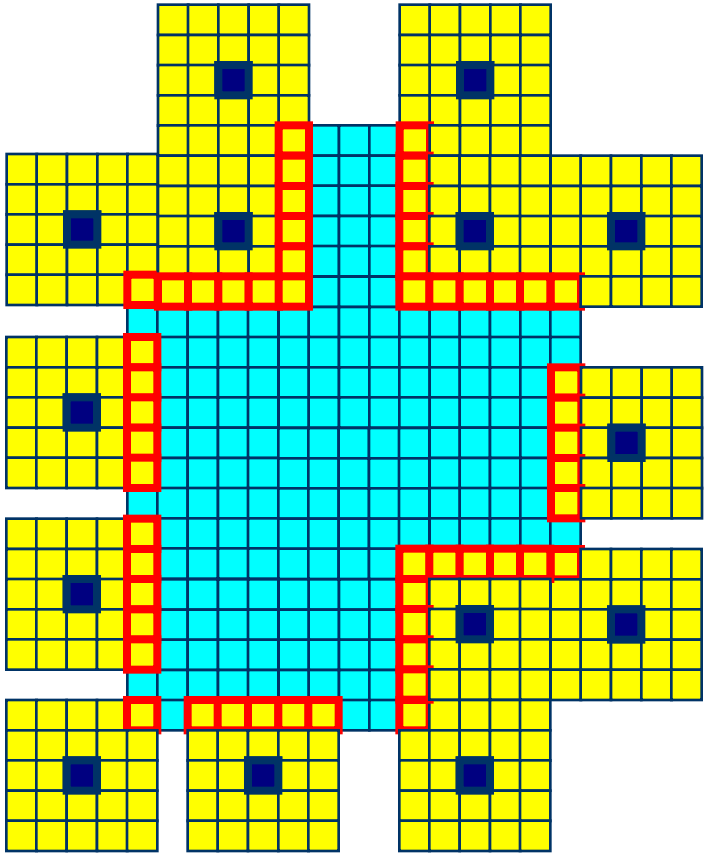


Dilation

$$A \oplus B = \left\{ z \middle| \left(B \right)_z \cap A \neq \varnothing \right\}$$

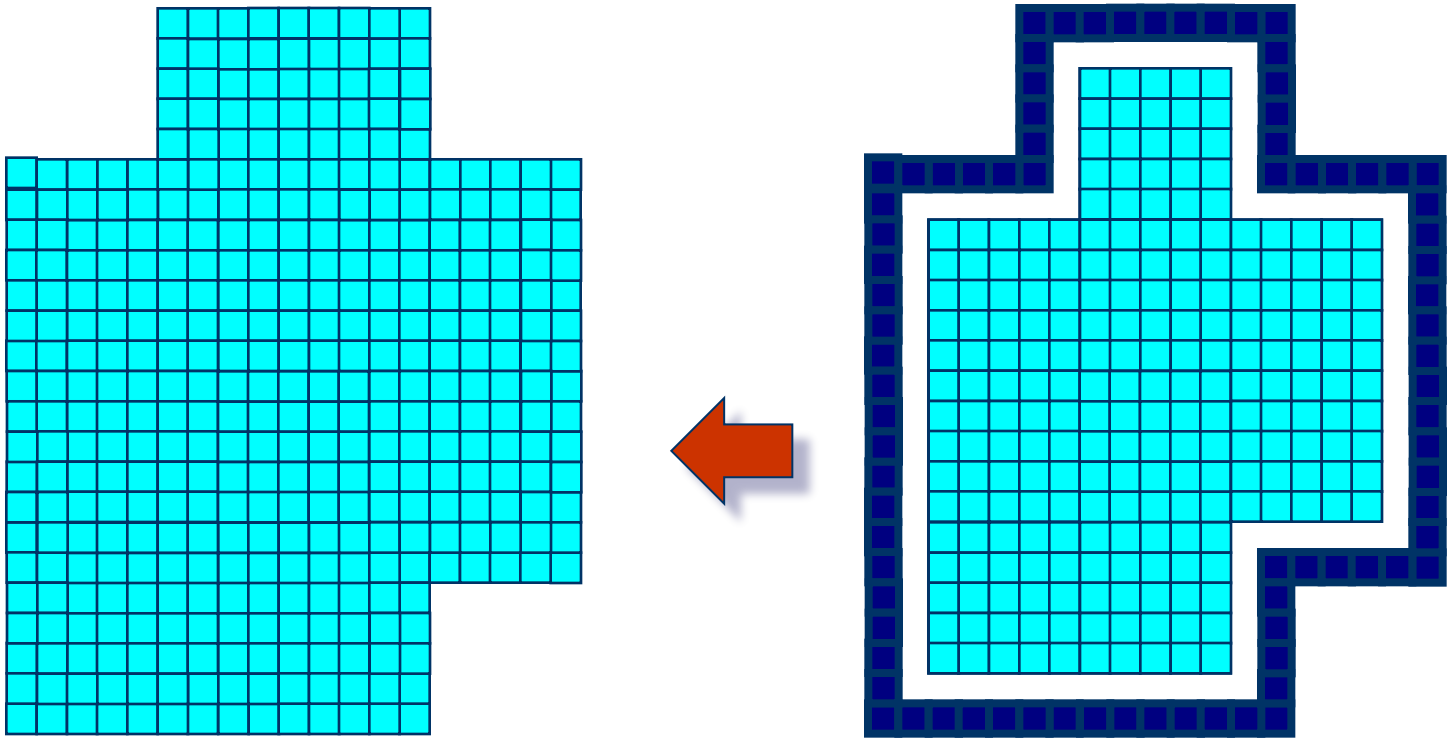






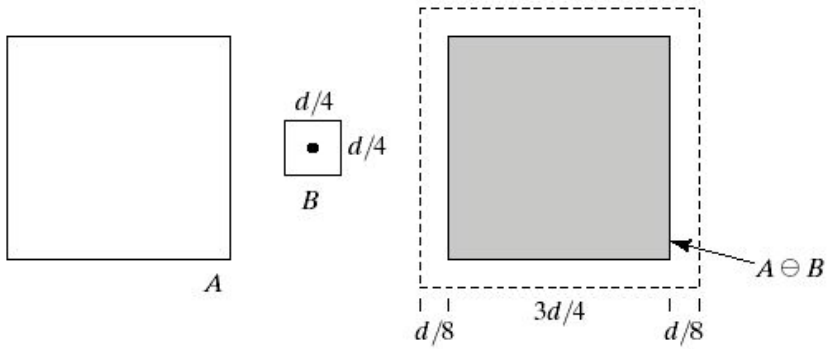
Dilation

$$A \oplus B = \left\{ z \middle| \left(B \right)_z \cap A \neq \varnothing \right\}$$

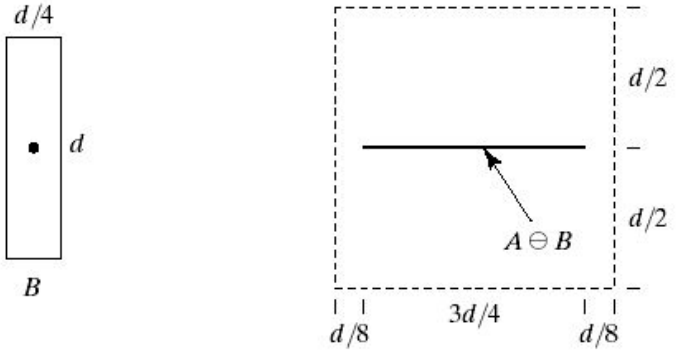


Erosion $A \ominus B$

$$A \ominus B = \left\{ z \middle| (B)_z \subseteq A \right\}$$

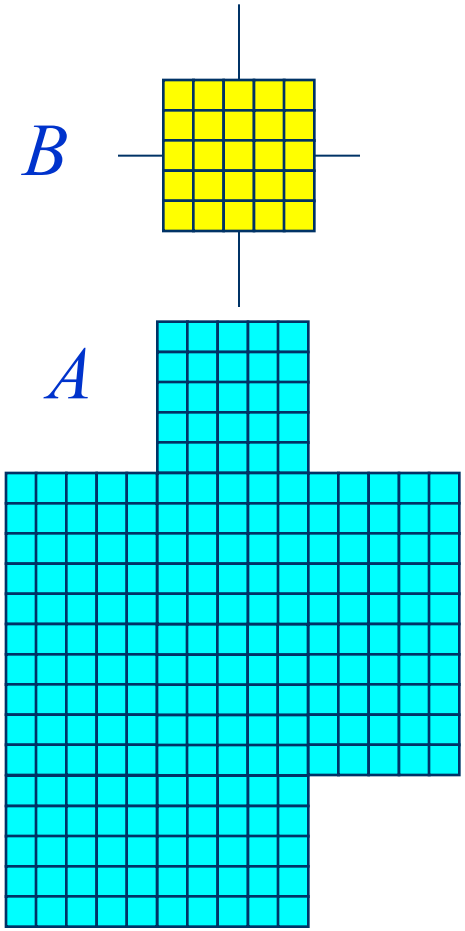


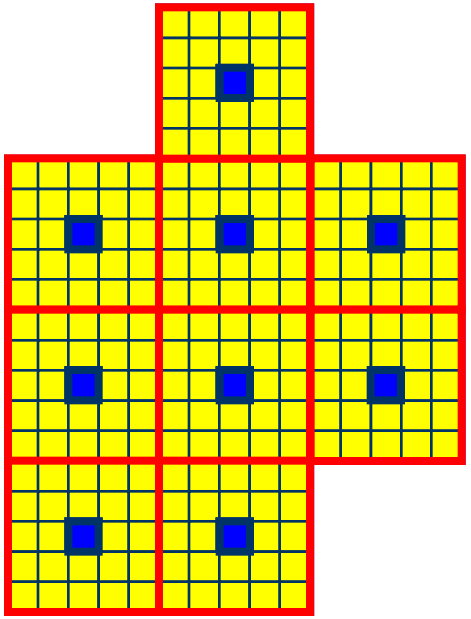
A = Object to be eroded
B = Structuring element



Erosion

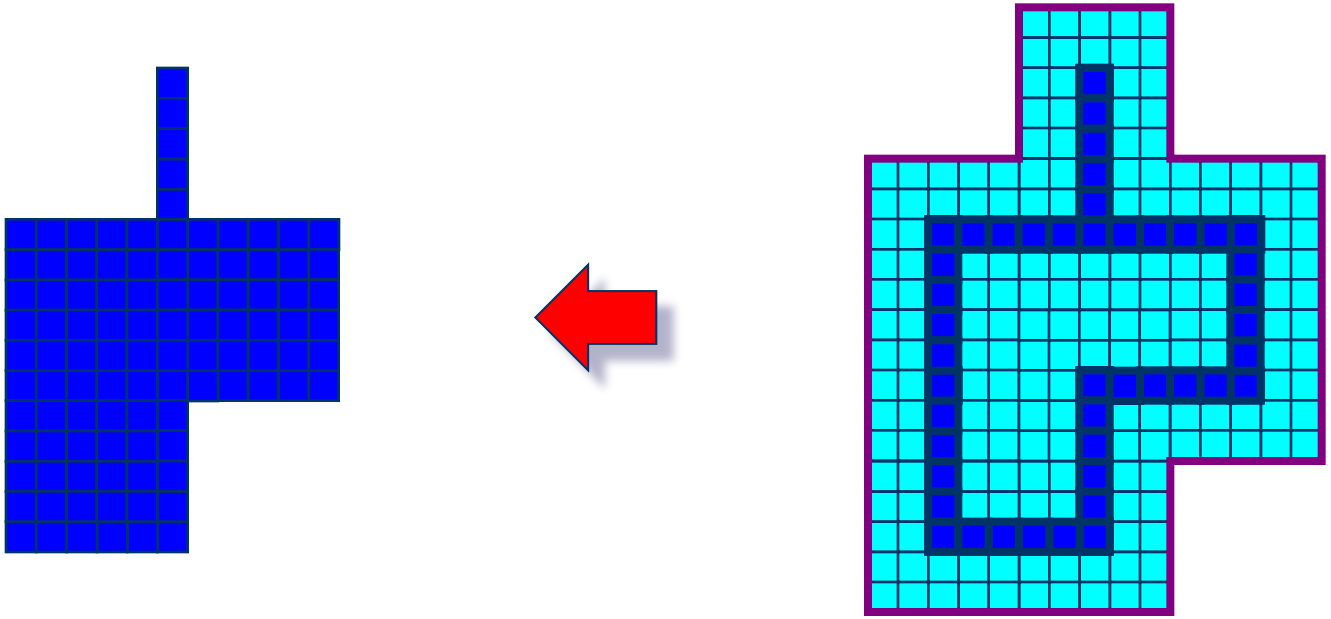
$$A \ominus B = \left\{ z \middle| (B)_z \subseteq A \right\}$$



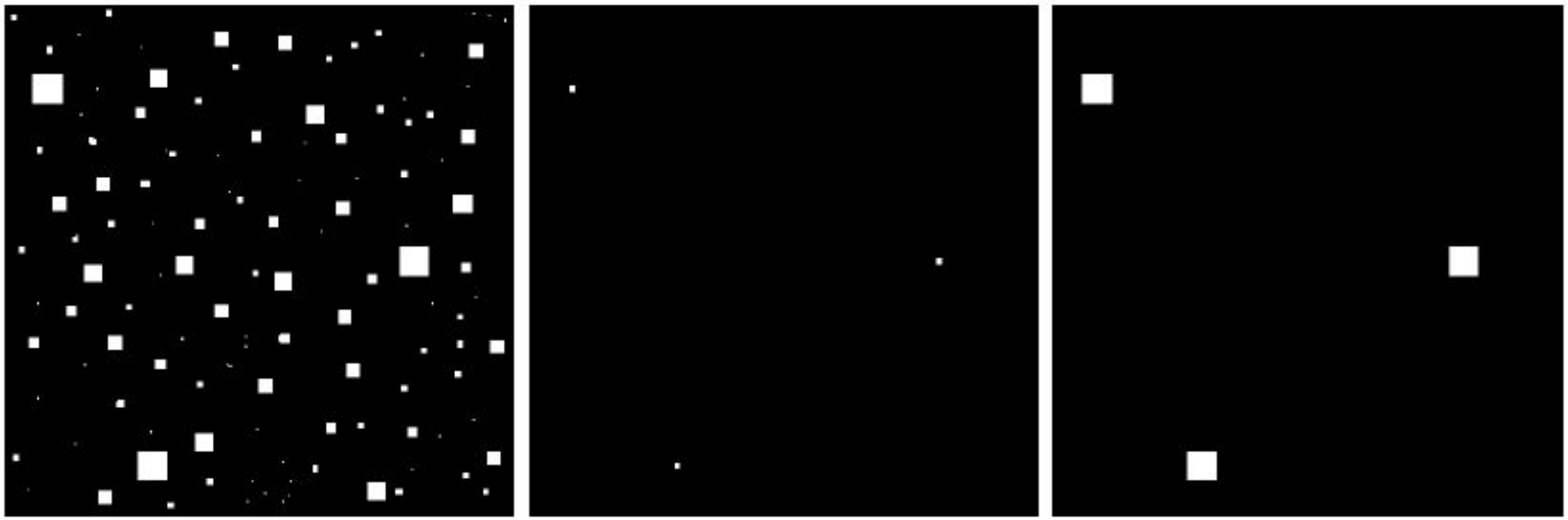


Erosion

$$A \ominus B = \left\{ z \middle| (B)_z \subseteq A \right\}$$



Example:
Application of Dilation and Erosion



a b c

FIGURE 9.7 (a) Image of squares of size 1, 3, 5, 7, 9, and 15 pixels on the side. (b) Erosion of (a) with a square structuring element of 1's, 13 pixels on the side. (c) Dilation of (b) with the same structuring element.

Remove small objects such as noise

Duality Between Dilation and Erosion

$$(A \boxplus B)^c = A^c \ominus B$$

Pf)

$$(A \boxplus B)^c = \left\{ z \middle| (B)_z \subseteq A \right\}^c$$

$$= \left\{ z \middle| (B)_z \cap A^c = \varnothing \right\}^c$$

$$= \left\{ z \middle| (B)_z \cap A^c \neq \varnothing \right\}$$

$$= A^c \ominus B$$

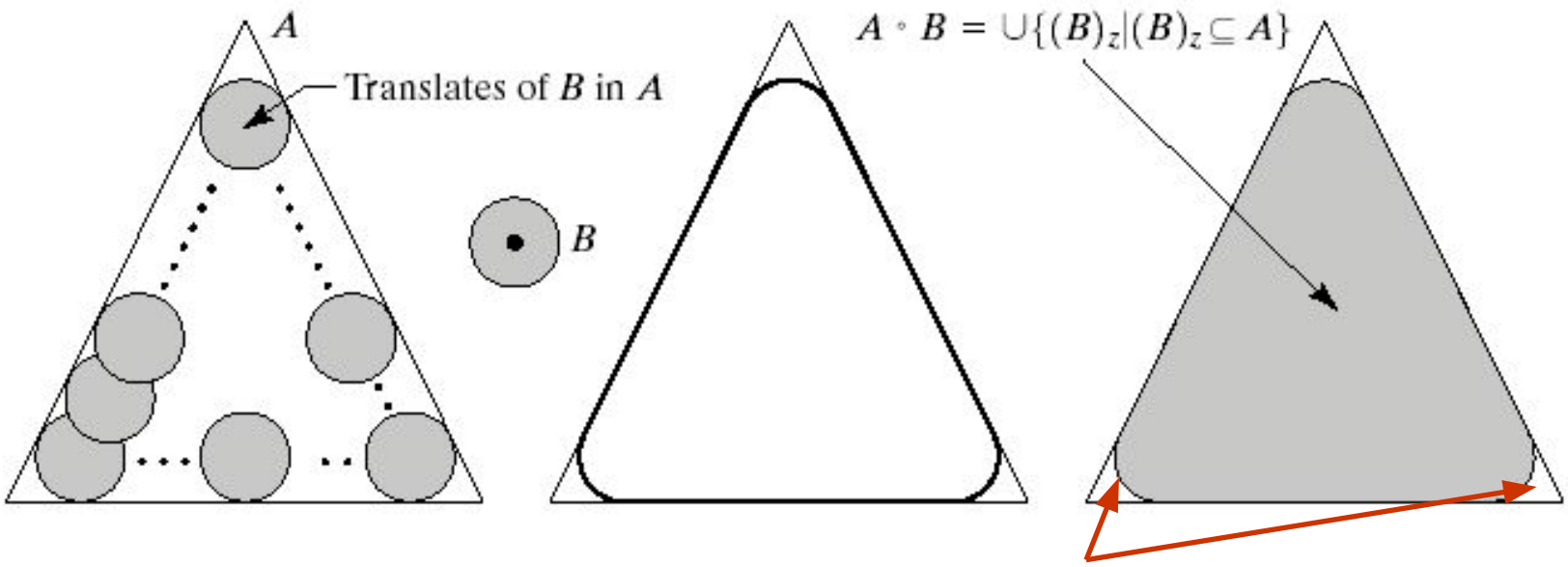
Opening

$$A \circ B = (A \ominus B) \oplus B$$

Opening

Combination of all parts of A that can completely contain B

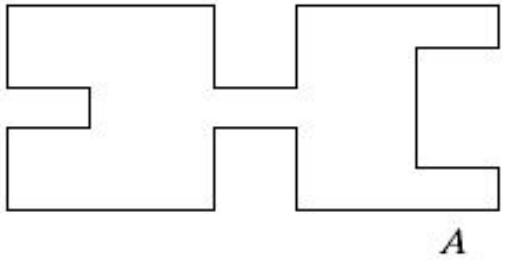
$$A \circ B = (A \ominus B) \oplus B \quad \Leftrightarrow \quad A \circ B = \cup \left\{ (B)_z \,\middle|\, (B)_z \subseteq A \right\}$$



Opening eliminates narrow and small details and corners.

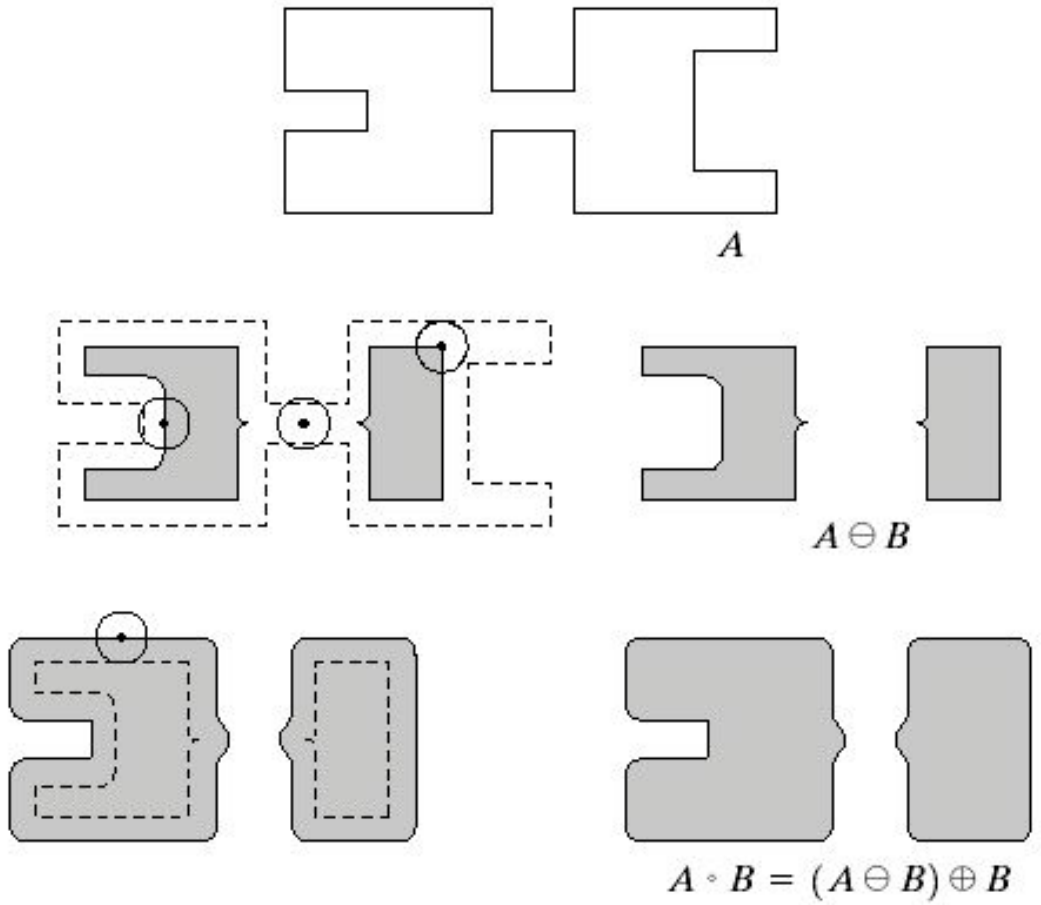
$$A \circ B = (A \ominus B) \oplus B$$

Example: Opening



$$A \circ B = (A \square B) \oplus B$$

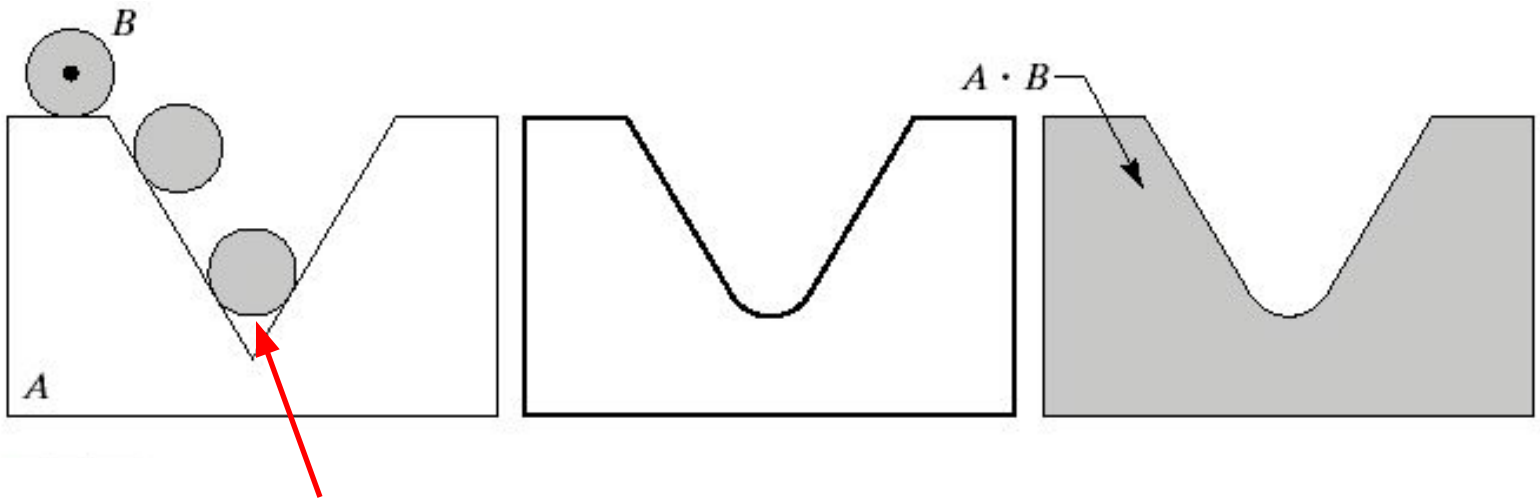
Example: Opening



Closing

$$A \bullet B = (A \oplus B) \ominus B$$

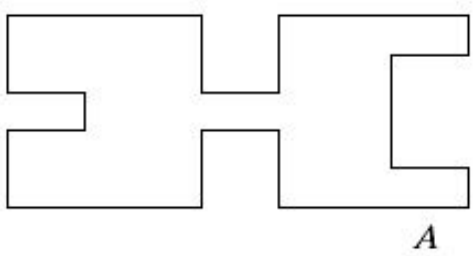
Closing $A \bullet B = (A \oplus B) \ominus B$



Closing fills narrow gaps and notches

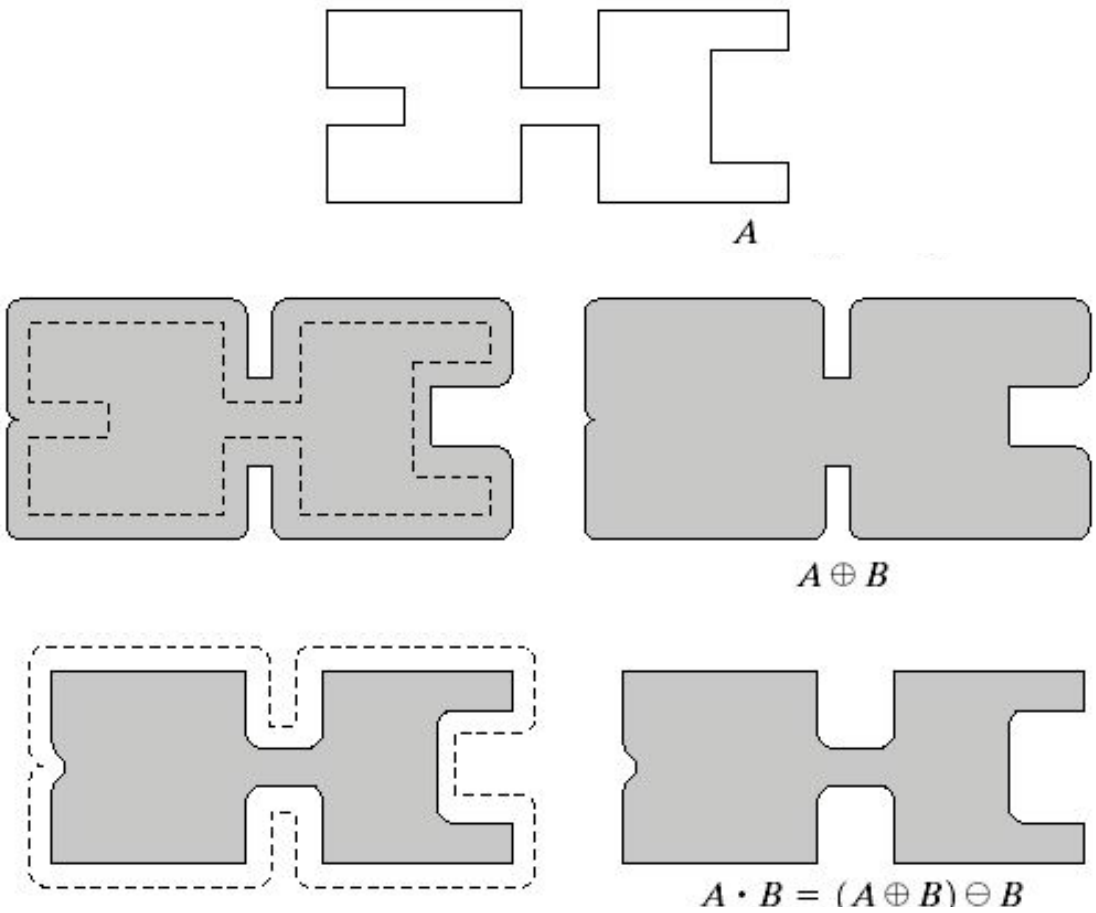
$$A \bullet B = (A \oplus B) \ominus B$$

Example: Closing



$$A \bullet B = (A \oplus B) \ominus B$$

Example: Closing



Opening vs. Closing

- Duality $(A \bullet B)^c = (A^c \circ B)$
- Properties of Opening
  1. $A \circ B \subseteq A$
  2. If $C \subset D$ then $C \circ B \subset D \circ B$
  3. $(A \circ B) \circ B = A \circ B$
- Properties of Closing
  1. $A \bullet B \supseteq A$
  2. If $C \subset D$ then $C \bullet B \subset D \bullet B$
  3. $(A \bullet B) \bullet B = A \bullet B$

Idempotent property: can't change any more

Example:
Application of Morphological Operations

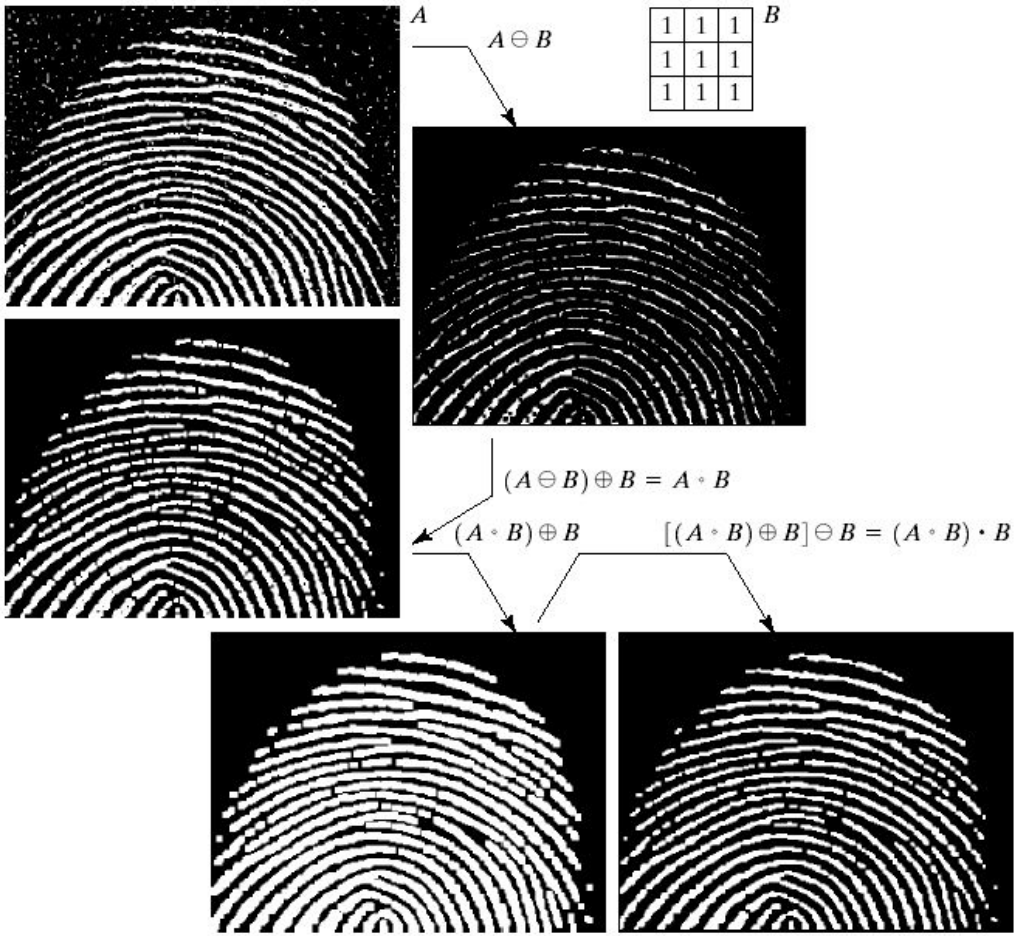
A

1	1	1
1	1	1
1	1	1

B

Finger print enhancement

Example:
Application of Morphological Operations



Finger print enhancement

Image Processing

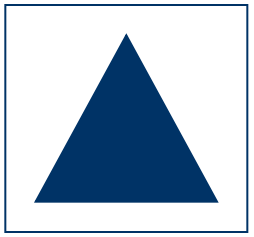
Some Applications of Image Morphology

Hit-or-Miss Transformation

$$A \circledast X = \left(A \boxminus X \right) \cap \left[A^c \boxminus (W - X) \right]$$

X

W



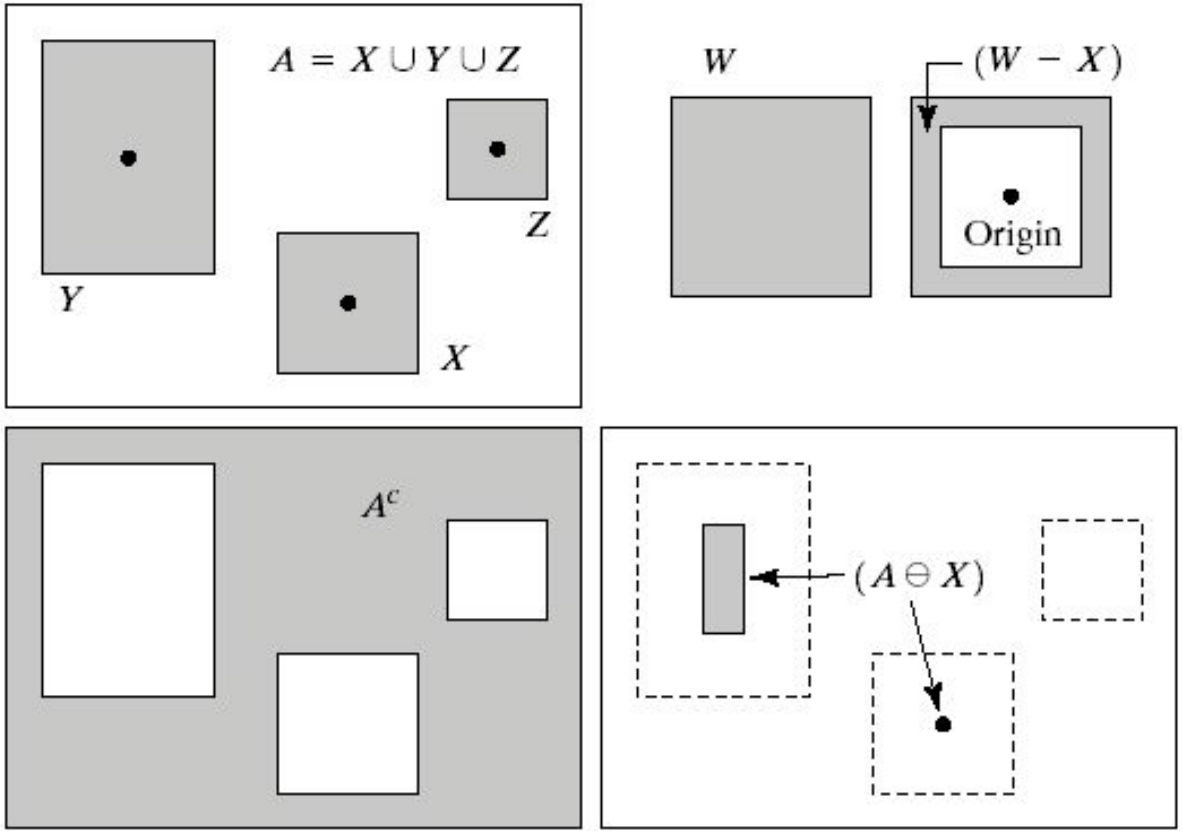
$W - X \approx X^c$

Hit-or-Miss Transformation

$$A \circledast X = \left(A \ominus X \right) \cap \left[A^c \ominus (W - X) \right]$$

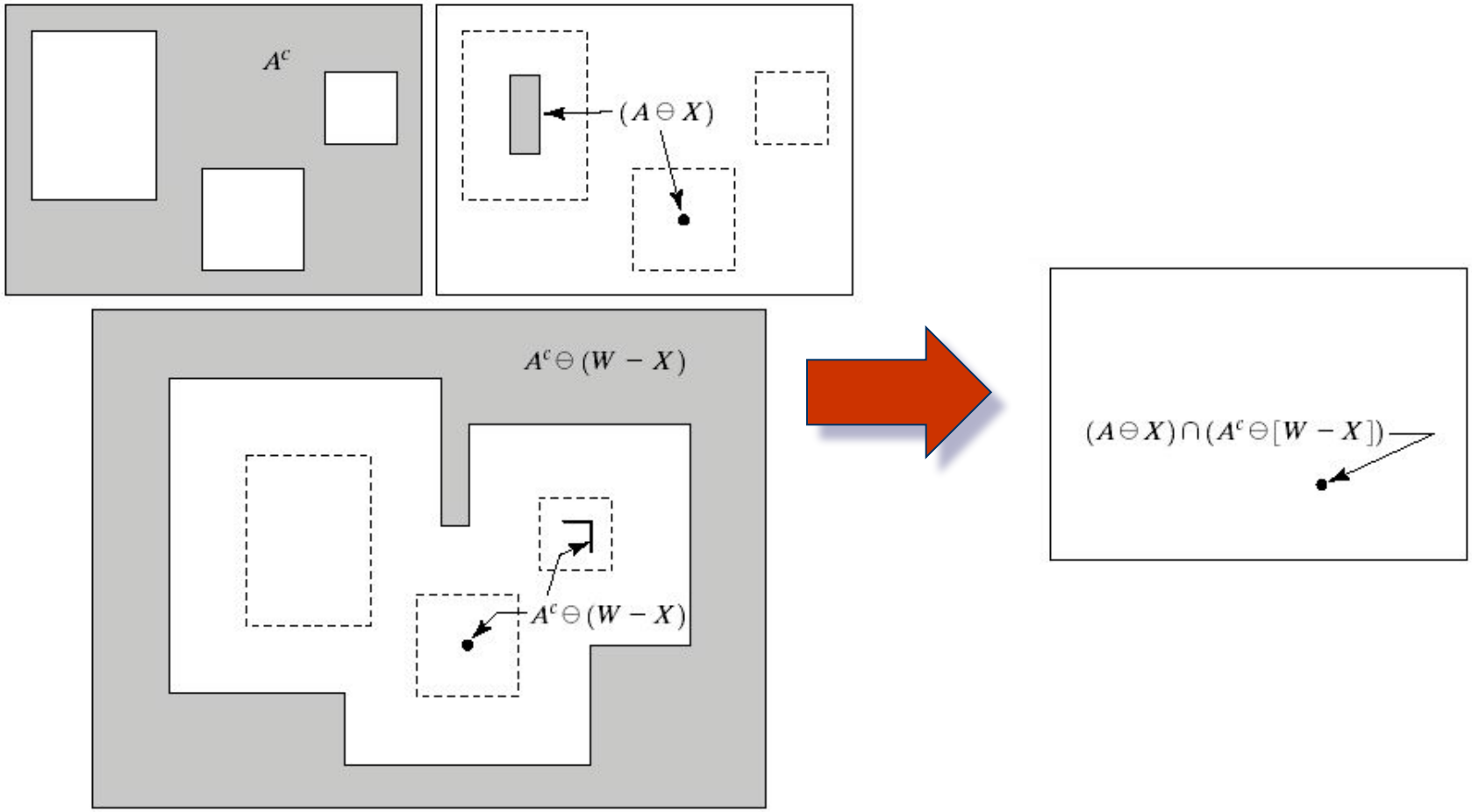
$$A \circledast X = \left(A \square X \right) \cap \left[A^c \square \left(W - X \right) \right]$$

Hit-or-Miss Transformation



$$A \circledast X = \left(A \square X\right) \cap \left[A^c \square (W - X)\right]$$

Hit-or-Miss Transformation

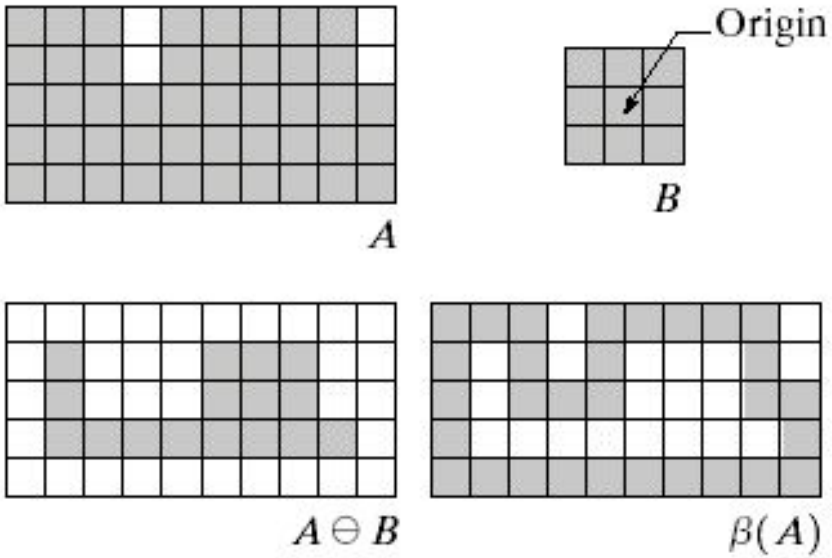


Boundary Extraction

$$\beta(A) = A - (A \ominus B)$$

$$\beta(A) = A - (A \ominus B)$$

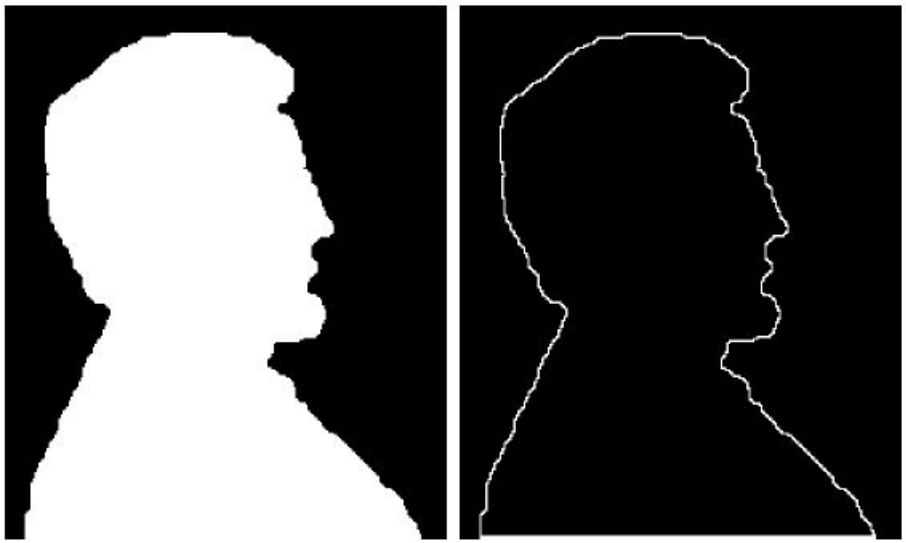
Boundary Extraction



$$\beta(A) = A - (A \ominus B)$$

Boundary Extraction

Original image



Boundary

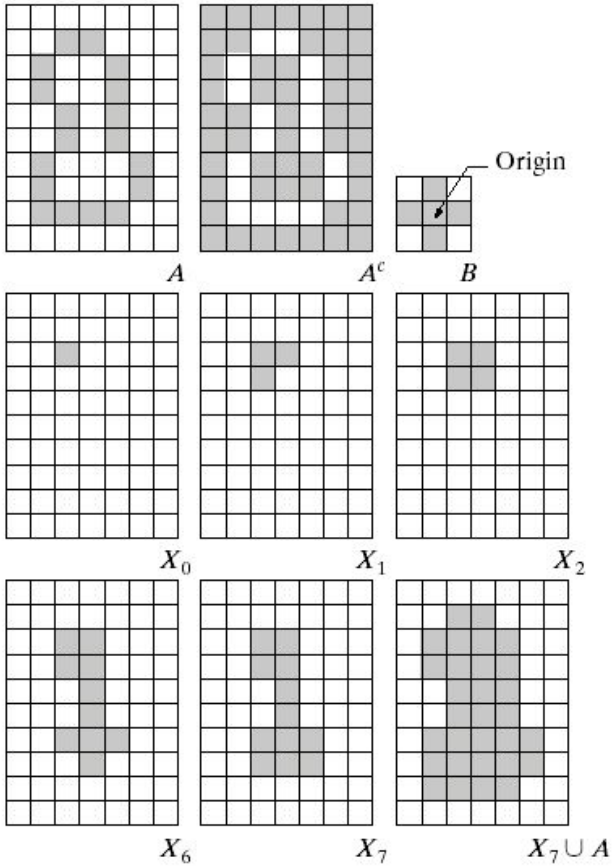
Region Filling

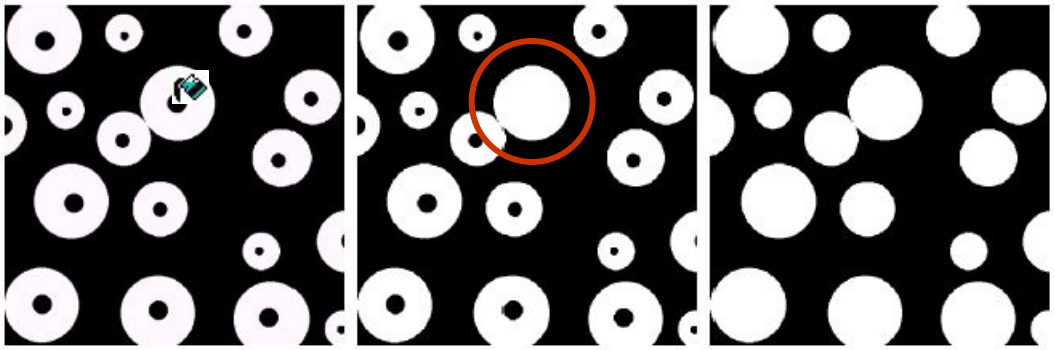
$$X_k = \left(X_{k-1} \oplus B\right) \cap A^c$$

$X_0 \in A$: seed pixel

$$X_k = \left(X_{k-1} \oplus B\right) \cap A^c$$

Region Filling





Original image

Results of region filling

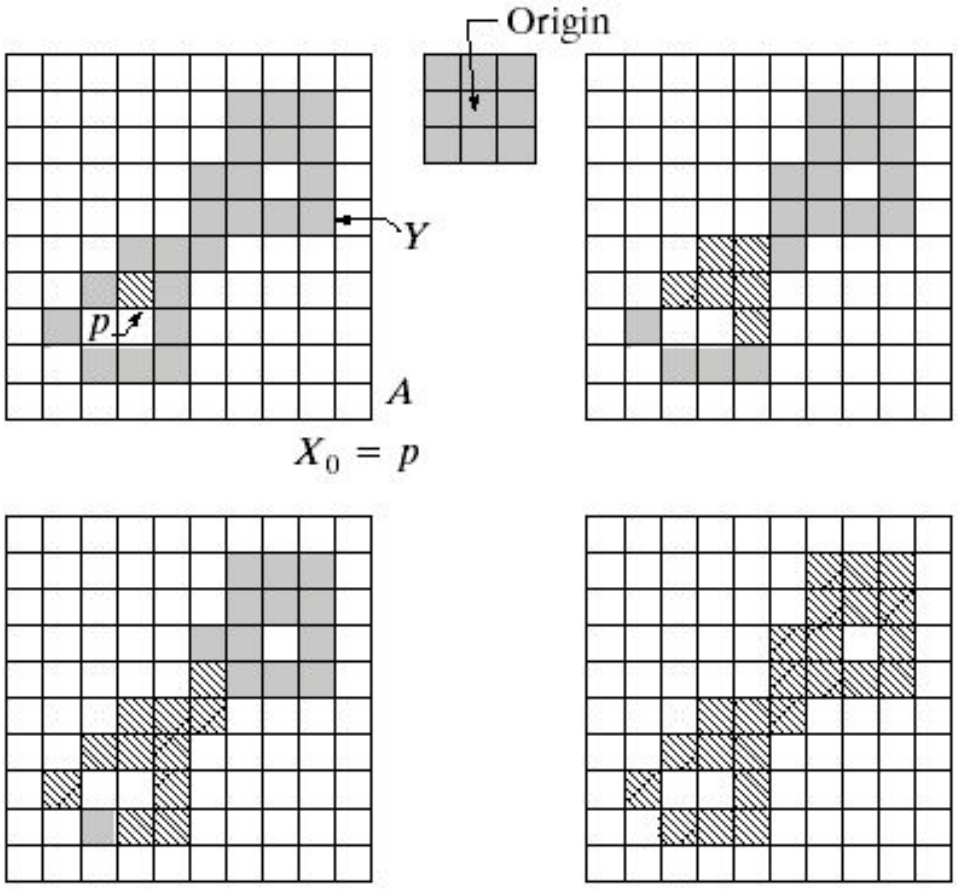
Extraction of Connected Components

$$X_k = \left(X_{k-1} \oplus B\right) \cap A$$

$X_0 \in A$: seed pixel

$$X_k = \left(X_{k-1} \oplus B \right) \cap A$$

Extraction of Connected Components



Morphological Operations for Gray Images

$$\text{erode}(x,y) = \min_{(x',y')\in \text{kernel}} \text{src}(x+x', y+y')$$

$$\text{dilate}(x,y) = \max_{(x',y')\in \text{kernel}} \text{src}(x+x', y+y')$$

Dilation and Erosion by OpenCV

```
void cvDilate(
  const CvArr* A,
  CvArr* C,
  IplConvKernel* B=0,
  int iterations=1
);
```

```
void cvErode(
  const CvArr* A,
  CvArr* C,
  IplConvKernel* B=0,
  int iterations=1
);
```

A
Source image.

C
Destination image.

B
Structuring element used for erosion. If it is NULL, a 3×3 rectangular structuring element is used.

iterations
Number of times erosion is applied.

Making Your Own Kernel

```
IplConvKernel* cvCreateStructuringElementEx(
  int nCols,
  int nRows,
  int anchorX,
  int anchorY,
  int shape,
  int* values
);
```

Shape value	Meaning
CV_SHAPE_RECT	The kernel is rectangular
CV_SHAPE_CROSS	The kernel is cross shaped
CV_SHAPE_ELLIPSE	The kernel is elliptical
CV_SHAPE_CUSTOM	The kernel is user-defined via values

More General Morphology

```
void cvMorphologyEx(
  const CvArr* A,
  CvArr* C,
  CvArr* temp,
  IplConvKernel* B,
  int operation,
  int iterations
);
```

A
Source image.

C
Destination image.

temp
Temporary image, required in some cases.

B
Structuring element.

operation
Type of morphological operation.

iterations
Number of times erosion is applied.

More General Morphology

```
void cvMorphologyEx(
  const CvArr* A,
  CvArr* C,
  CvArr* temp,
  IplConvKernel* B,
  int operation,
  int iterations
);
```

Value of operation	Morphological operator	Requires temp image?
CV_MOP_OPEN	Opening	No
CV_MOP_CLOSE	Closing	No
CV_MOP_GRADIENT	Morphological gradient	Always
CV_MOP_TOPHAT	Top Hat	For in-place only (src = dst)
CV_MOP_BLACKHAT	Black Hat	For in-place only (src = dst)

Destination image.

temp

Temporary image, required in some cases.

B

Structuring element.

operation

Type of morphological operation.

iterations

Number of times erosion is applied.

More General Morphology

Value of operation	Morphological operator	Requires temp image?
CV_MOP_OPEN	Opening	No
CV_MOP_CLOSE	Closing	No
CV_MOP_GRADIENT	Morphological gradient	Always
CV_MOP_TOPHAT	Top Hat	For in-place only (src = dst)
CV_MOP_BLACKHAT	Black Hat	For in-place only (src = dst)

```
CV_MOP_OPEN:
    C = open(A,B) = dilate(erode(A,B),B)
CV_MOP_CLOSE:
    C = close(A,B) = erode(dilate(A,B),B)
CV_MOP_GRADIENT:
    C = morph_grad(A,B) = dilate(A,B) - erode(A,B)
CV_MOP_TOPHAT:
    C = tophat(A,B) = A - erode(A,B)
CV_MOP_BLACKHAT:
    C = blackhat(A,B) = dilate(A,B) - A
```

Exercise

Download
Test Program

Image Processing

Flood Fill

Flood Fill

Flood Fill

- Used to mark or isolate portions of an image for further processing or analysis
- Used to derive, from an input image, masks that can be used for subsequent routines to speed or restrict processing to only those pixels indicated by the mask.
- Some routines support mask

Flood Fill

```
void cvFloodFill(
  IplImage* img,
  CvPoint seedPoint,
  CvScalar newVal,
  CvScalar loDiff =
cvScalarAll(0),
  CvScalar upDiff =
cvScalarAll(0),
  CvConnectedComp* comp = NULL,
  int flags = 4,
  CvArr* mask = NULL
);
```

Flood Fill

Download
Test Program

Image Processing

Resize

Resize

```
void cvResize(
  const CvArr* src,
  CvArr* dst,
  int interpolation=CV_INTER_LINEAR
);
```

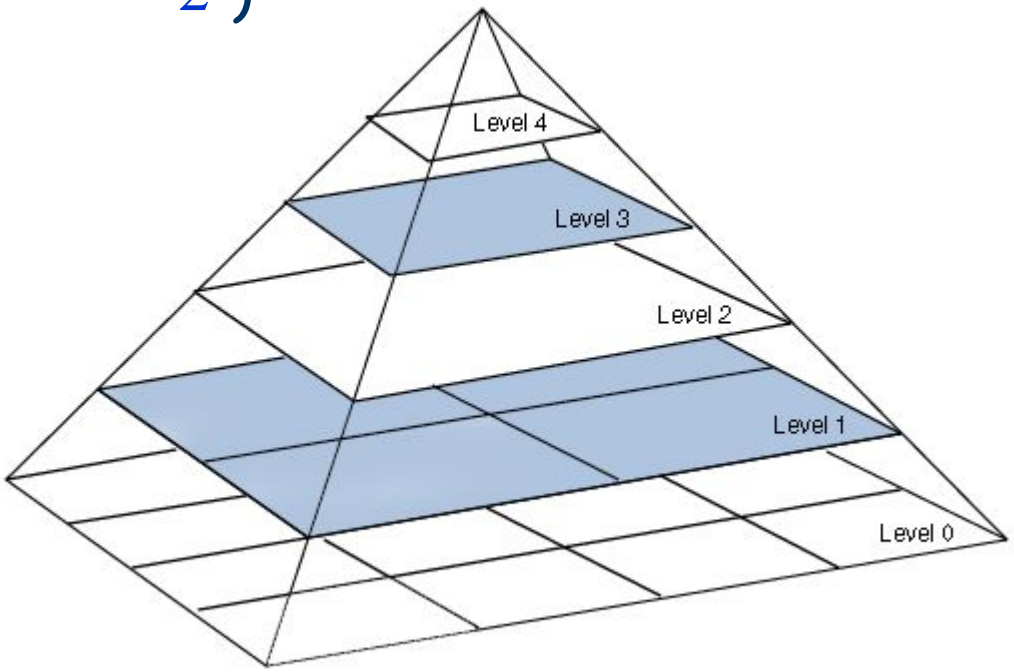
Interpolation	Meaning
CV_INTER_NN	Nearest neighbor
CV_INTER_LINEAR	Bilinear
CV_INTER_AREA	Pixel area re-sampling
CV_INTER_CUBIC	Bicubic interpolation

Image Processing

Image Pyramids

Image Pyramids

Represent $N \times N$ image as a "pyramid" of 1×1, 2×2, 4×4, $2^k \times 2^k$ images (assuming $N = 2^k$)



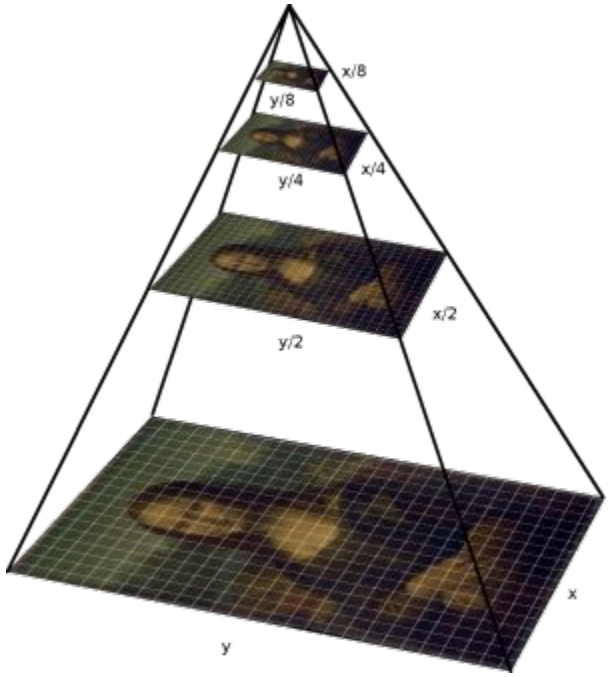
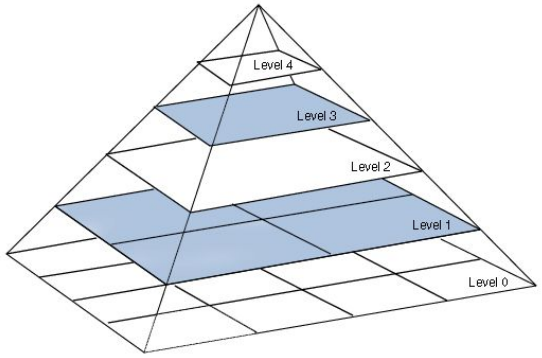
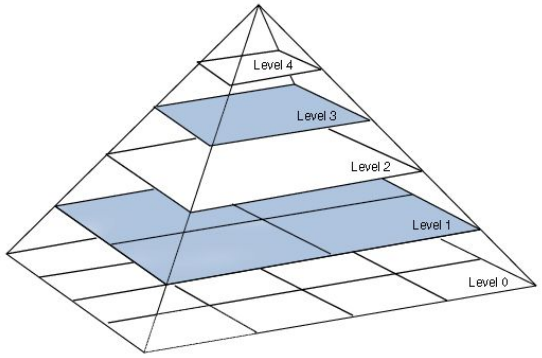


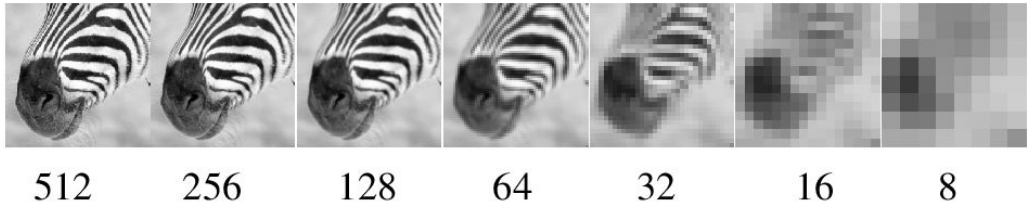
Image Pyramids



- Represent the image using a collection of images
  - Called a pyramid because the resolution of the images decreases
- Most common pyramid: Gaussian pyramid
  - At each level, generate the next level by blurring the image with a Gaussian, then downsampling
- For reconstruction: Laplacian pyramid needed

Gaussian Pyramid

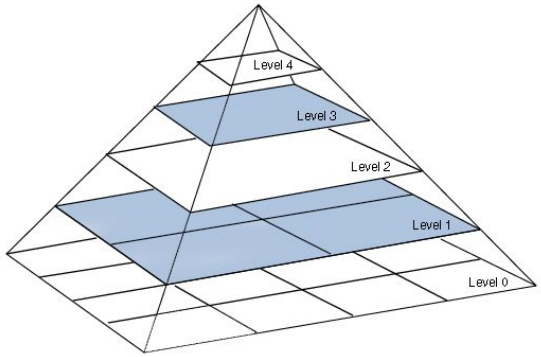


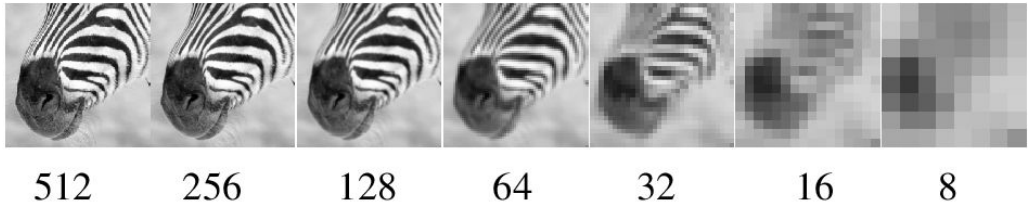


512 256 128 64 32 16 8

At each level, generate the next level by blurring the image with a Gaussian, then downsampling

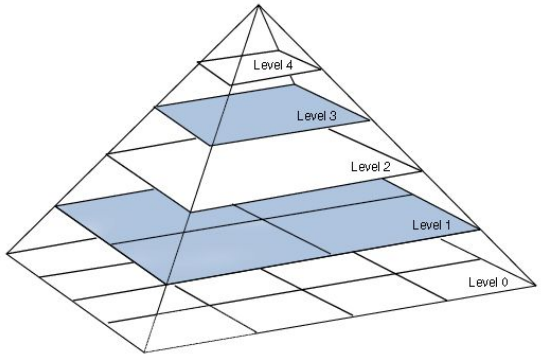
Problems on Gaussian Pyramid





- It is redundant
- Each level contains all of the low-frequencies that are available at the lower levels.

Laplacian Pyramid



Original Image

Reduce

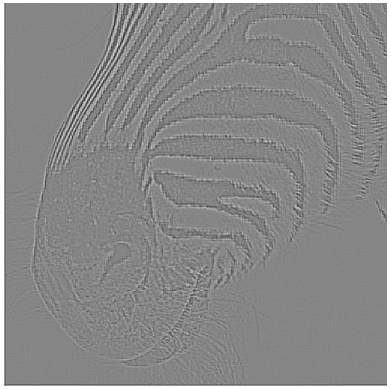
Enlarg

Original Image

\-

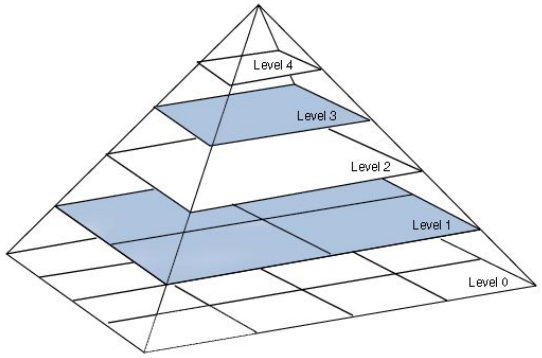
Shrunk and Enlarged Version

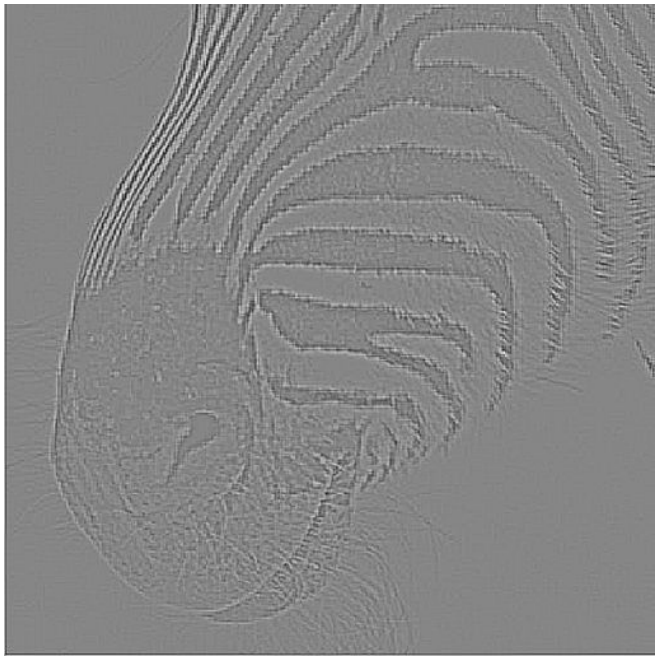
=



Detail

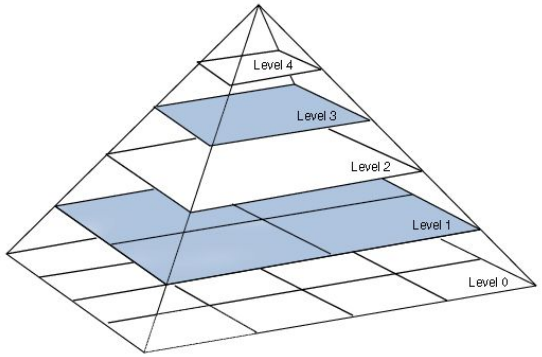
The Detail

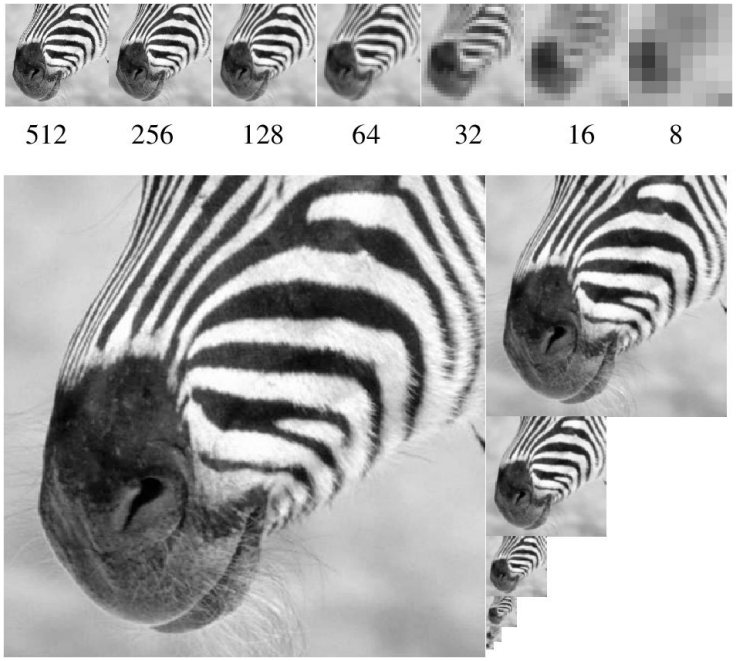




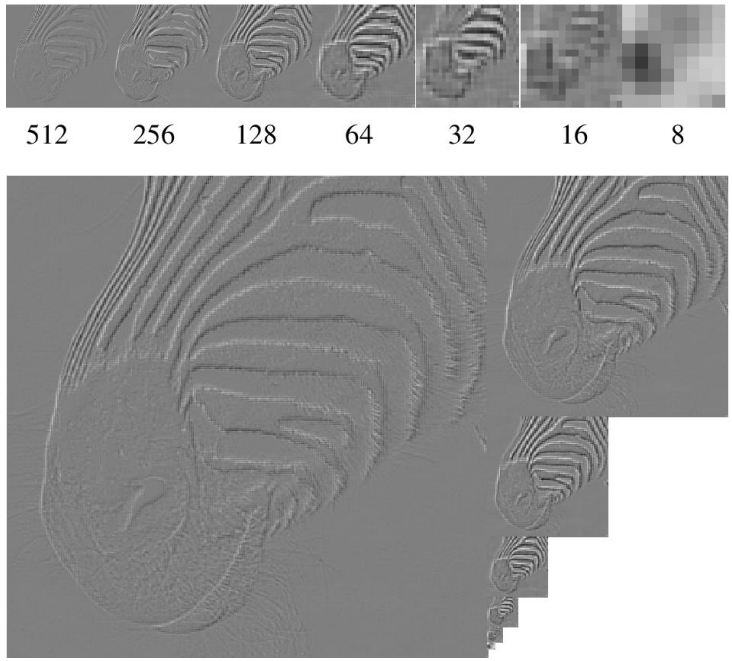
The image represents everything in the high-resolution image that cannot be represented in a low-resolution image

Gaussian Pyramid & Lapacian Pyramid



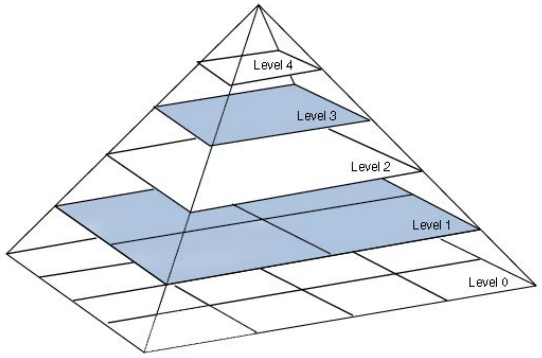


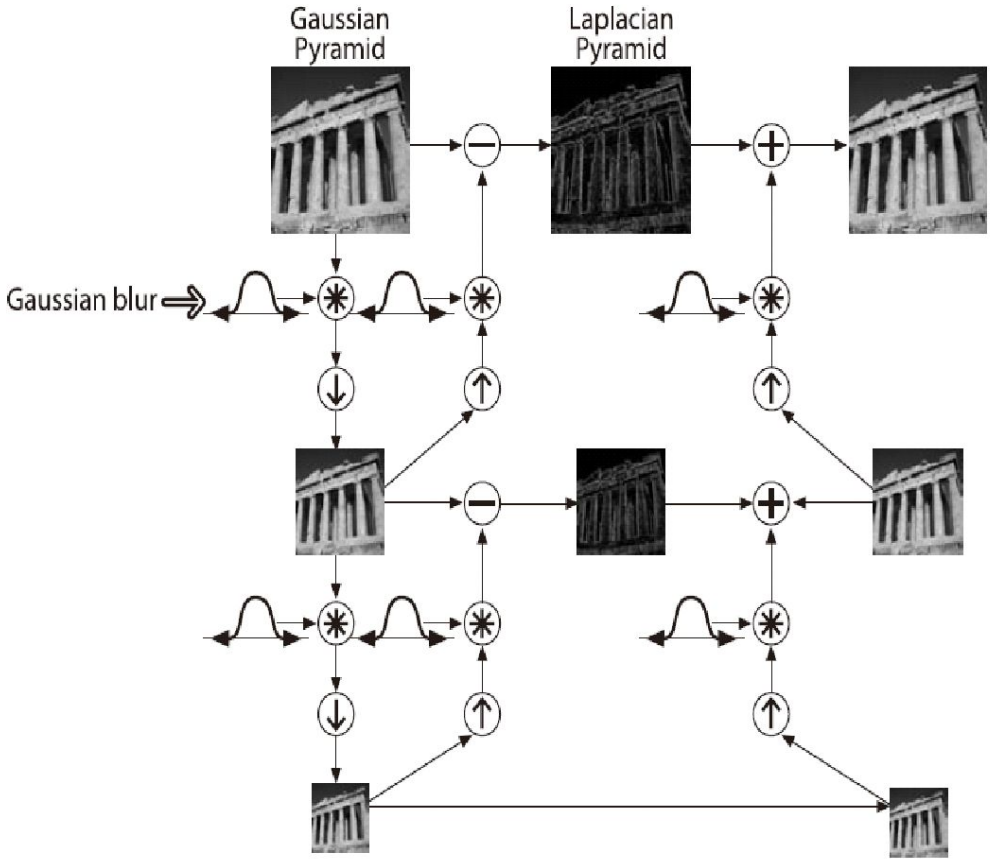
Gaussian Pyramid



Laplacian Pyramid

Image Compression





PyrDown

```
void cvPyrDown(
   const CvArr* src,
   CvArr* dst,
   int filter=CV_GAUSSIAN_5x5
);
```

The function performs the downsampling step of the Gaussian pyramid decomposition. First it convolves the source image with the specified filter and then downsamples the image by rejecting even rows and columns.

PyrUp

```
void cvPyrUp(
   const CvArr* src,
   CvArr* dst,
   int filter=CV_GAUSSIAN_5x5
);
```

The function cvPyrUp performs up-sampling step of Gaussian pyramid decomposition. First it upsamples the source image by injecting even zero rows and columns and then convolves result with the specified filter multiplied by 4 for interpolation. So the destination image is four times larger than the source image.

Image Segmentation

Image Segmentation

Pyramid Image Segmentation

```
void cvPyrSegmentation(
  IplImage* src,
  IplImage* dst,
  CvMemStorage* storage,
  CvSeq** comp,
  int level,
  double threshold1,
  double threshold2
)
```

Pyramid Image Segmentation

Download
Test Program

Image Processing

Threshold

Threshold

```
double cvThreshold(
  CvArr* src,
  CvArr* dst,
  double threshold,
  double max_value,
  int threshold_type
);
```

Threshold type	Operation
CV_THRESH_BINARY	$dst_i = (src_i > T) \,?\, M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T) \,?\, 0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T) \,?\, M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T) \,?\, 0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T) \,?\, src_i : 0$

Threshold

```
double cvThreshold(
  CvArr* src,
  CvArr* dst,
  double threshold,
  double max_value,
  int threshold_type
);
```

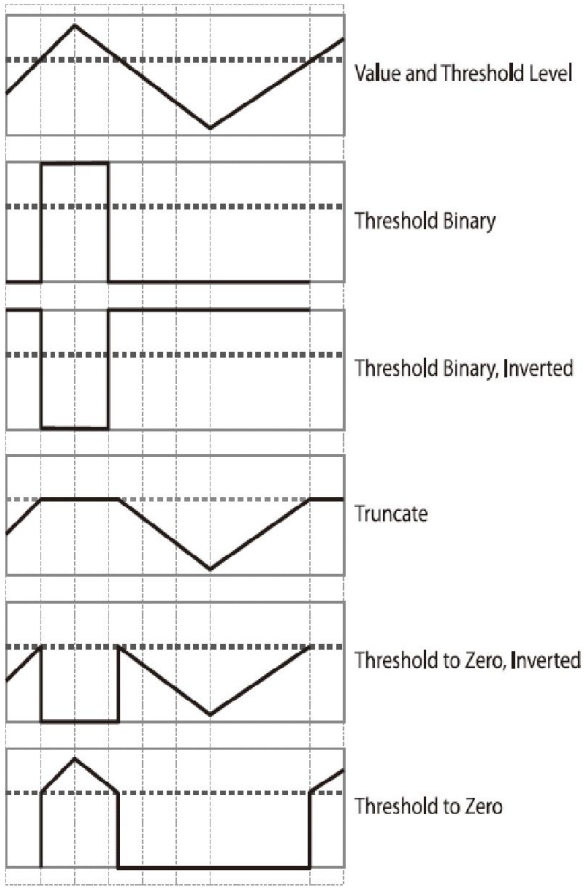
Threshold type	Operation
CV_THRESH_BINARY	$dst_i = (src_i > T) \,?\, M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T) \,?\, 0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T) \,?\, M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T) \,?\, 0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T) \,?\, src_i : 0$

CV_THRESH_OTSU

Threshold

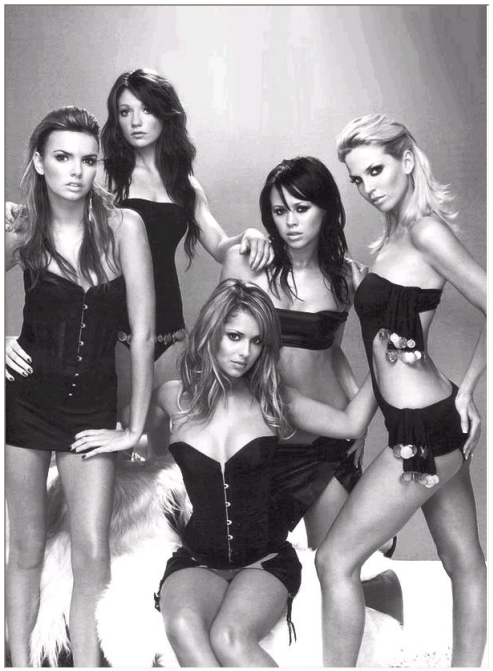
Threshold type	Operation
CV_THRESH_BINARY	$dst_i = (src_i > T) \; ? \; M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T) \; ? \; 0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T) \; ? \; M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T) \; ? \; 0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T) \; ? \; src_i : 0$

```
double cvThreshold(
  CvArr* src,
  CvArr* dst,
  double threshold,
  double max_value,
  int threshold_type
);
```



Threshold

Threshold type	Operation
CV_THRESH_BINARY	$dst_i = (src_i > T) \,?\, M : 0$
CV_THRESH_BINARY_INV	$dst_i = (src_i > T) \,?\, 0 : M$
CV_THRESH_TRUNC	$dst_i = (src_i > T) \,?\, M : src_i$
CV_THRESH_TOZERO_INV	$dst_i = (src_i > T) \,?\, 0 : src_i$
CV_THRESH_TOZERO	$dst_i = (src_i > T) \,?\, src_i : 0$



CV_THRESHOLD_BINARY

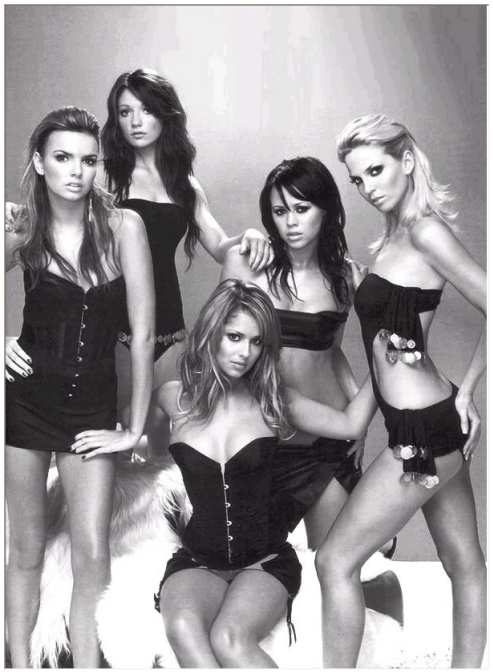
threshold=100

CV_THRESHOLD_BINARY | CV_THRESHOLD_OTSU

Adaptive Threshold

```
void cvAdaptiveThreshold(
  CvArr* src,
  CvArr* dst,
  double max_val,
  int adaptive_method =
CV_ADAPTIVE_THRESH_MEAN_C
  int threshold_type = CV_THRESH_BINARY,
  int block_size = 3,
  double param1 = 5
);
```

Adaptive Threshold



CV_ADAPTIVE_THRESH_MEAN_C

CV_THRESHOLD_BINARY

threshold=5

Threshold & Adaptive Threshold

Download
Test Program