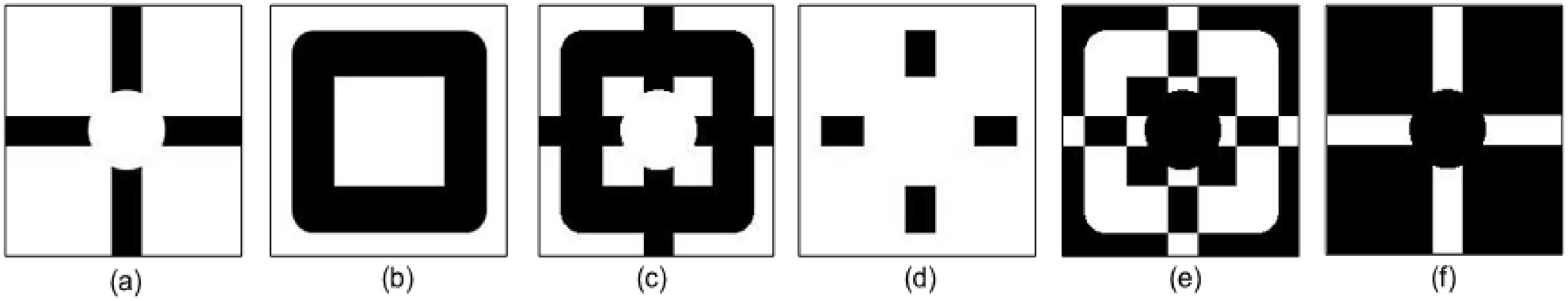
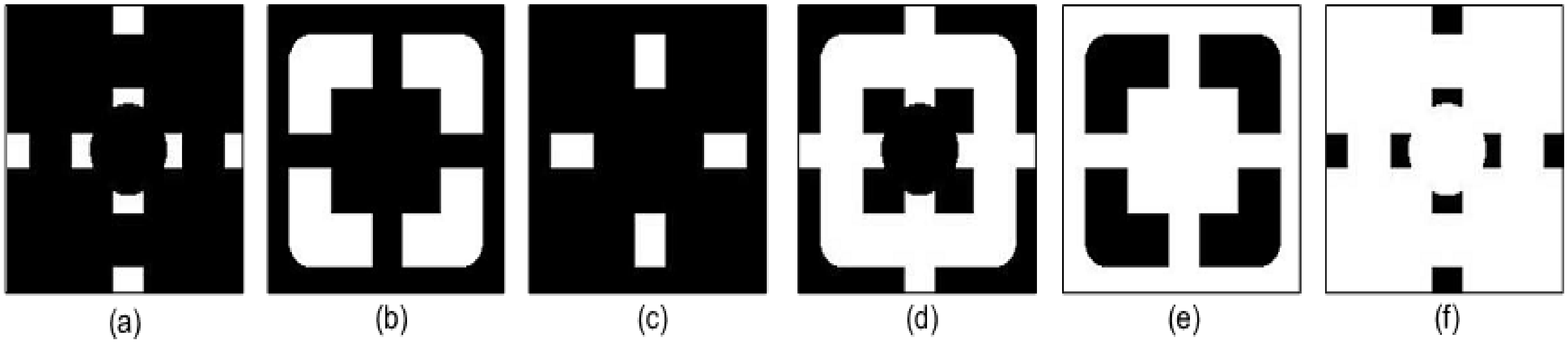
MORPHOLOGICAL IMAGE PROCESSING
XỬ LÝ HÌNH THÁI ẢNH



Basic Boolean logical operators. (a) Binary image A; (b) Binary image B; (c) A AND B; (d) A OR B; (e) A XOR B; (f) NOT A.



Combined Boolean logical operators. (a) (NOT A) AND B; (b) A AND (NOT B); (c) (NOT A) AND (NOT B); (d) NOT (A AND B); (e) (NOT A) OR B; (f) A OR (NOT B).

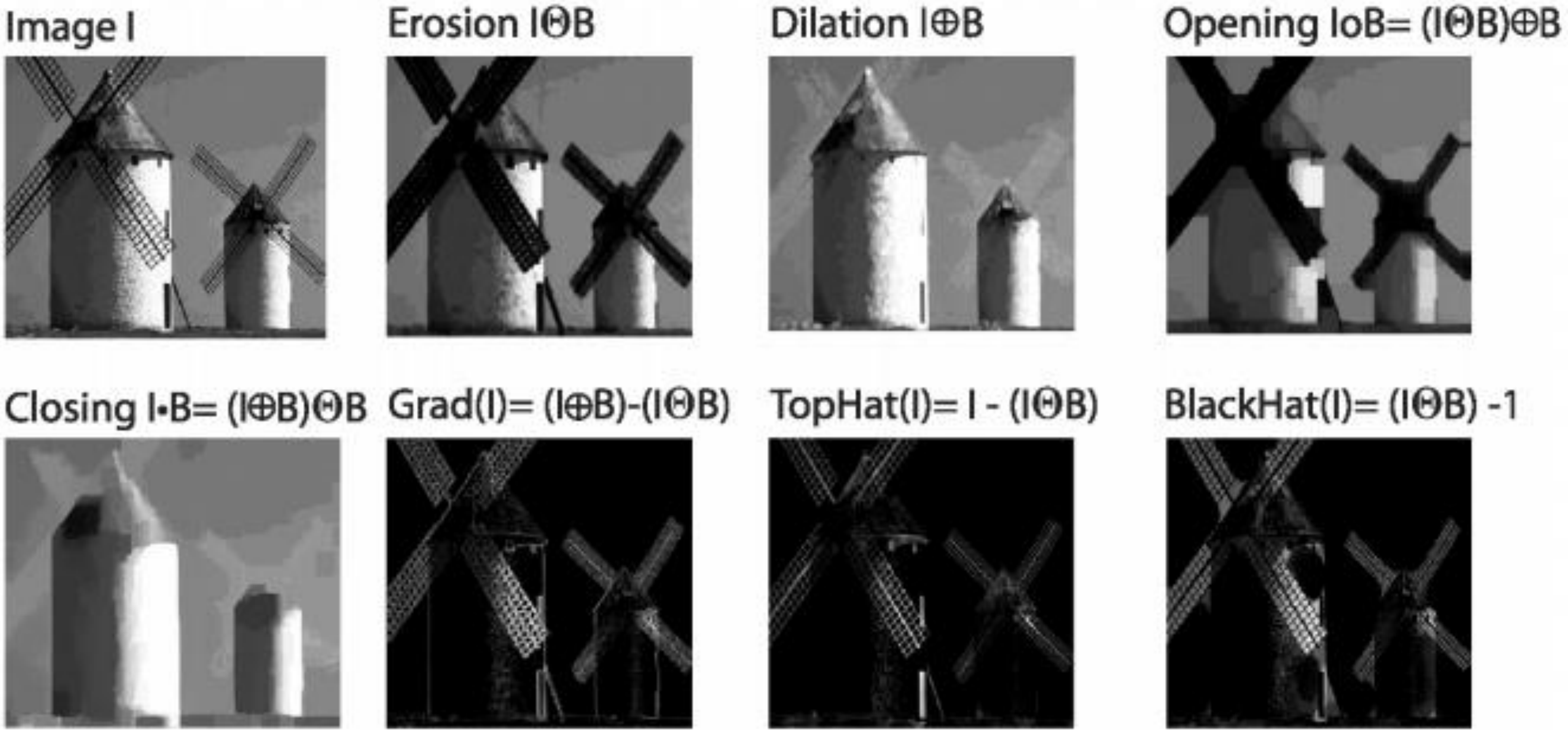
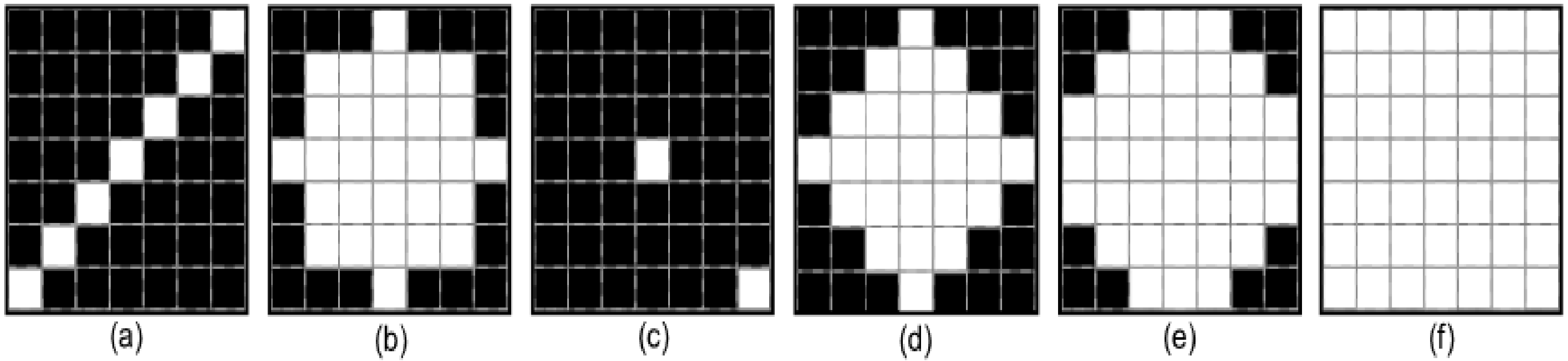


Image I
Erosion IΘB
Dilation I⊕B
Opening IoB= (IΘB)⊕B
Closing I•B= (I⊕B)ΘB
Grad(I)= (I⊕B)-(IΘB)
TopHat(I)= I - (IΘB)
BlackHat(I)= (IΘB) -1

CÁC PHẦN TỬ CẤU TRÚC KERNEL

- Có dạng đoạn thẳng, đĩa, kim cương, hình vuông, cặp điểm, là ma trận các giá trị 0,1
- Có điểm gốc hay còn gọi là điểm neo (anchor point), thường là tâm kernel hay có thể chọn tùy ý bên trong kernel



Some typical structure elements. (a) A line segment SE with the length 7 and the angle 45°; (b) A disk SE with the radius 3; (c) A pair of points SE containing two points with the offset 3; (d) A diamond-shaped SE; (e) A octagonal SE; (f) A 7×7 square S.

HÀM MORPHO MATLAB

`bwhitmiss`	Binary hit-miss operation
`bwmorph`	Morphological operations on binary images
`bwmorph3`	Morphological operations on binary volume
`bwskel`	Reduce all objects to lines in 2-D binary image or 3-D binary volume
`bwulterode`	Ultimate erosion
`bwareaopen`	Remove small objects from binary image

`imbothat`	Bottom-hat filtering
`imclearborder`	Suppress light structures connected to image border
`imclose`	Morphologically close image
`imdilate`	Dilate image
`imerode`	Erode image
`imextendedmax`	Extended-maxima transform
`imextendedmin`	Extended-minima transform

HÀM MORPHO MATLAB

imhmax	H-maxima transform
imhmin	H-minima transform
imimposemin	Impose minima
imopen	Morphologically open image
imreconstruct	Morphological reconstruction
imregionalmax	Regional maxima
imregionalmin	Regional minima
imtophat	Top-hat filtering
watershed	Watershed transform
conndef	Create connectivity array
iptcheckconn	Check validity of connectivity argument

applylut	Neighborhood operations on binary images using lookup tables
bwlookup	Nonlinear filtering using lookup tables
makelut	Create lookup table for use with bwlookup

- Hàm strel tạo SE
- Shape: hình dạng, ‘diamond’, ‘disk’,…
- Parameters: thông số kích thước
- Ví dụ:

```
se = strel(shape, parameters)
```

```
se1 = strel('square',11)      % 11-by-11 square
se2 = strel('line',10,45)     % line, length 10, angle 45 degrees
se3 = strel('disk',15)        % disk, radius 15
se4 = strel('ball',15,5)      % ball, radius 15, height 5
```

- SE phức tạp thường được phân tích thành các SE đơn giản để tăng tốc xử lý

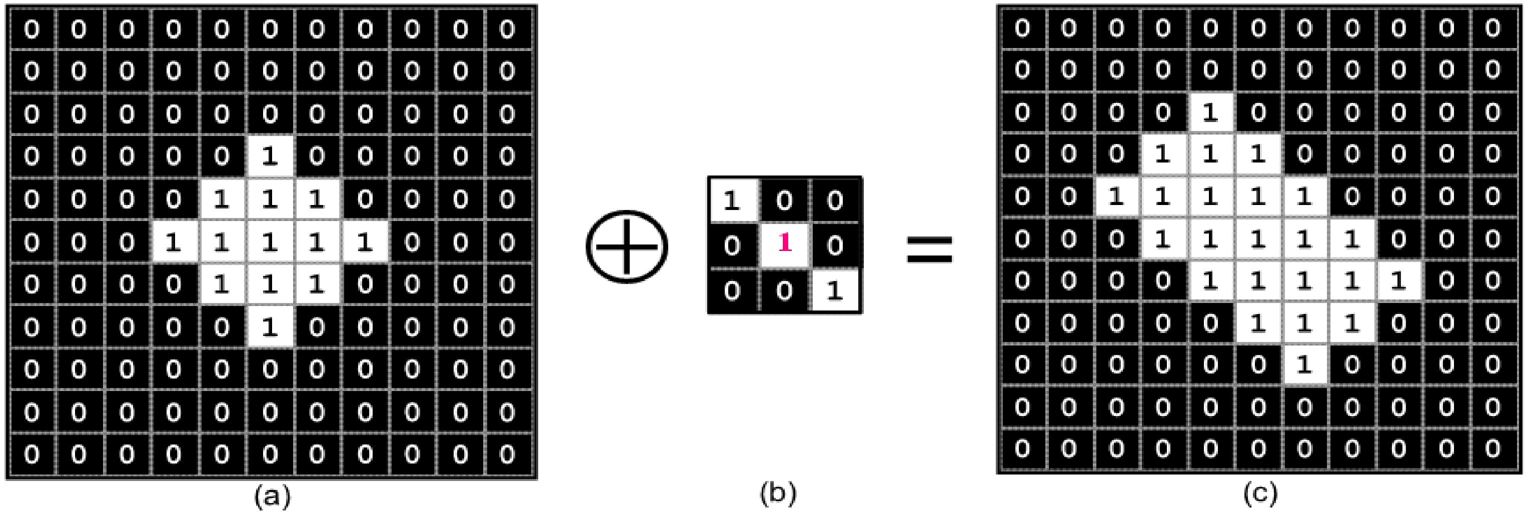
$$\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & \boxed{1} & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix} \qquad \begin{bmatrix} 1 & 1 & \boxed{1} & 1 & 1 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ \boxed{1} \\ 1 \\ 1 \end{bmatrix}$$

GIÃN NỞ DILATION

- Làm phần ảnh trắng to ra hay dầy hơn
- Cho ảnh nhị phân A và SE B, giãn A bởi B biểu diễn bởi biểu thức

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

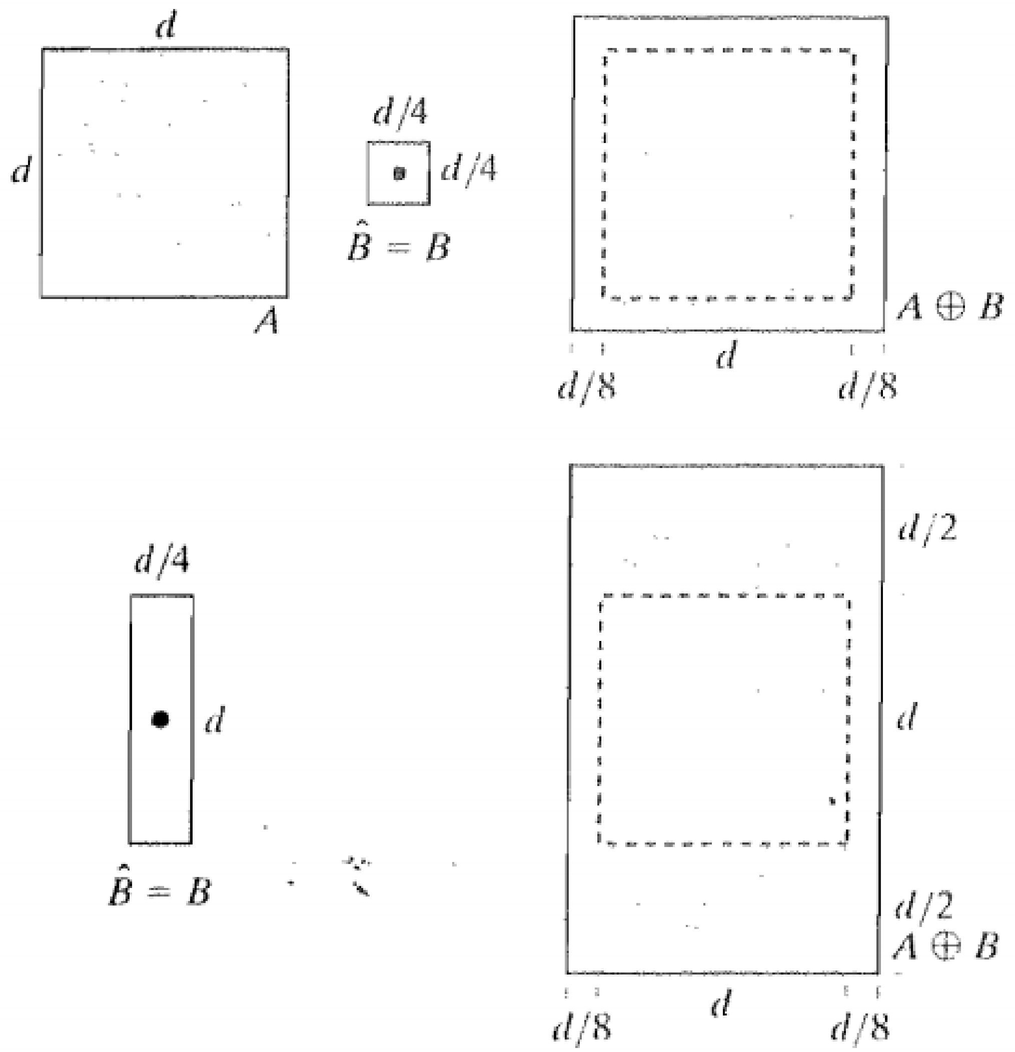
- Cho điểm neo SE di chuyển bên trong ảnh, pixel dưới điểm neo có giá trị là cực đại của các giá trị pixel dưới các phần tử 1 của SE

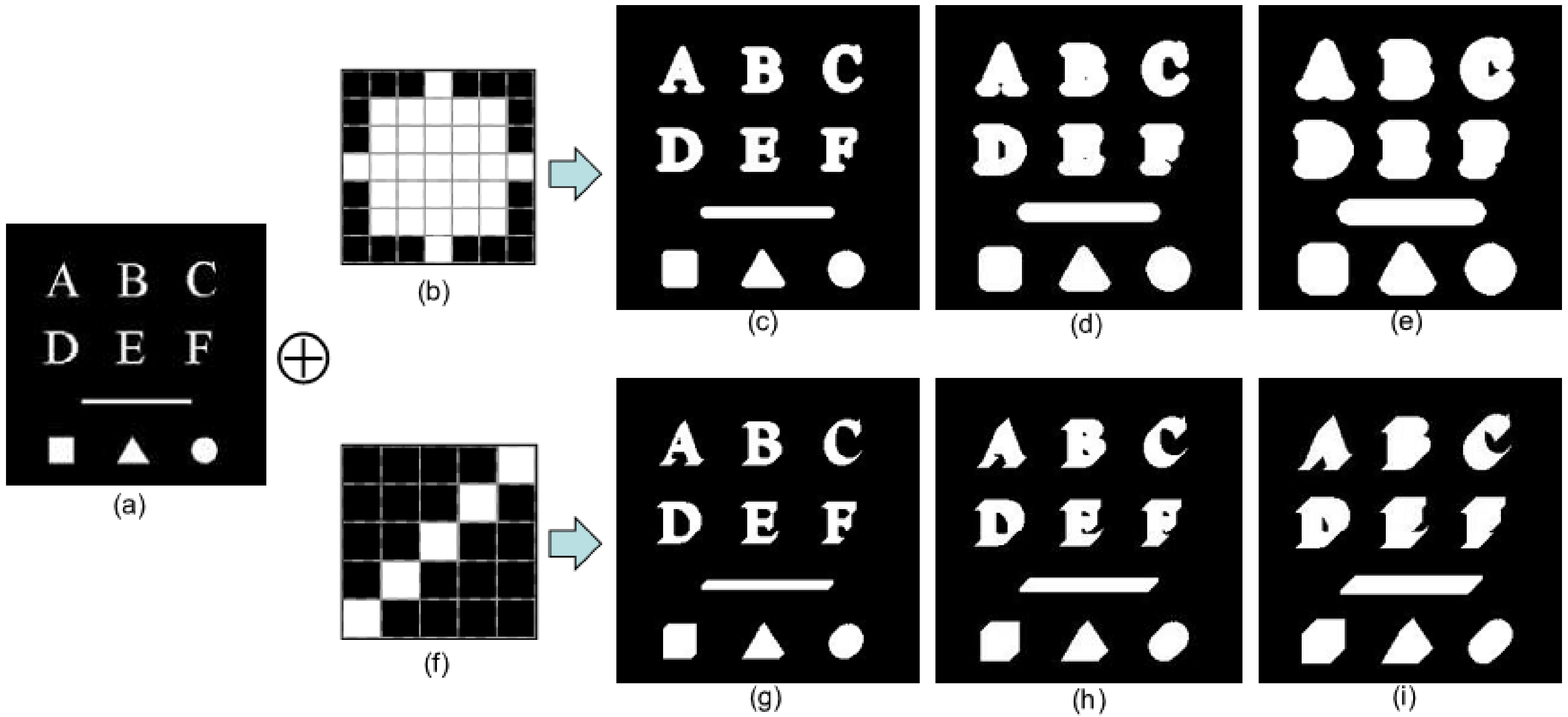


$$\begin{bmatrix}0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&0&0&0&0&0\\0&0&0&0&1&1&1&0&0&0&0\\0&0&0&1&1&1&1&1&0&0&0\\0&0&0&0&1&1&1&0&0&0&0\\0&0&0&0&0&1&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\end{bmatrix} \oplus \begin{bmatrix}1&0&0\\0&\color{red}{1}&0\\0&0&1\end{bmatrix} = \begin{bmatrix}0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&1&0&0&0&0&0&0\\0&0&0&1&1&1&0&0&0&0&0\\0&0&1&1&1&1&1&0&0&0&0\\0&0&0&1&1&1&1&1&0&0&0\\0&0&0&0&1&1&1&1&1&0&0\\0&0&0&0&0&1&1&1&0&0&0\\0&0&0&0&0&0&1&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0&0\end{bmatrix}$$

Illustration of morphological dilation. (a) Original binary image with a diamond object; (b) Structure element with three pixels arranged in a diagonal line at angle of 135°, the origin of the structure element is clearly identified by a red 1; (c) Dilated image, 1 at each location of the origin such that the structure element overlaps at least one 1-valued pixel in the input image (a).

GIÃN NỞ DILATION

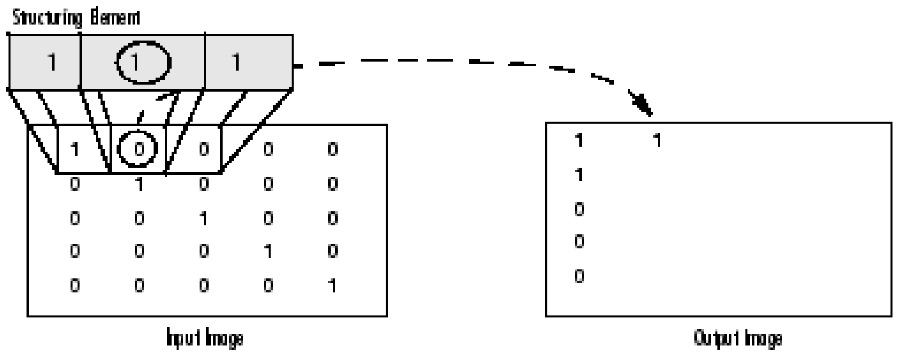




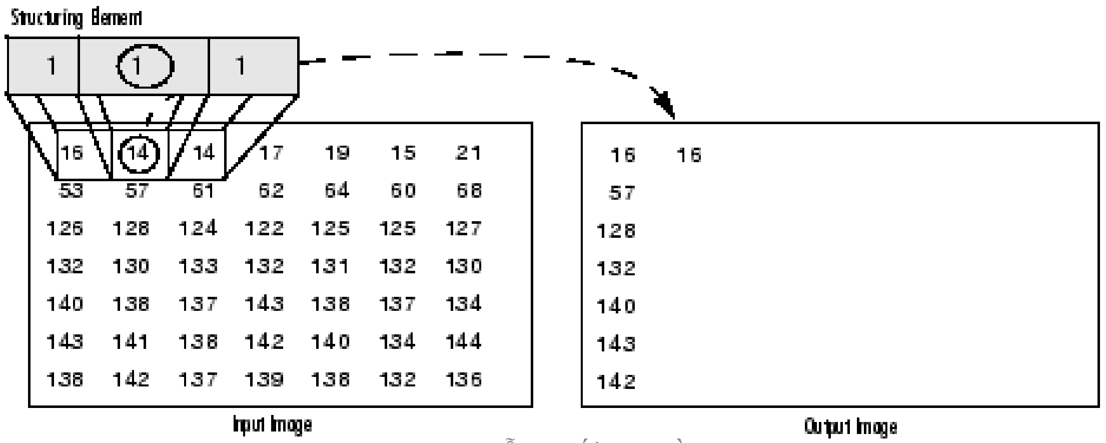
Example of morphological dilation. (a) A binary input image; (b) A disk structure element; (c) Dilated image of (a) by SE (b); (d) After twice dilation by SE (b); (e) After three times dilation by SE (b); (f) A line structure element; (g) Dilated image of (a) by SE (f); (h) After twice dilation by SE (f); (i) After three times dilation by SE (f).

- Với ảnh xám hay nhị phân ta cho SE di chuyển trên ảnh A, cường độ sáng của điểm ảnh tại gốc SE là giá trị cực đại các pixel bao bởi các phần tử 1 của SE. Với ảnh màu ta xét từng mặt phẳng màu
- Dilation mở rộng vùng sáng, lấp các lỗ tối nhỏ

Morphological Dilation of a Binary Image



Morphological Dilation of a Grayscale Image

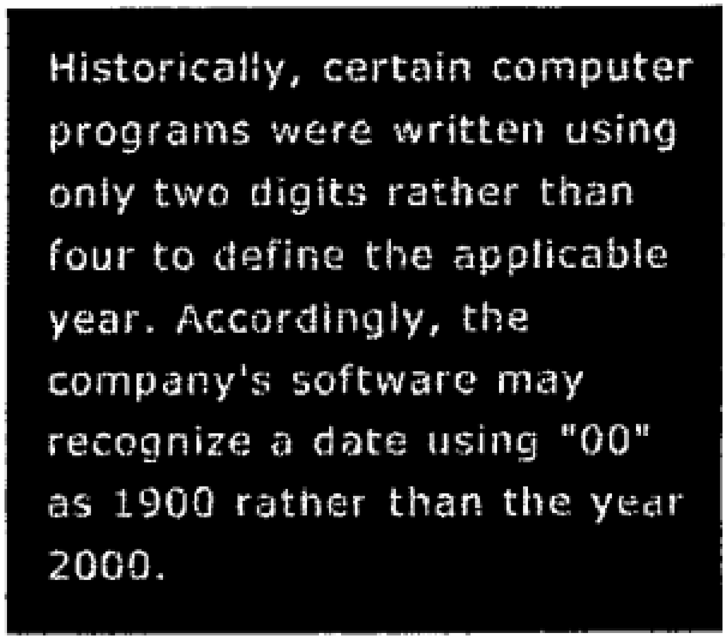


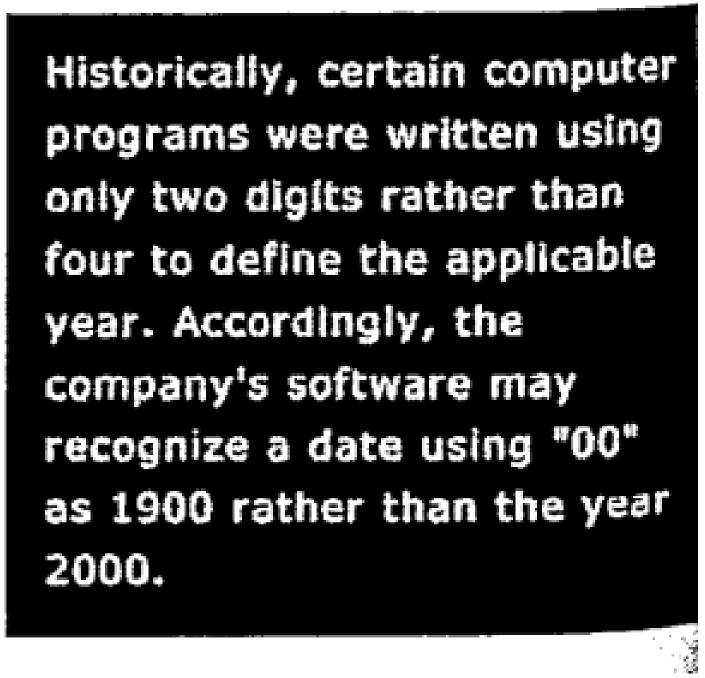
Matlab: hàm A2=imdilate(A,B)

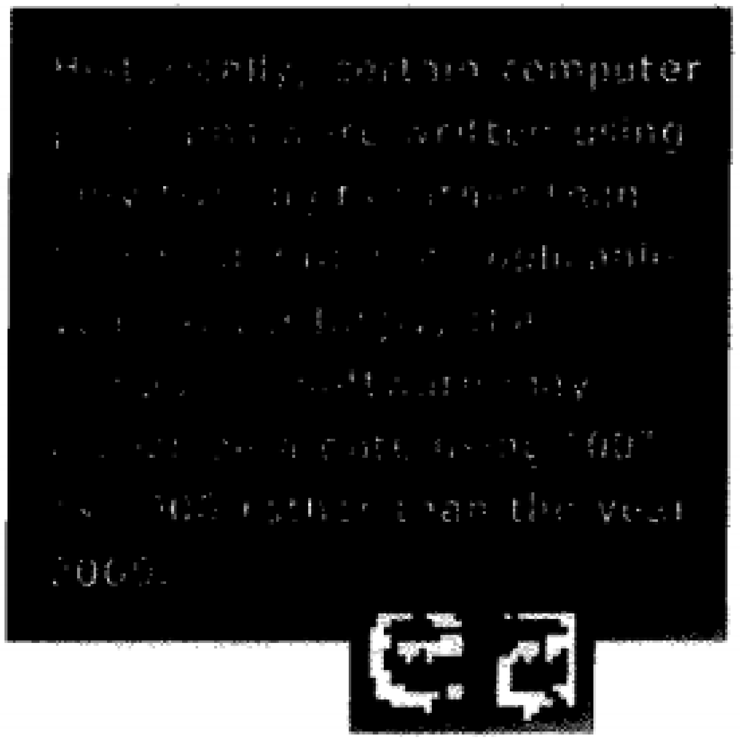
A là ảnh xám hoặc nhị phân, B là SE biểu thị dạng ma trận, điểm neo có tọa độ floor((size(B)+1)/2).

```
A = imread('broken_text.tif');
B = [0 1 0; 1 1 1; 0 1 0];
A2 = imdilate(A, B);
imshow(A2)
```

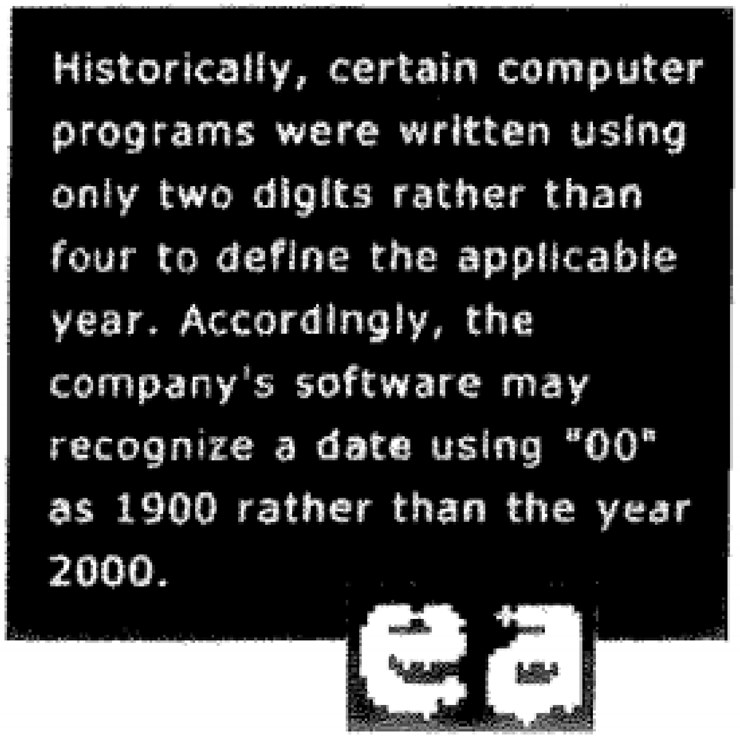
$$\begin{matrix} 0 & 1 & 0 \\ 1 & \boxed{1} & 1 \\ 0 & 1 & 0 \end{matrix}$$


Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.


Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



Historically, certain computer
programs were written using
[illegible]
[illegible] applicable
[illegible]
[illegible] software may
[illegible] a date using "00"
as 1900 rather than the year
2000.

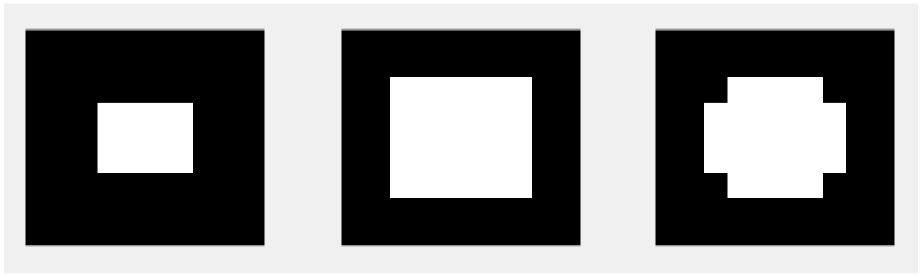


Historically, certain computer
programs were written using
only two digits rather than
four to define the applicable
year. Accordingly, the
company's software may
recognize a date using "00"
as 1900 rather than the year
2000.

0	1	0
1	1	1
0	1	0

MATLAB imdilate

```
BW = zeros(9,10);
BW(4:6,4:7) = 1;
SE = strel('square',3);
BW2 = imdilate(BW,SE);
SE2 = strel('diamond',1);
BW3 = imdilate(BW,SE2);
subplot(1,3,1); imshow(BW);
subplot(1,3,2); imshow(BW2);
subplot(1,3,3); imshow(BW3);
```

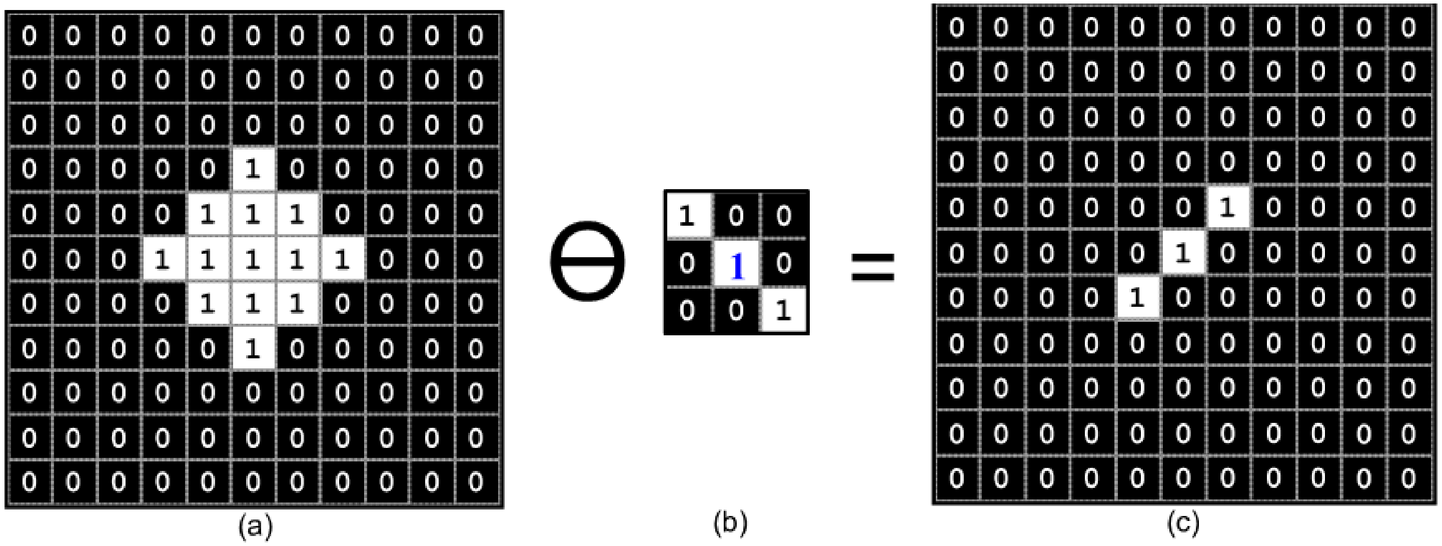


ĂN MÒN EROSION

- Erosion làm mỏng phần ảnh trắng, bào mòn các điểm nhọn, ngược với dilation

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

- SE di chuyển trên ảnh A, độ sáng điểm ảnh ở điểm neo là cực tiểu độ sáng các điểm ảnh bao bởi các phần tử 1 của SE



(a)

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

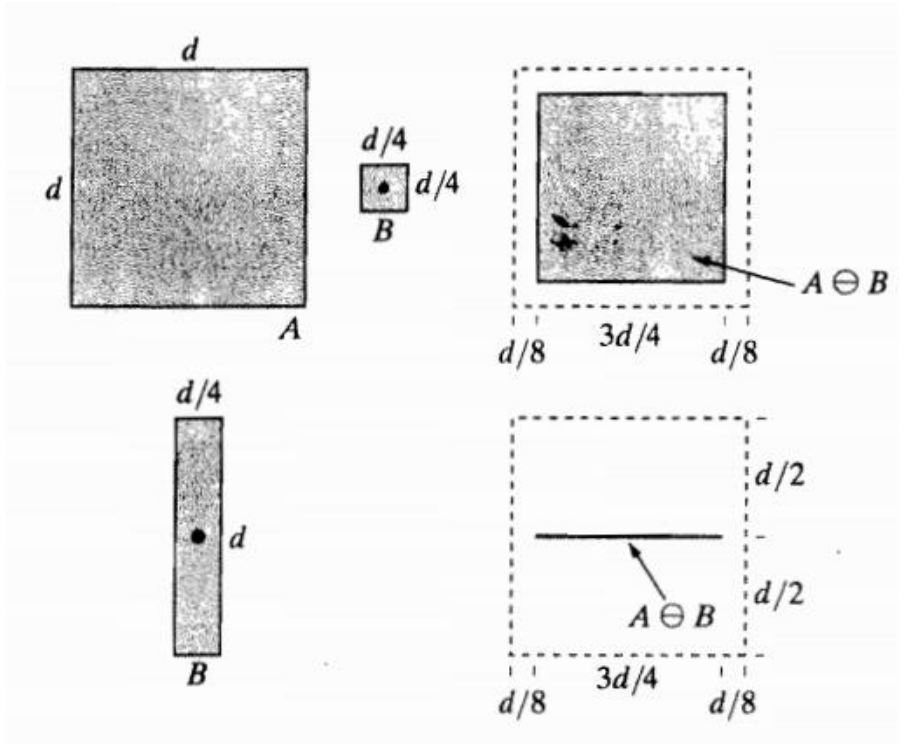
(b)

1	0	0
0	1	0
0	0	1

(c)

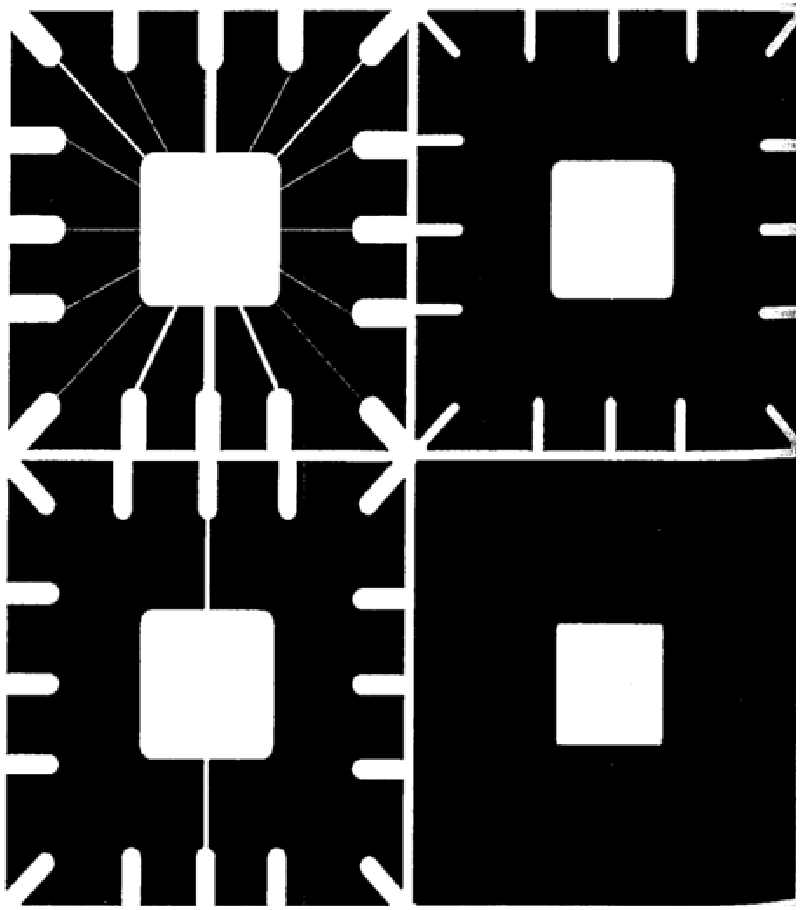
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Illustration of morphological erosion. (a) Original binary image with a diamond object; (b) Structure element with three pixels arranged in a diagonal line at angle of 1350, the origin of the structure element is clearly identified by a red 1; (c) Eroded image, a value of 1 at each location of the origin of the structure element, such that the element overlaps only 1-valued pixels of the input image (i.e., it does not overlap any of the image background).


d
d
A
d/4
d/4
B
A ⊖ B
d/8
3d/4
d/8
d/4
d
B
d/2
d/2
A ⊖ B
d/8
3d/4
d/8

a b
c d

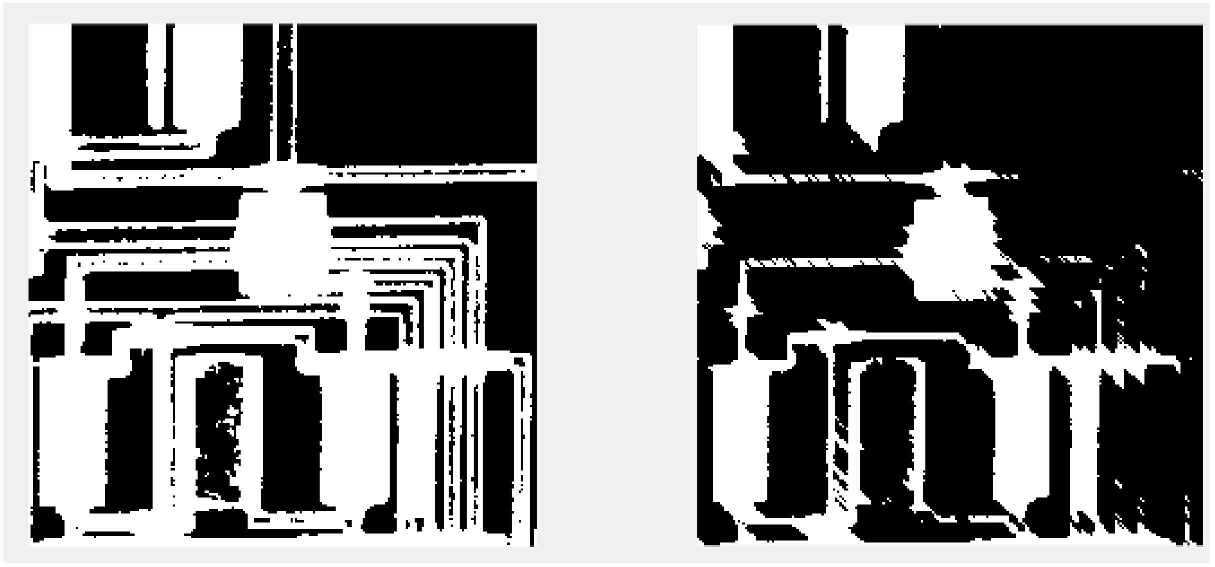
An illustration of erosion.
(a) Original image.
(b) Erosion with a disk of radius 10.
(c) Erosion with a disk of radius 5.
(d) Erosion with a disk of radius 20.

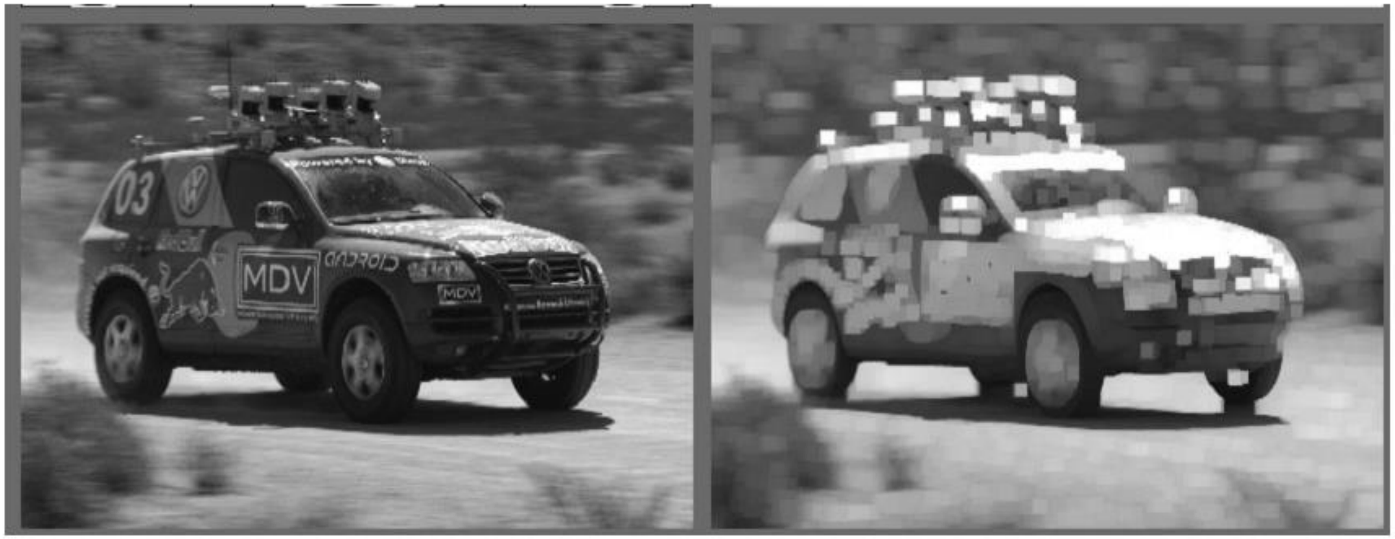


```
A = imread('wirebond_mask.tif');
se = strel('disk', 10);
A2 = imerode(A, se);
imshow(A2)
```

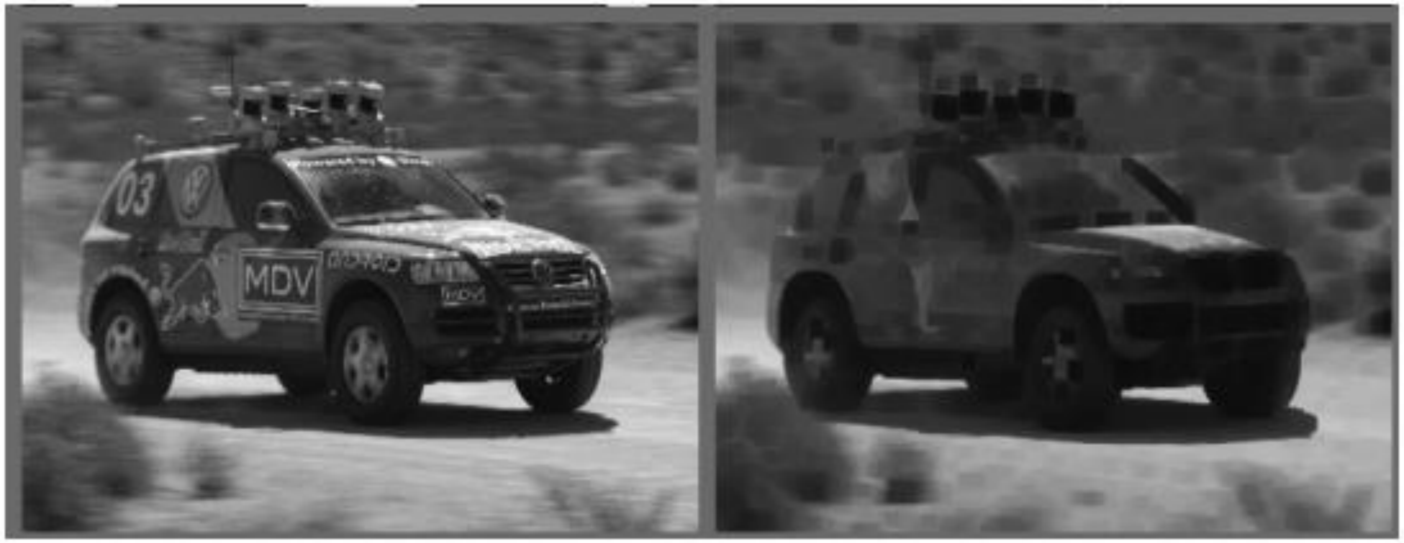
EROSION MATLAB

```
SE = strel('arbitrary',eye(7));
BW1 = imread('circbw.tif');
BW2 = imerode(BW1,SE);
subplot(1,2,1),imshow(BW1);
subplot(1,2,2),imshow(BW2);
```

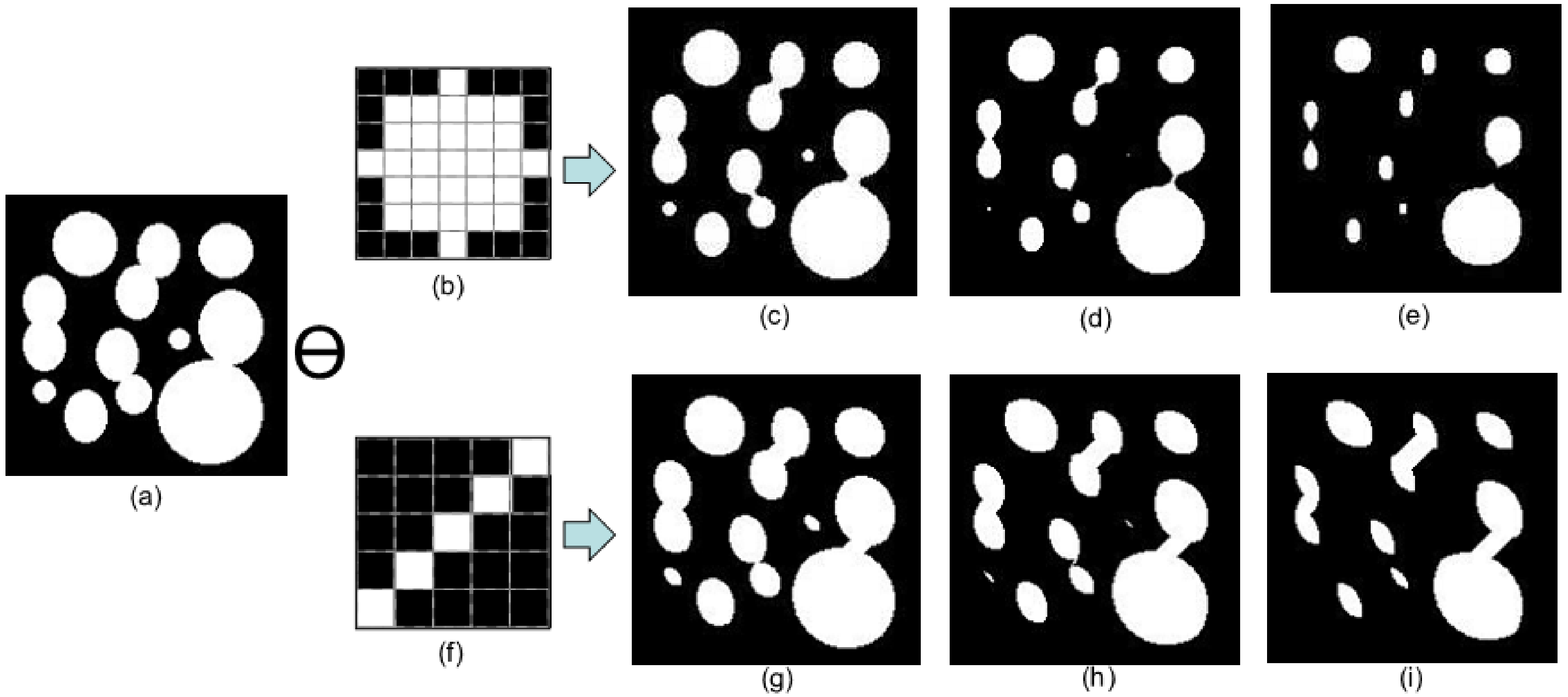




Results of the dilation, or "max", operator: bright regions are expanded and often joined

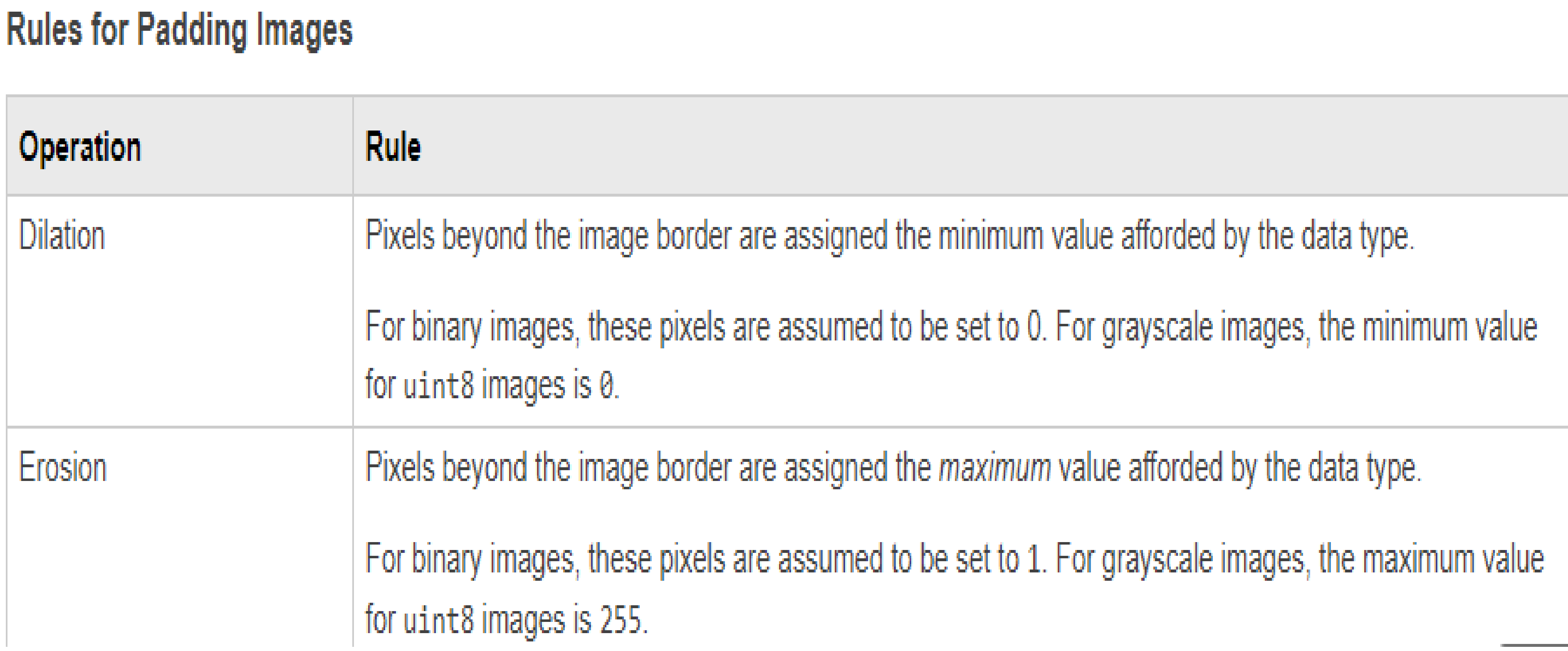


Results of the erosion, or "min," operator: bright regions are isolated and shrunk



Example of morphological erosion. (a) A binary input image; (b) A disk structure element; (c) Eroded image of (a) by SE (b); (d) After twice erosion by SE (b); (e) After three times erosion by SE (b); (f) A line structure element; (g) Eroded image of (a) by SE (f); (h) After twice erosion by SE (f); (i) After three times erosion by SE (f).

- Với các điểm ảnh ở biên ,Matlab bổ sung thêm các điểm ảnh để có thể tính cường độ pixel ở điểm neo



Rules for Padding Images

Operation	Rule
Dilation	Pixels beyond the image border are assigned the minimum value afforded by the data type. For binary images, these pixels are assumed to be set to 0. For grayscale images, the minimum value for `uint8` images is `0`.
Erosion	Pixels beyond the image border are assigned the *maximum* value afforded by the data type. For binary images, these pixels are assumed to be set to 1. For grayscale images, the maximum value for `uint8` images is 255.

KERNEL OPENCV

- Khai báo kernel kích thước lẻ với anchor point là tâm kernel nếu dùng Point(-1,-1), Point(x,y) là tọa độ anchor với gốc tọa độ top left kernel

```
cv::Mat cv::getStructuringElement(
  int       shape,                    // Element shape, e.g., cv::MORPH_RECT
  cv::Size  ksize,                    // Size of structuring element (odd num!)
  cv::Point anchor = cv::Point(-1,-1) // Location of anchor point
);
```

Value of shape	Element	Description
cv::MORPH_RECT	Rectangular	$E_{i,j} = 1, \forall\, i, j$
cv::MORPH_ELLIPSE	Elliptic	Ellipse with axes ksize.width and ksize.height.
cv::MORPH_CROSS	Cross-shaped	$E_{i,j} = 1$, iff $i == anchor.y$ or $j == anchor.x$

ERODE DILATE C

C++: void **erode**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar& **borderValue**=morphologyDefaultBorderValue())

Python: `cv2.erode`(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst

C: void **cvErode**(const CvArr* **src**, CvArr* **dst**, IplConvKernel* **element**=NULL, int **iterations**=1)

Python: `cv.Erode`(src, dst, element=None, iterations=1) → None

Parameters

src – input image; the number of channels can be arbitrary, but the depth should be one of `CV_8U, CV_16U, CV_16S, CV_32F' or ''CV_64F`.

dst – output image of the same size and type as `src`.

element – structuring element used for erosion; if `element=Mat()` , a `3 x 3` rectangular structuring element is used.

anchor – position of the anchor within the element; default value `(-1, -1)` means that the anchor is at the element center.

iterations – number of times erosion is applied.

borderType – pixel extrapolation method (see `borderInterpolate()` for details).

borderValue – border value in case of a constant border (see `createMorphologyFilter()` for details).

The function erodes the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the minimum is taken:

$$\texttt{dst}(x,y) = \min_{(x',y'):\,\texttt{element}(x',y')\neq 0} \texttt{src}(x+x',y+y')$$

The function supports the in-place mode. Erosion can be applied several (`iterations`) times. In case of multi-channel images, each channel is processed independently.

DILATE OPENCV

C++: void **dilate**(InputArray **src**, OutputArray **dst**, InputArray **kernel**, Point **anchor**=Point(-1,-1), int **iterations**=1, int **borderType**=BORDER_CONSTANT, const Scalar& **borderValue**=morphologyDefaultBorderValue())

Python: cv2.**dilate**(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst

C: void **cvDilate**(const CvArr* **src**, CvArr* **dst**, IplConvKernel* **element**=NULL, int **iterations**=1)

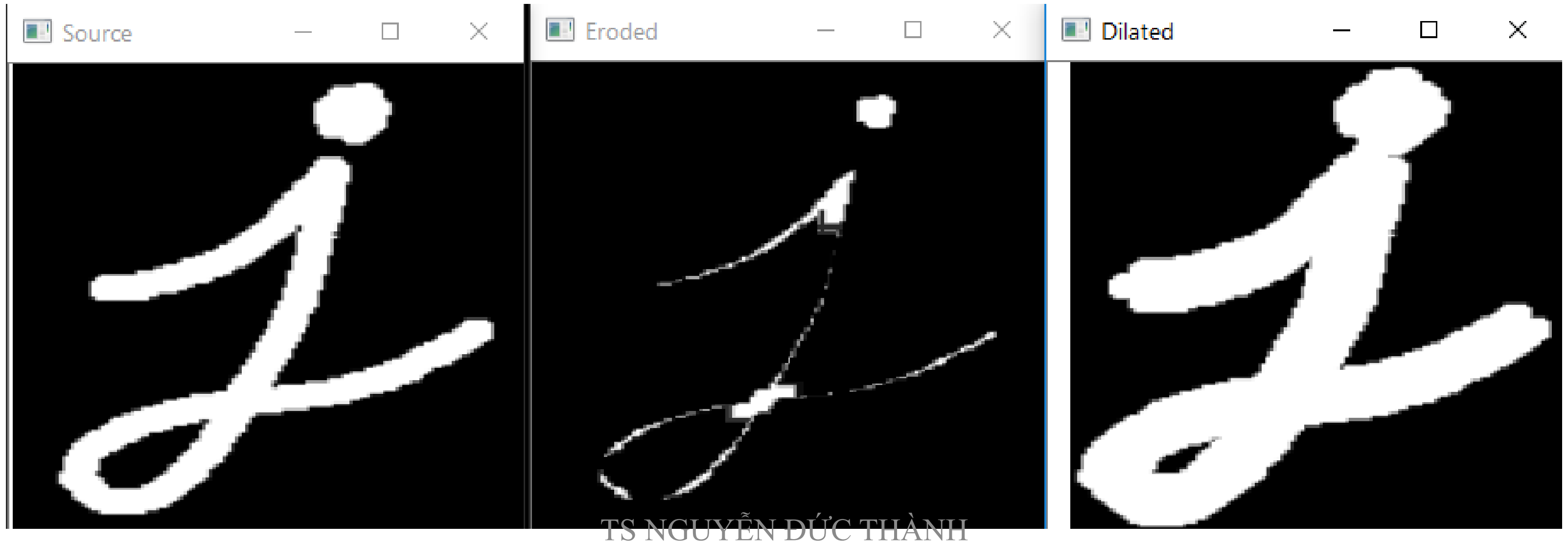
$$\texttt{dst}(x,y) = \max_{(x',y'):\,\texttt{element}(x',y')\neq 0} \texttt{src}(x+x', y+y')$$

ERODE DILATE C++

```
#include "opencv2/opencv.hpp"
#include <iostream>
 using namespace cv;
using namespace std;
 int main( )
{ Mat image,dst1, dst2 ;
  image = imread(“d:/j.png", 1);
int erosion_size = 6;   Mat element =
getStructuringElement(cv::MORPH_CROSS,
              cv::Size(2 * erosion_size + 1, 2 * erosion_size + 1),
              cv::Point(erosion_size, erosion_size) );//anchor at center
erode(image,dst1,element);
```

ERODE DILATE C++

```
dilate(image,dst2,element);
namedWindow("Source", 0);
namedWindow("Eroded", 0);
namedWindow("Dilated", 0);
imshow("Source", image);
imshow("Eroded", dst1);
imshow("Dilated", dst2);
waitKey(0); return 0; }
```



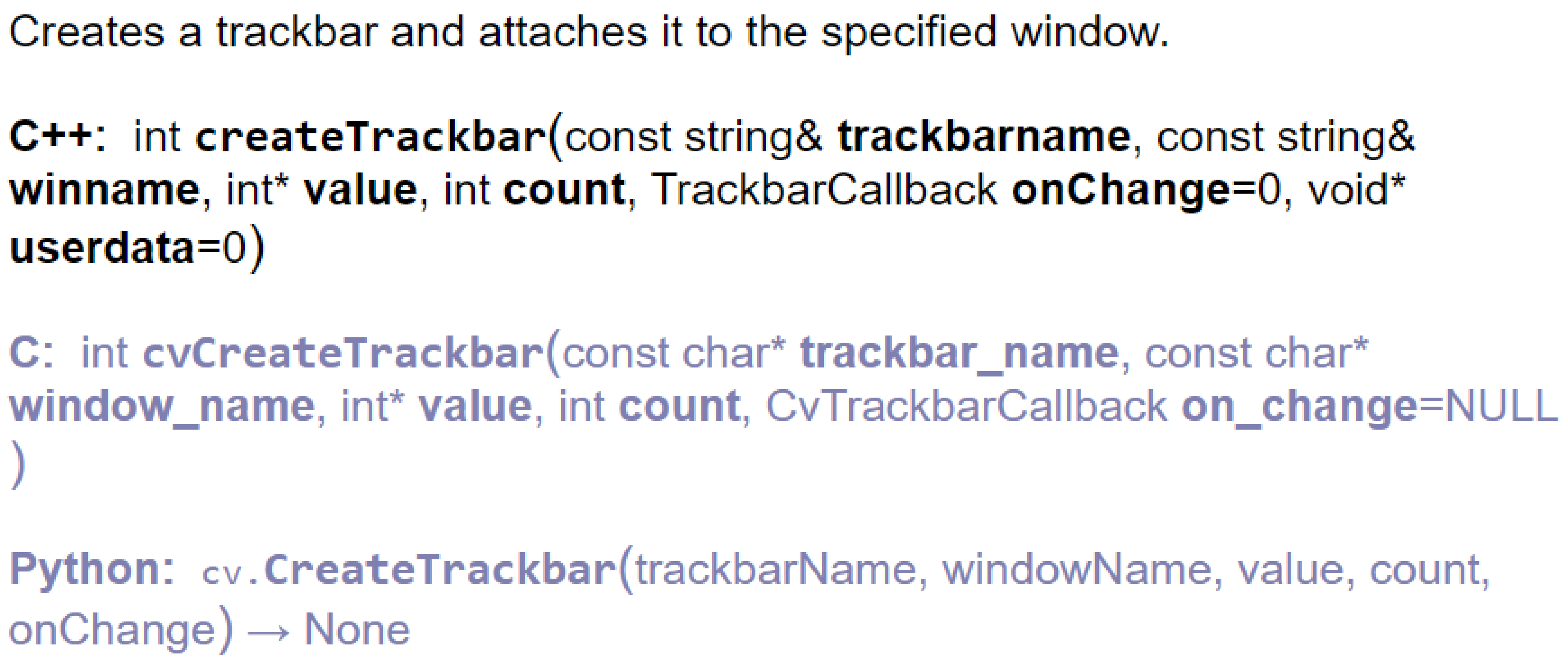
KERNEL

- Nếu muốn tạo kernel khác ta khai báo ma trân kernel. Ví dụ kernel là vạch nằm ngang

```
Mat element = (cv::Mat_<uchar>(5,5) <<
                        0,0,0,0,0,
                        0,0,0,0,0,
                        1,1,1,1,1,
                         0,0,0,0,0,
                         0,0,0,0,0);
erode(source, dst1, element); //anchor at the center
erode(source, dst1, element,Point(3,5));//anchor hàng giữa bên phải
```

TRACKBAR

- Đôi khi ta muốn thay đổi một giá trị biến nào đó trong chương trình mà không muốn sửa chương trình, lúc đó dùng Trackbar với hàm createTrackbar()



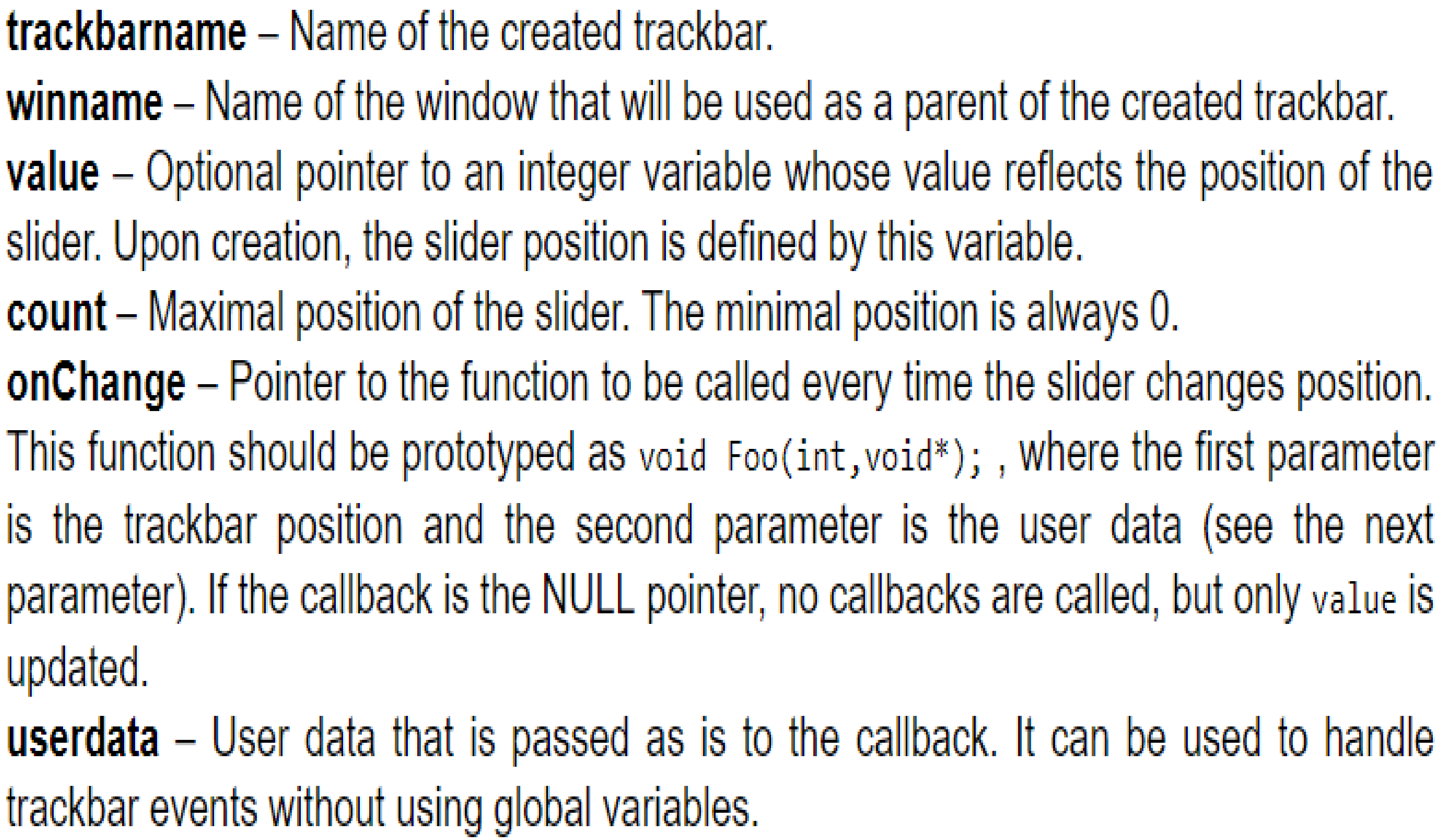
Creates a trackbar and attaches it to the specified window.

C++: int **createTrackbar**(const string& **trackbarname**, const string& **winname**, int* **value**, int **count**, TrackbarCallback **onChange**=0, void* **userdata**=0)

C: int **cvCreateTrackbar**(const char* **trackbar_name**, const char* **window_name**, int* **value**, int **count**, CvTrackbarCallback **on_change**=NULL)

Python: cv.**CreateTrackbar**(trackbarName, windowName, value, count, onChange) → None

TRACKBAR



trackbarname – Name of the created trackbar.

winname – Name of the window that will be used as a parent of the created trackbar.

value – Optional pointer to an integer variable whose value reflects the position of the slider. Upon creation, the slider position is defined by this variable.

count – Maximal position of the slider. The minimal position is always 0.

onChange – Pointer to the function to be called every time the slider changes position. This function should be prototyped as `void Foo(int,void*);` , where the first parameter is the trackbar position and the second parameter is the user data (see the next parameter). If the callback is the NULL pointer, no callbacks are called, but only `value` is updated.

userdata – User data that is passed as is to the callback. It can be used to handle trackbar events without using global variables.

EROSION DILATE TRACKBAR

- Load an image (BGR or grayscale)
- Create two windows (one for dilation output, the other for erosion)
- Create a set of two Trackbars for each operation:
- The first trackbar "Element" returns either erosion_elem or dilation_elem
- The second trackbar "Kernel size" return erosion_size or dilation_size for the corresponding operation.
- Every time we move any slider, the user's function Erosion or Dilation will be called and it will update the output image based on the current trackbar values.

EROSION DILATE TRACKBAR

```
#include "stdafx.h"
#include "opencv2/opencv.hpp"
using namespace cv;
Mat src, erosion, dilation;
int erosion_elem = 0;
int erosion_size = 0;
int dilation_elem = 0;
int dilation_size = 0;
int const max_elem = 2;
int const max_kernel_size = 21;
void Erosion(int, void*);
void Dilation(int, void*);
int main()
{
```

EROSION DILATE TRACKBAR

```
src = imread("d:/mouse.jpg");
namedWindow("Erosion",1); namedWindow("Dilation");
moveWindow("Dilation", 1.5*src.cols, 0);//move to new position
createTrackbar("KernelType", "Erosion", &erosion_elem, max_elem,
Erosion);
createTrackbar("KernelSize", "Erosion",&erosion_size,
ax_kernel_size,Erosion);
createTrackbar("KernelType", "Dilation",&dilation_elem,
max_elem,Dilation);
createTrackbar("KernelSize", "Dilation",&dilation_size,
max_kernel_size, Dilation);
Erosion(0, 0);Dilation(0, 0);
waitKey(0);return 0;}
```

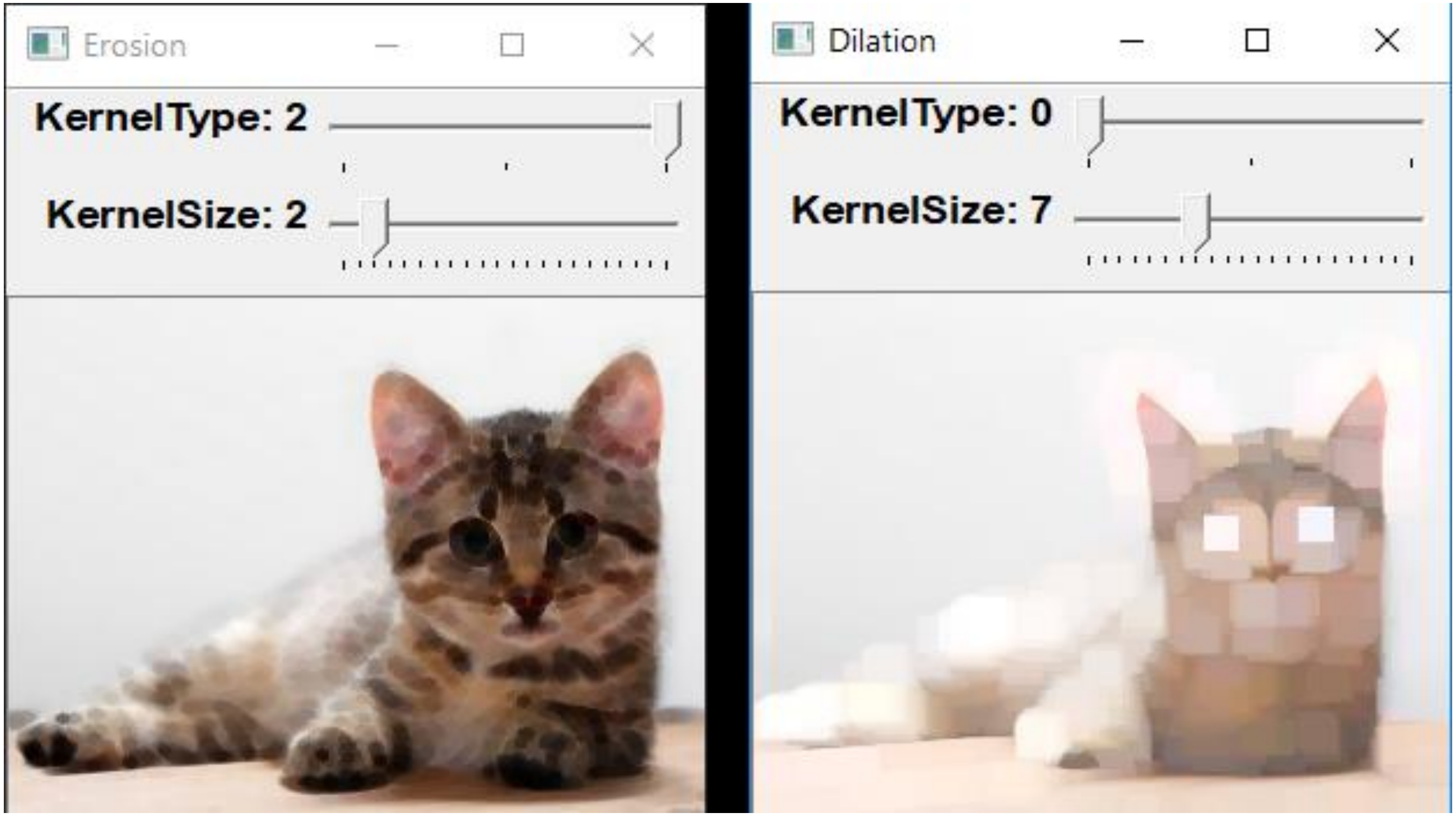
EROSION DILATE TRACKBAR

```
void Erosion(int, void*)
{
		int erosion_type = 0;
		if (erosion_elem == 0) { erosion_type = MORPH_RECT; }
		else if (erosion_elem == 1) { erosion_type = 
MORPH_CROSS; }
		else if (erosion_elem == 2) { erosion_type = 
MORPH_ELLIPSE; }
		Mat element = getStructuringElement(erosion_type,
				Size(2 * erosion_size + 1, 2 * erosion_size + 1),
				Point(erosion_size, erosion_size));
		erode(src, erosion, element);
		imshow("Erosion", erosion);}
```

EROSION DILATE TRACKBAR

```
void Dilation(int, void*)
{
        int dilation_type = 0;
        if (dilation_elem == 0) { dilation_type = MORPH_RECT; }
        else if (dilation_elem == 1) { dilation_type = 
MORPH_CROSS; }
        else if (dilation_elem == 2) { dilation_type = 
MORPH_ELLIPSE; }
        Mat element = getStructuringElement(dilation_type,
                Size(2 * dilation_size + 1, 2 * dilation_size + 1),
                Point(dilation_size, dilation_size));
        dilate(src, dilation, element);
        imshow("Dilation", dilation);
}
```

EROSION DILATE TRACKBAR



KernelType: 2

KernelSize: 2

KernelType: 0

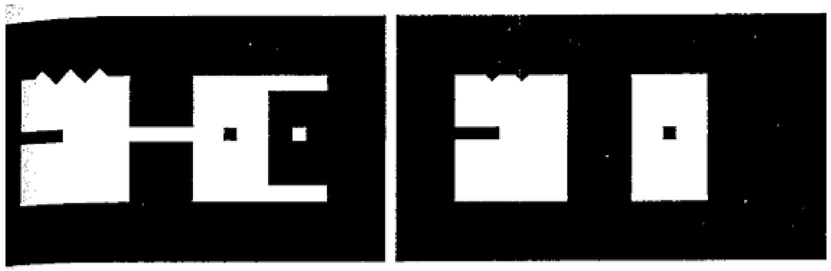
KernelSize: 7

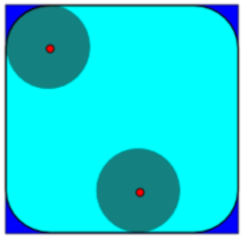
KẾT HỢP DILATON VÀ EROSION

- Opening: Erosion rồi Dilation, làm trơn biên, bỏ các điểm nhọn , các đường nối mỏng, xóa các đốm sáng nhỏ

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

- Cho SE thường là dạng đĩa, vuông, di chuyển trong phần ảnh sáng của ảnh, không ra ngoài ảnh, không chạm phần ảnh tối, phần ảnh sáng không thể được bao bởi SE sẽ lấy giá trị tối,

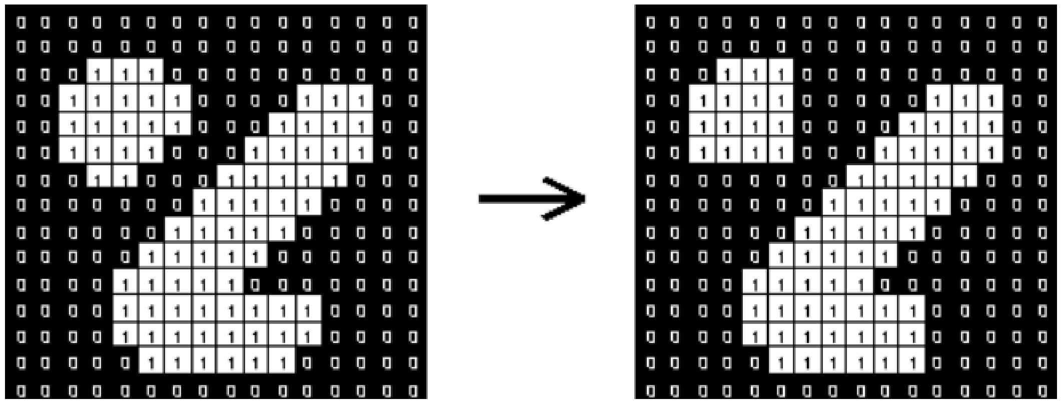




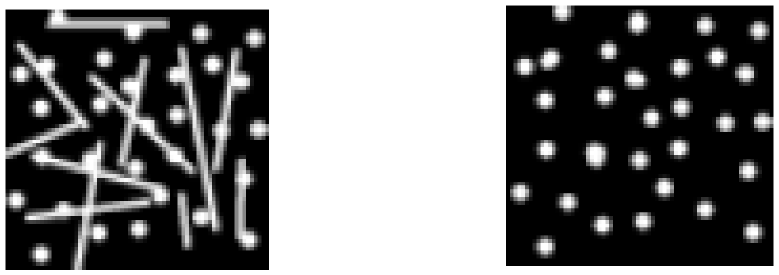
- Closing: Dilation rồi Erosion, làm trơn biên, tạo các đường nối, lấp các lỗ nhỏ tối, ngược với opening. Cho SE di chuyển trong vùng ảnh tối, không ra ngoài ảnh, không chạm phần ảnh sáng, phần ảnh tối không thể được bao bởi SE sẽ lấy giá trị sáng

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

OPENING



Opening với SE chữ nhật 3x3



Opening SE đĩa 11

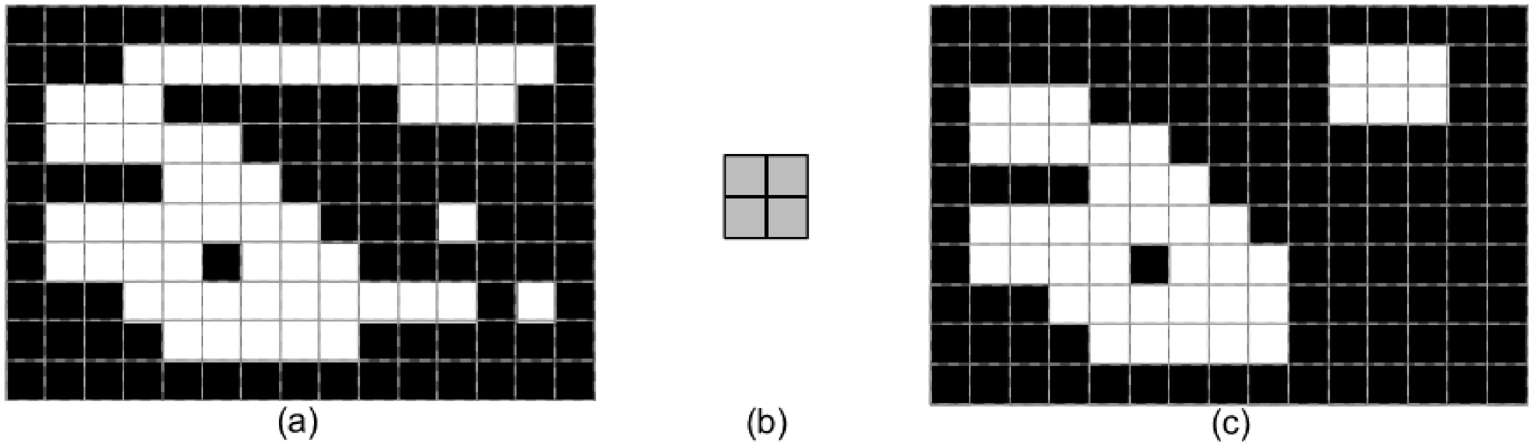
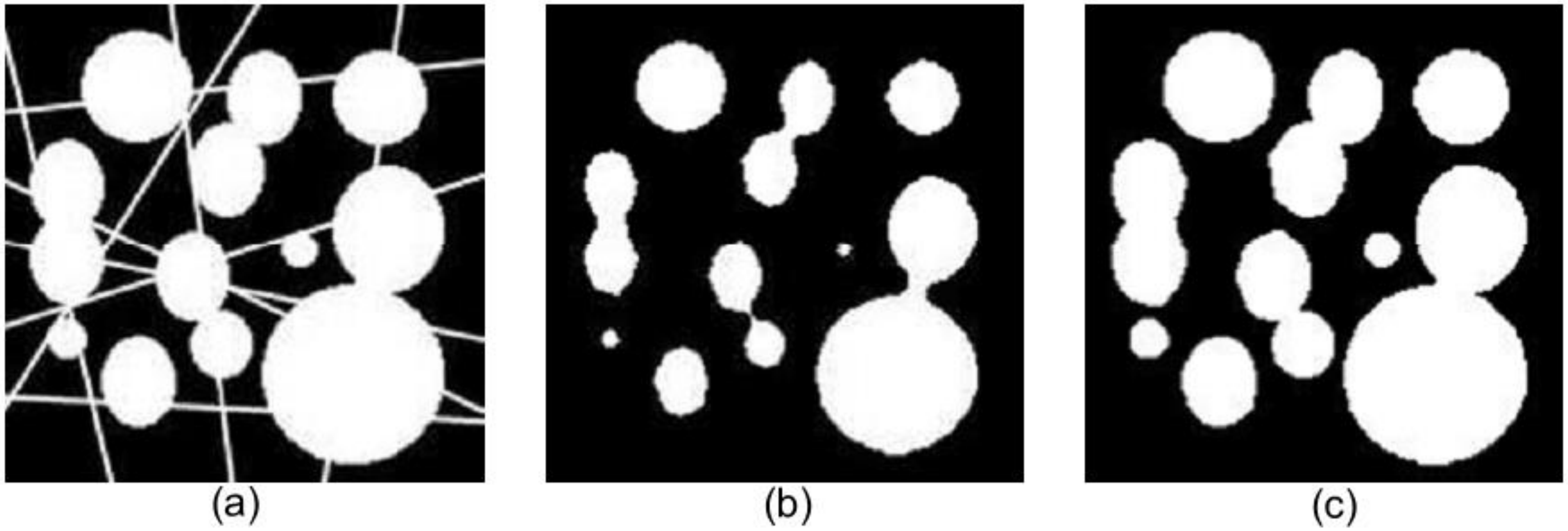
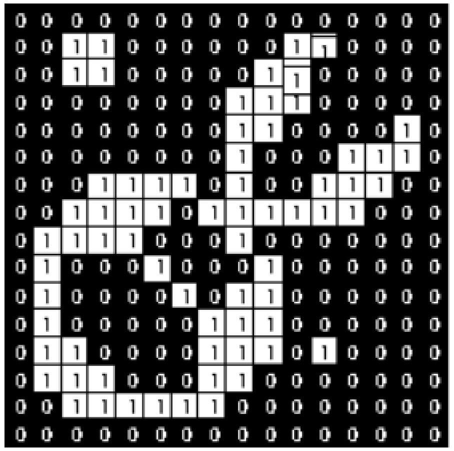


Illustration of opening. (a) A 10 × 15 discrete binary image (the object pixels are the white pixels); (b) A 2 × 2 structure element; (c) Opening of image (a) by SE (b). All object pixels that cannot be covered by the structure element when it fits the object pixels are removed.



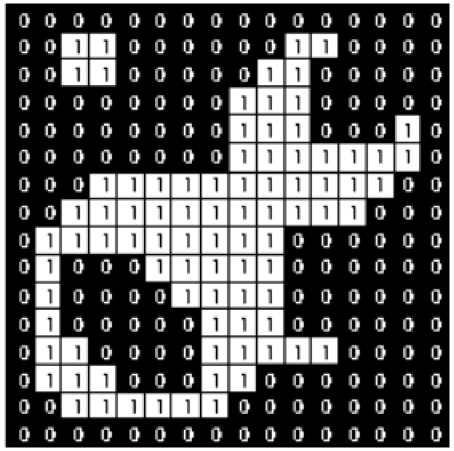
Shape matching by opening. (a) An original binary image A with some disks and lines; (b) Eroded image AΘB by a disk structure element B, where the radius of the disk is 5 pixels; (c) Dilated image of (b) by the same disk structure element B: (AΘB) $\oplus$ B.

CLOSING



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

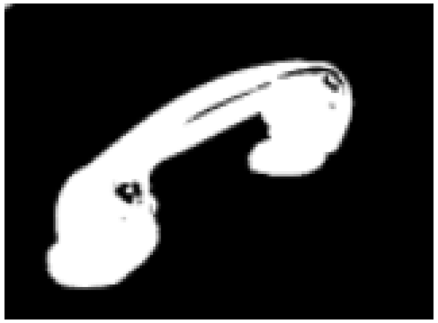
$\longrightarrow$



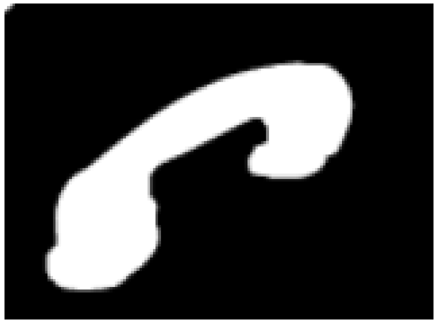
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	1	0	0	0	1	1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

Closing với SE chữ nhật 3x3

Ảnh xám

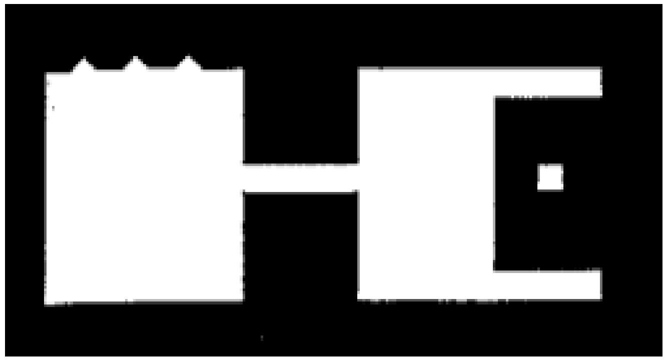


Ảnh nhị phân



Closing

OPENING CLOSING



```
f = imread('shapes.tif');
se = strel('square', 20);
fo = imopen(f, se);
imshow(fo)

fc = imclose(f, se);
imshow(fc)
```

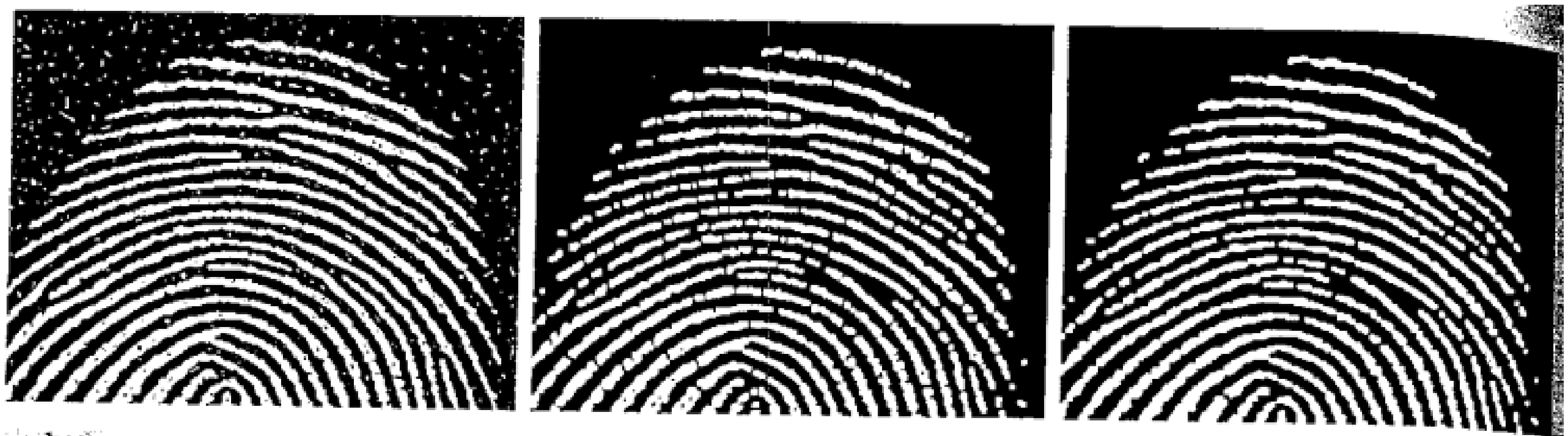
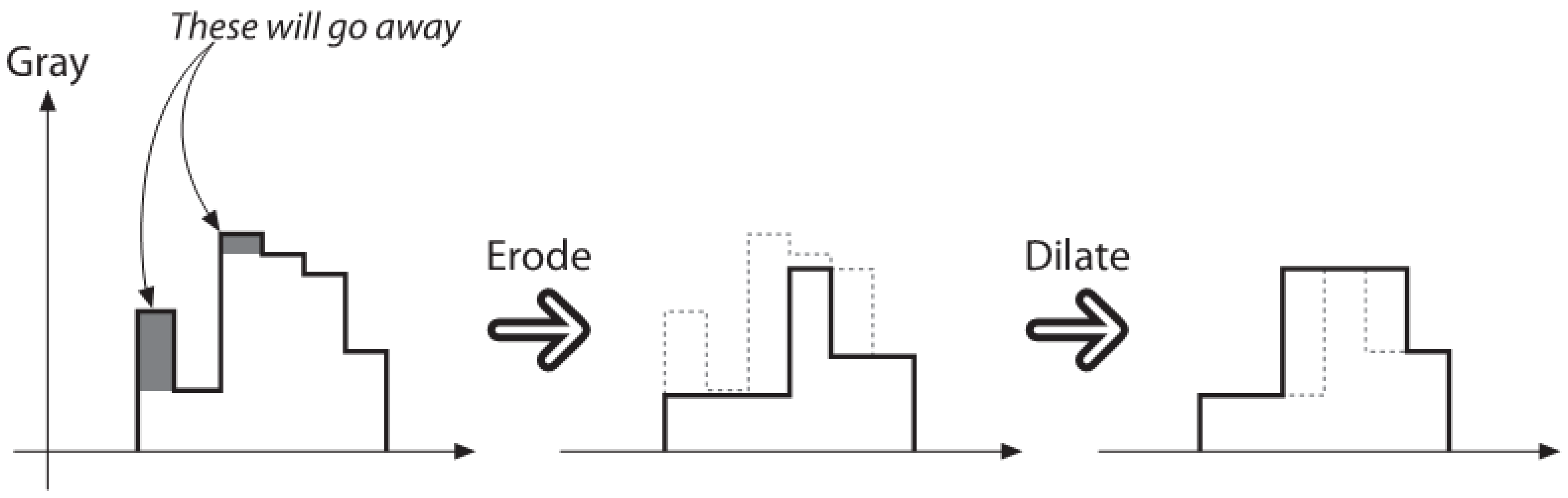


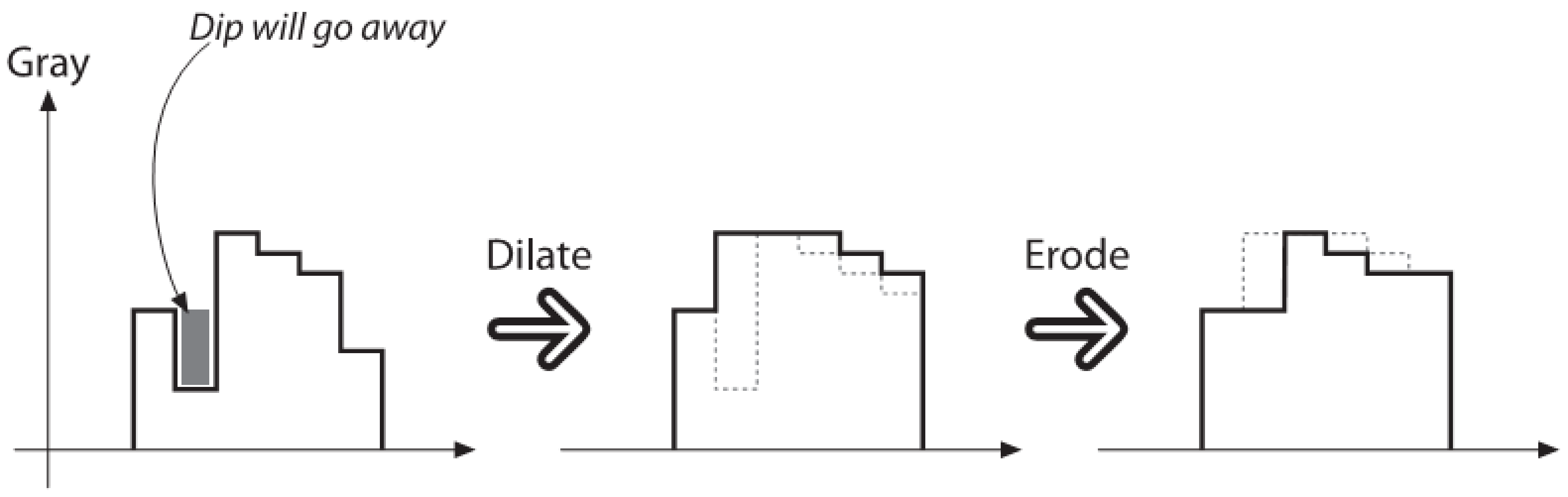
FIGURE 9.11 (a) Noisy fingerprint image. (b) Opening of image. (c) Opening followed by closing. (Original image courtesy of the National Institute of Standards and Technology.)

```
f = imread('fingerprint.tif');
se = strel('square', 3);
fo = imopen(f, se);
imshow(fo)

foc = imclose(fo,se);
imshow(foc)
```

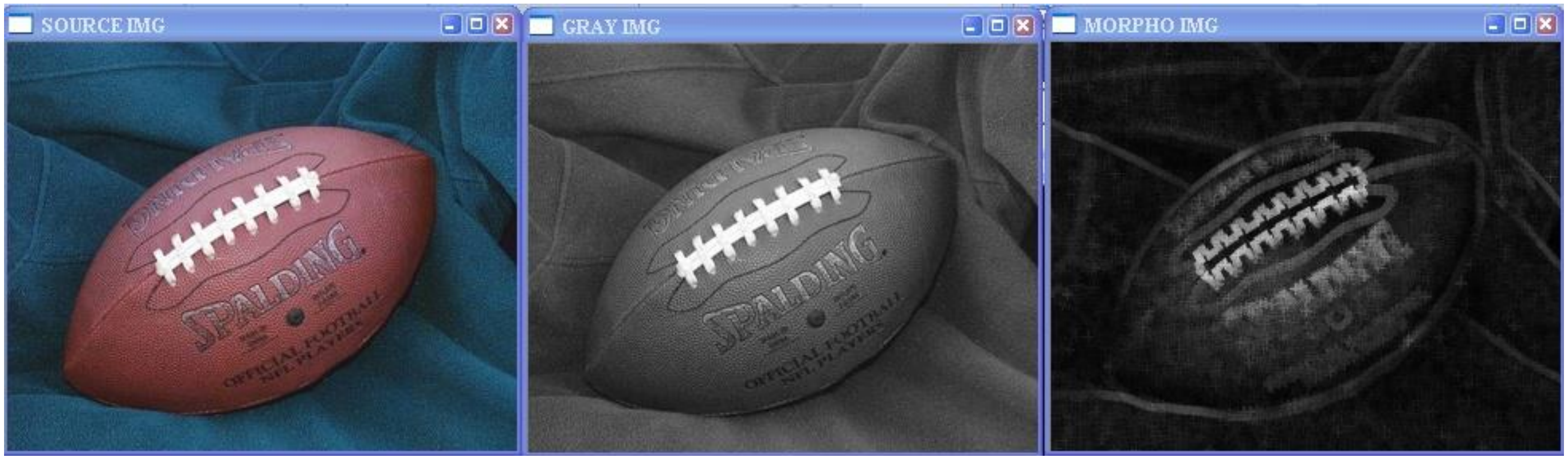


Morphological opening operation: the upward outliers are eliminated as a result

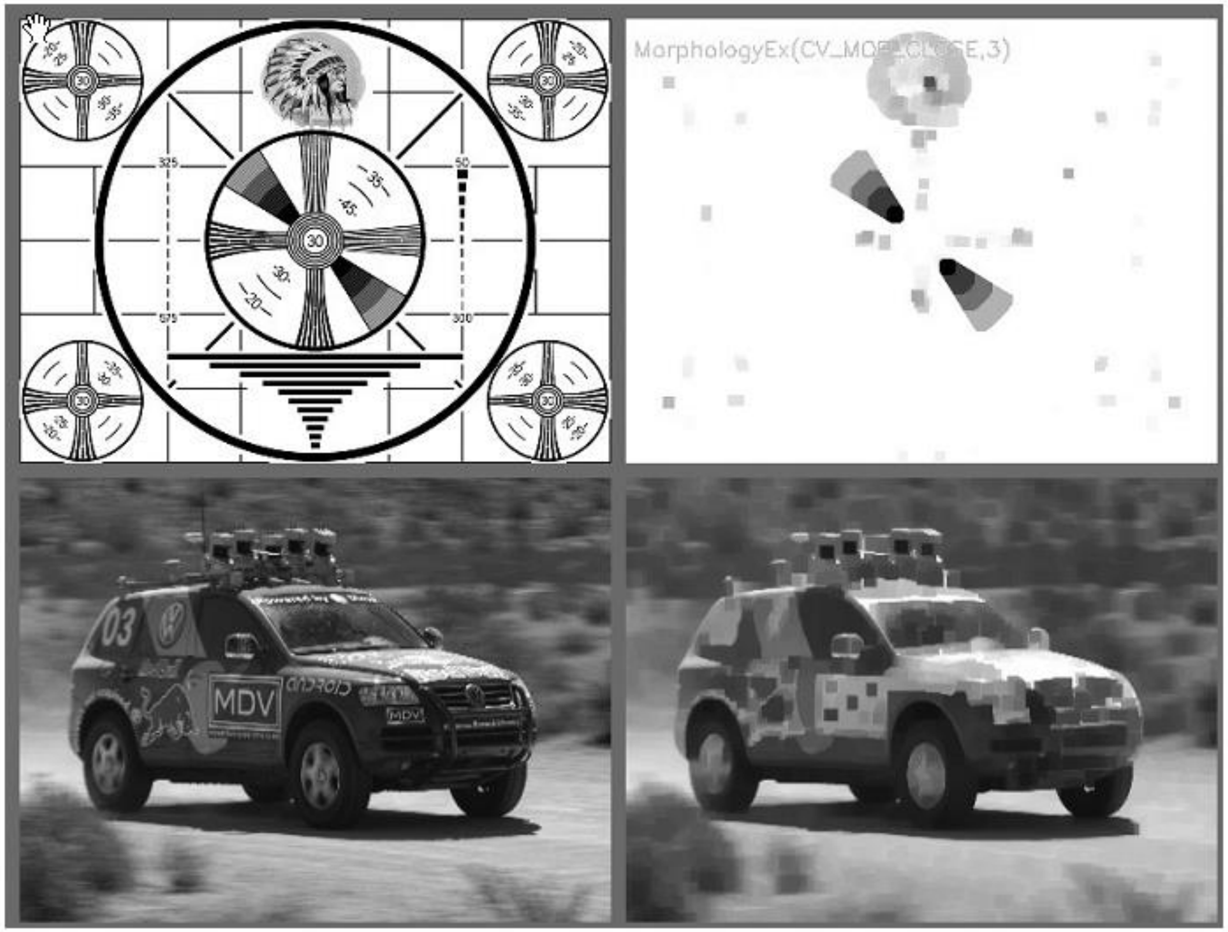


Morphological closing operation: the downward outliers are eliminated as a result

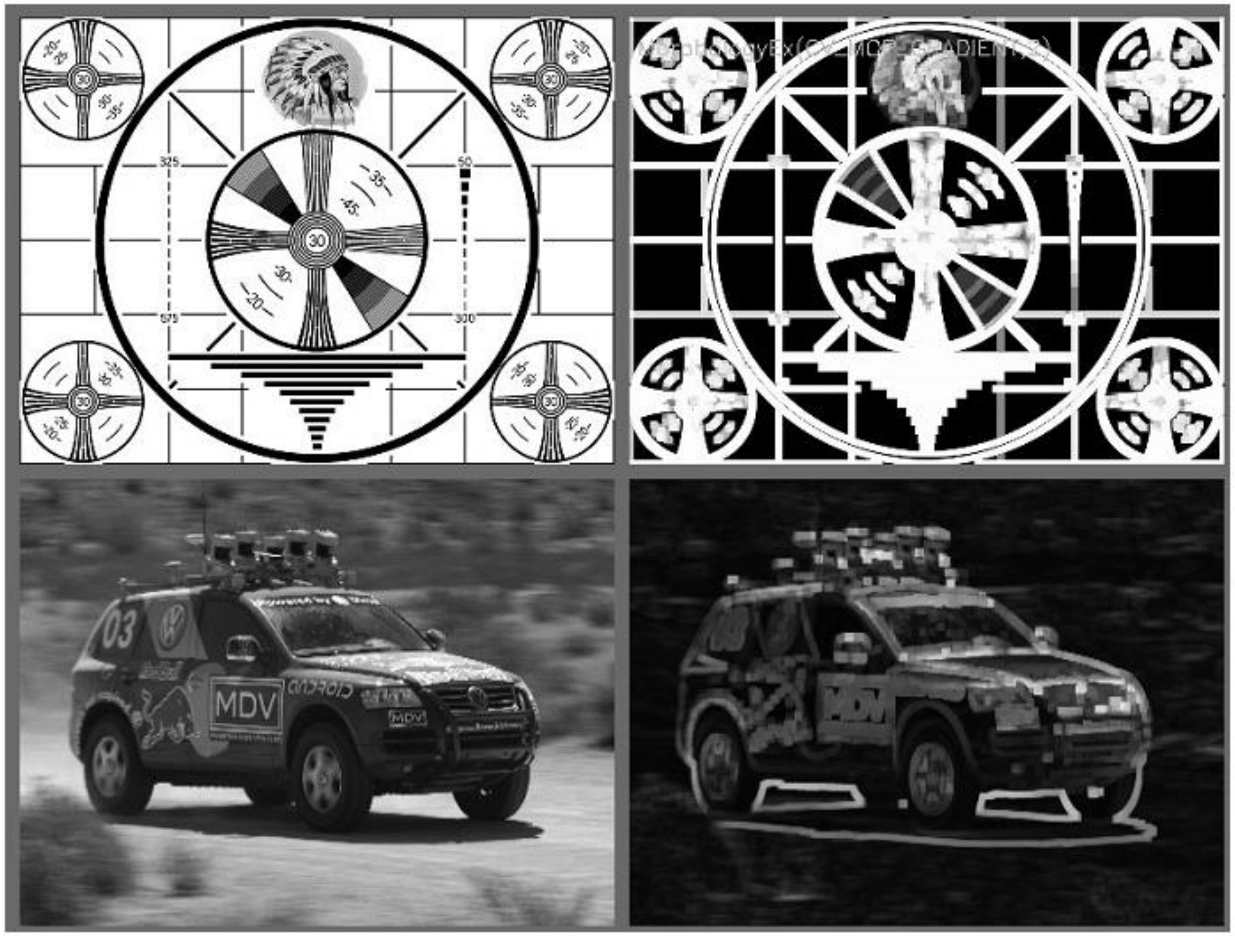
KẾT QUẢ OPENING



KẾT QUẢ CLOSING



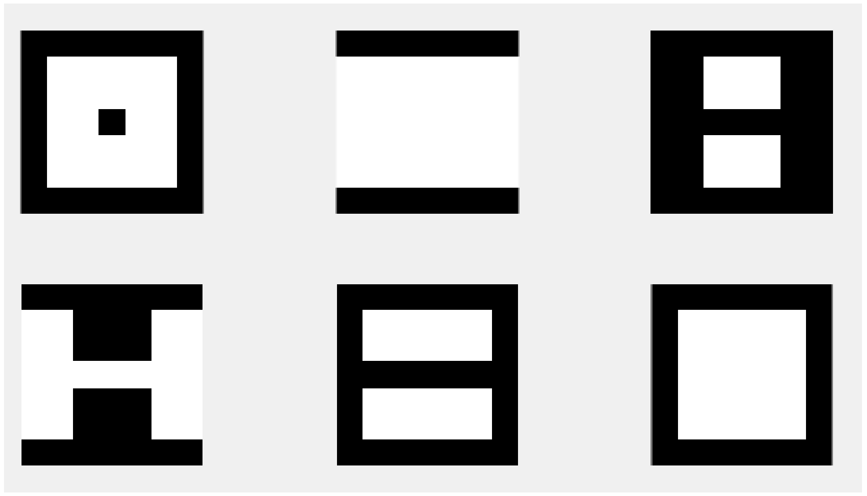
KẾT QUẢ GRADIENT=DILATION-EROSION



Làm nổi bật các đường biên

DILATION, EROSION, GRADIENT, OPENING CLOSING

```
close all;
I = [0 0 0 0 0 0 0; 0 1 1 1 1 1 0; 0 1 1 1 1 1 0; 0 1 1
0 1 1 0; 0 1 1 1 1 1 0; 0 1 1 1 1 1 0; 0 0 0 0 0 0 0];
se=[1 1 1];
d = imdilate(I, se);e = imerode(I, se); g=d-e;
o=imopen(I,se);c=imclose(I,se),
subplot(3,3,1);imshow(I,[]);
subplot(3,3,2);imshow(d,[]);
subplot(3,3,3);imshow(e,[]);
subplot(3,3,4);imshow(g,[]);
subplot(3,3,5);imshow(o,[]);
subplot(3,3,6);imshow(c,[]);
```



HIT OR MISS TRÚNG HAY TRẬT

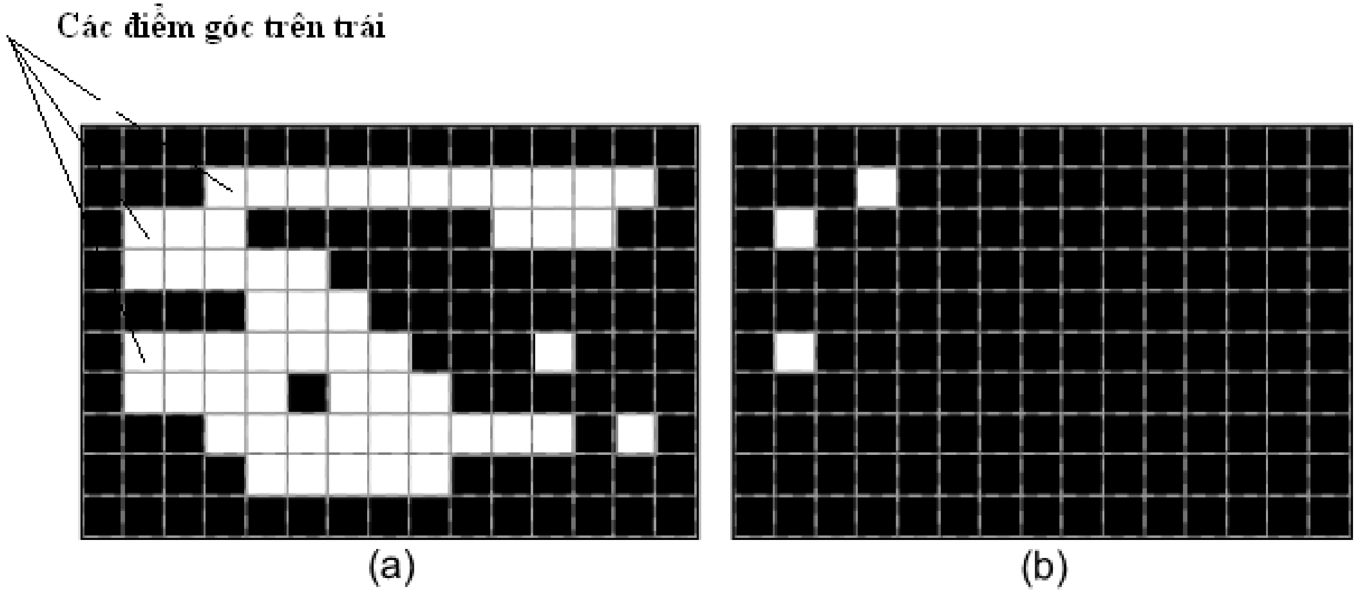
- Dùng để tách vùng ảnh có đặc trưng nào đó

$$A \otimes B = (A \Theta B_1) \cap (A^c \Theta B_2)$$

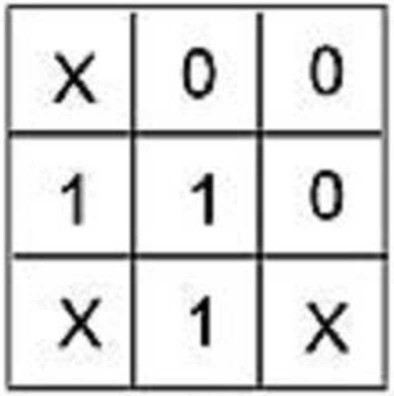
- A được ăn mòn bởi B1, ảnh bù của A được ăn mòn bởi B2, sau đó hai ảnh được giao nhau, B1 và B2 là hai phần của một SE và không giao với nhau, ý nghia thuật toán là tìm điểm ảnh có láng giềng giống B1 và không giống B2
- Thường dùng để phát giác các điểm góc, điểm cuối của một đường
- Ví dụ tìm các điểm góc trên trái của một hình, dùng SE B như sau

$$B = \begin{array}{|c|c|c|} \hline 0 & 0 & X \\ \hline 0 & 1 & 1 \\ \hline X & 1 & X \\ \hline \end{array} \qquad B1 = \begin{array}{|c|c|c|} \hline & & \\ \hline & 1 & 1 \\ \hline & 1 & \\ \hline \end{array} \qquad B2 = \begin{array}{|c|c|c|} \hline 1 & 1 & \\ \hline 1 & & \\ \hline & & \\ \hline \end{array}$$

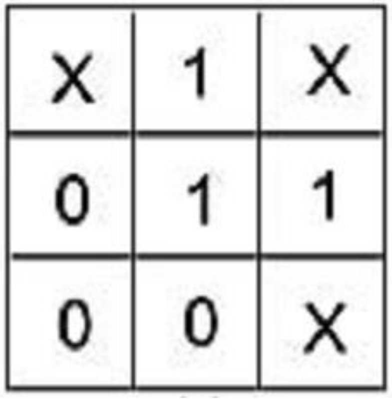
- Cho B di chuyển trên ảnh nhị phân A, nếu vùng ảnh A bao bởi B trùng với mẫu của B thì điểm ảnh đó cho là 1(hit), ngược lại là 0 (miss)



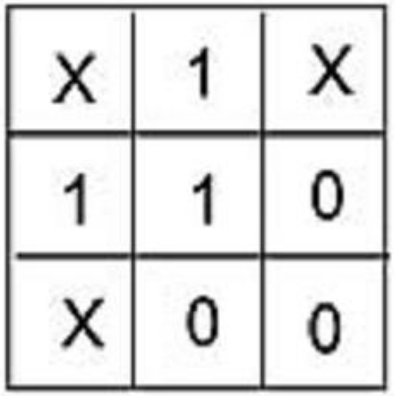
- Thay đổi B ta có thể phát giác các điểm góc trên phải, góc dưới trái, góc dưới phải



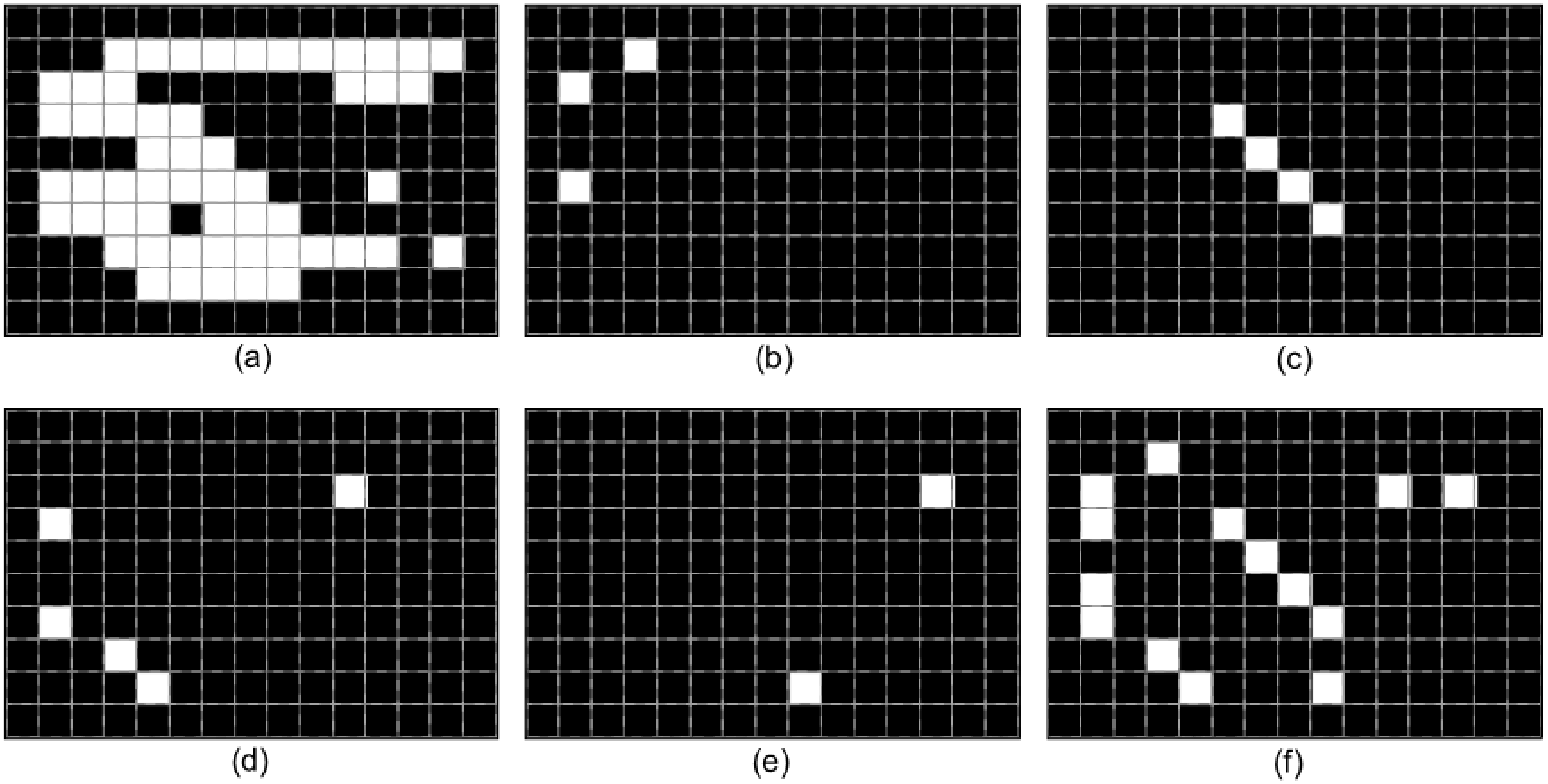
X	0	0
1	1	0
X	1	X



X	1	X
0	1	1
0	0	X



X	1	X
1	1	0
X	0	0



Corner detection by using hit-or-miss transform. (a) A 10 × 15 discrete binary image (the object pixels are the white pixels; (b) Detection for top right corners by hit-or-miss transformation of image(a) by structure element ; (c) Detection for top left corners; (d) Detection for bottom left corners; (e) Detection for bottom right corners; (f) All right-angle corners by applying "OR" operator on (b), (c), (d) and (e).

morphologyEx

Performs advanced morphological transformations.

C++: void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point anchor=Point(-1,-1), int iterations=1, int borderType=BORDER_CONSTANT, const Scalar& borderValue=morphologyDefaultBorderValue())

Python: cv2.morphologyEx(src, op, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]]) → dst

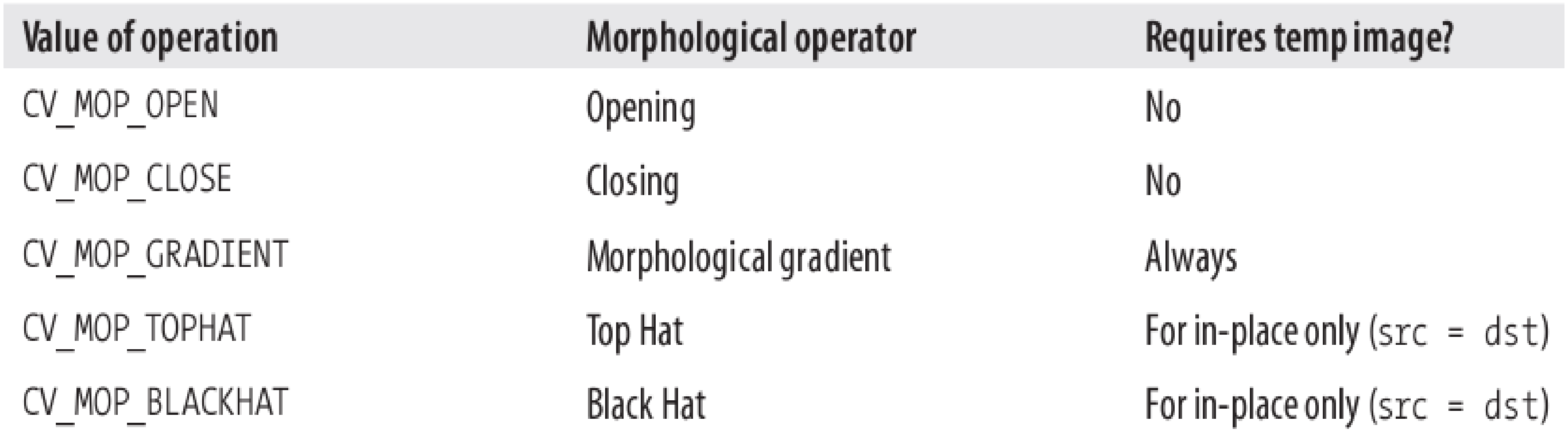
C: void cvMorphologyEx(const CvArr* src, CvArr* dst, CvArr* temp, IplConvKernel* element, int operation, int iterations=1)

Python: cv.MorphologyEx(src, dst, temp, element, operation, iterations=1) → None

```
•  void cvMorphologyEx(
     const CvArr*   src,
     CvArr*         dst,
     CvArr*         temp,
     IplConvKernel* structure element,
     int            operation,
     int            iterations   = 1
   );
```

Temp là ảnh tạm

cvMorphologyEx() operation options



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)

morphologyEx

src – Source image. The number of channels can be arbitrary. The depth should be one of CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.

dst – Destination image of the same size and type as src .

element – Structuring element.

op – Type of a morphological operation that can be one of the following:

MORPH_OPEN - an opening operation (2)

MORPH_CLOSE - a closing operation (3)

MORPH_GRADIENT - a morphological gradient (4)

MORPH_TOPHAT - “top hat” (5)

MORPH_BLACKHAT - “black hat” (6)

MORPH_HITMISS - “hit and miss” (7)

VÍ DỤ C

```
#include "stdafx.h"
#include "cv.h"
#include "highgui.h"
int main( ) {
    IplImage* src = cvLoadImage( "C:/football.jpg" );
    IplImage *gray = cvCreateImage(cvSize(src->width,src-
       >height),IPL_DEPTH_8U, 1);
    cvCvtColor(src, gray, CV_RGB2GRAY);
    IplImage *temp, *dst;
    temp = cvCloneImage(gray);
    dst = cvCloneImage(gray);
    IplConvKernel *se =cvCreateStructuringElementEx( 7, 7, 3,
       3,CV_SHAPE_ELLIPSE);
    cvMorphologyEx( gray, dst, temp, se, CV_MOP_OPEN, 1);
```

VÍ DỤ C

```
cvShowImage( "SOURCE IMG", src );
cvShowImage( "MORPHO IMG",dst );
cvWaitkey(0);
cvReleaseImage( &src );
cvDestroyWindow( "SOURCE IMG");
cvReleaseImage( &gray );
cvReleaseImage( &dst );
cvDestroyWindow( "MORPHO IMG");
cvReleaseStructuringElement(&se );
}
```

HÀM BWMORPH

- Hàm Matlab xử lý hình thái ảnh đen trắng

BW2 = BWMORPH(BW1,OPERATION)

BW2 = BWMORPH(BW1,OPERATION,N) applies the operation N times. N can be Inf, in which case the operation is repeated until the image no longer changes.

- BW1 là ảnh xám, BW2 là ảnh nhị phân
- OPERATION là chuỗi

'bothat'	Subtract the input image from its closing
'branchpoints'	Find branch points of skeleton
'bridge'	Bridge previously unconnected pixels
'clean'	Remove isolated pixels (1's surrounded by 0's)
'close'	Perform binary closure (dilation followed by erosion)
'diag'	Diagonal fill to eliminate 8-connectivity of background
'dilate'	Perform dilation using the structuring element ones(3)
'endpoints'	Find end points of skeleton

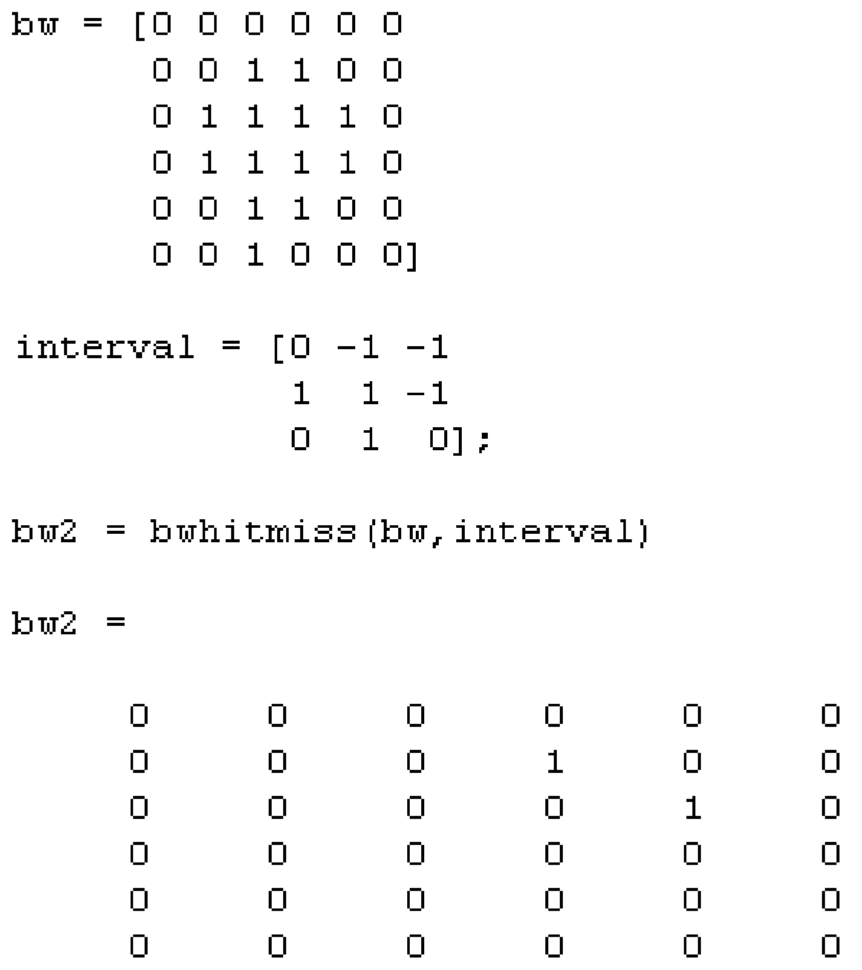
HÀM BWMORPH

'erode'	Perform erosion using the structuring element ones(3)
'fill'	Fill isolated interior pixels (0's surrounded by 1's)
'hbreak'	Remove H-connected pixels
'majority'	Set a pixel to 1 if five or more pixels in its 3-by-3 neighborhood are 1's
'open'	Perform binary opening (erosion followed by dilation)
'remove'	Set a pixel to 0 if its 4-connected neighbors are all 1's, thus leaving only boundary pixels
'shrink'	With N = Inf, shrink objects to points; shrink objects with holes to connected rings
'skel'	With N = Inf, remove pixels on the boundaries of objects without allowing objects to break apart

HÀM BWMORPH

'spur'	Remove end points of lines without removing small objects completely
'thicken'	With N = Inf, thicken objects by adding pixels to the exterior of objects without connected previously unconnected objects
'thin'	With N = Inf, remove pixels so that an object without holes shrinks to a minimally connected stroke, and an object with holes shrinks to a ring halfway between the hold and outer boundary
'tophat'	Subtract the opening from the input image

- BW2 = bwhitmiss(BW1,SE1,SE2)
  BW2 = bwhitmiss(BW1,INTERVAL)
- Ví dụ tìm pixel góc trên phải



```
bw = [0 0 0 0 0 0
      0 0 1 1 0 0
      0 1 1 1 1 0
      0 1 1 1 1 0
      0 0 1 1 0 0
      0 0 1 0 0 0]

interval = [0 -1 -1
            1  1 -1
            0  1  0];

bw2 = bwhitmiss(bw,interval)

bw2 =

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

- Ví dụ tìm pixel góc trên trái

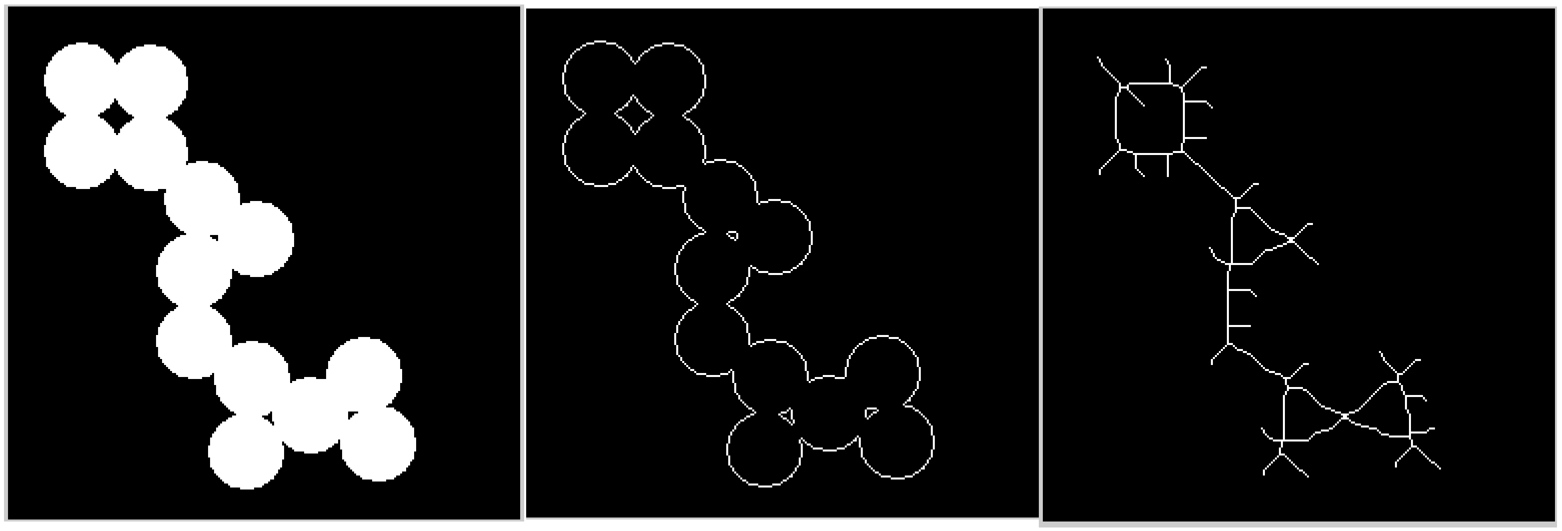
```
B1 = strel([0 0 0; 0 1 1; 0 1 0]);
B2 = strel([1 1 1; 1 0 0; 1 0 0]);

g = bwhitmiss(f, B1 ,B2);
imshow(g)
```

Matlab có hàm tổng hợp thực hiện các phép biến đổi

```
g = bwmorph(f, operation, n)
```

```
BW1 = imread('circles.png');
  figure, imshow(BW1)
  BW2 = bwmorph(BW1,'remove');
  BW3 = bwmorph(BW1,'skel',Inf);
  figure, imshow(BW2)
  figure, imshow(BW3)
```

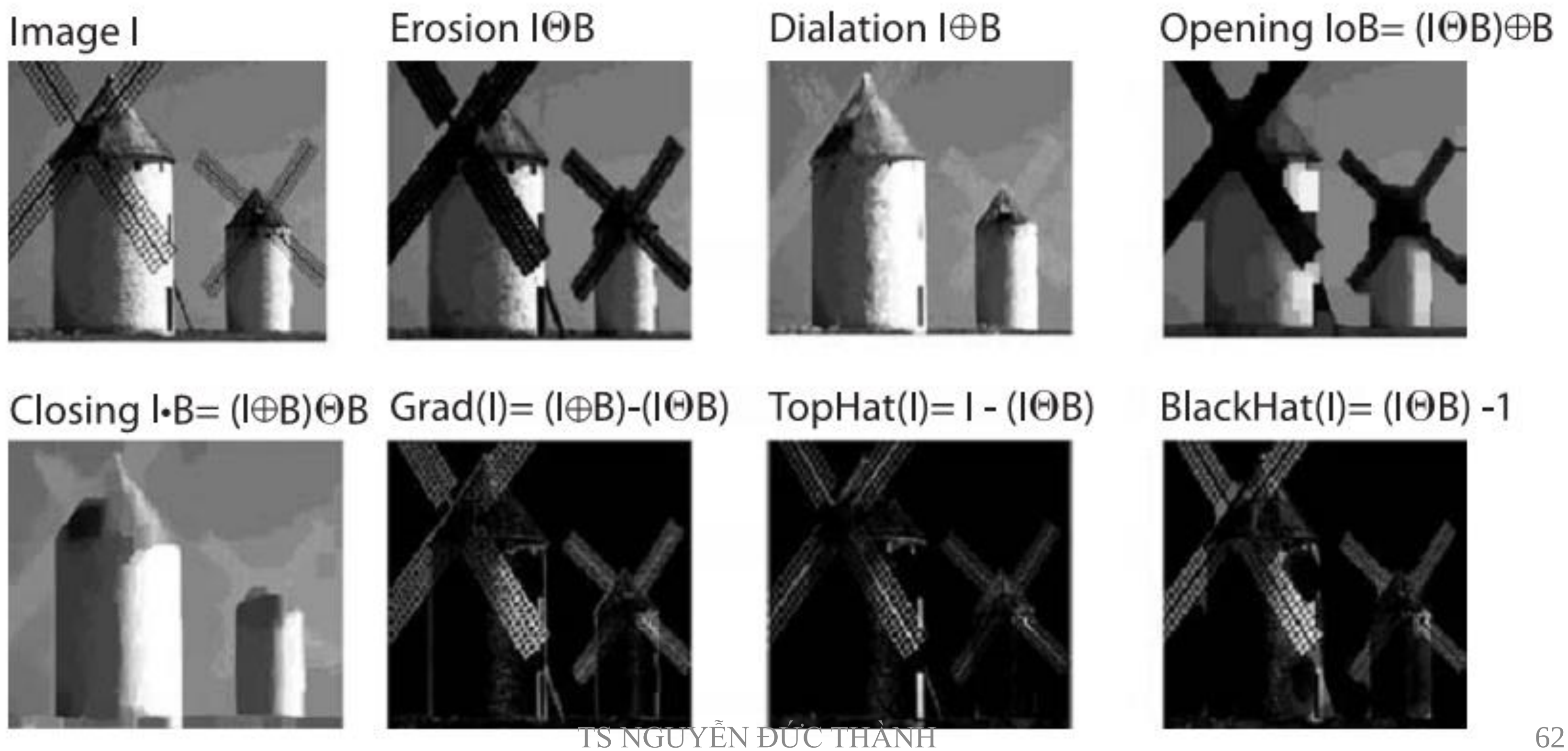


TOP HAT BLACK HAT

- Dùng để cách ly các đốm sáng hơn hay tối hơn lân cận

$$TopHat(src) = src - open(src)$$

$$BlackHat(src) = close(src) - src$$



THINNING THICKEN

- Thinning dùng để tạo các đường bề dầy một pixel
- Ảnh A trừ với ảnh hit or miss

$$A \phi B = A - (A \otimes B)$$

- Tám SE có thể dùng cho Thinning

$$\begin{bmatrix} 0 & 0 & 0 \\ X & 1 & X \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} X & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & X \end{bmatrix} \begin{bmatrix} 1 & X & 0 \\ 1 & 1 & 0 \\ 1 & X & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & X \\ 1 & 1 & 0 \\ X & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ X & 1 & X \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} X & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & X \end{bmatrix} \begin{bmatrix} 0 & X & 1 \\ 0 & 1 & 1 \\ 0 & X & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & X \\ 0 & 1 & 1 \\ X & 1 & 1 \end{bmatrix}$$

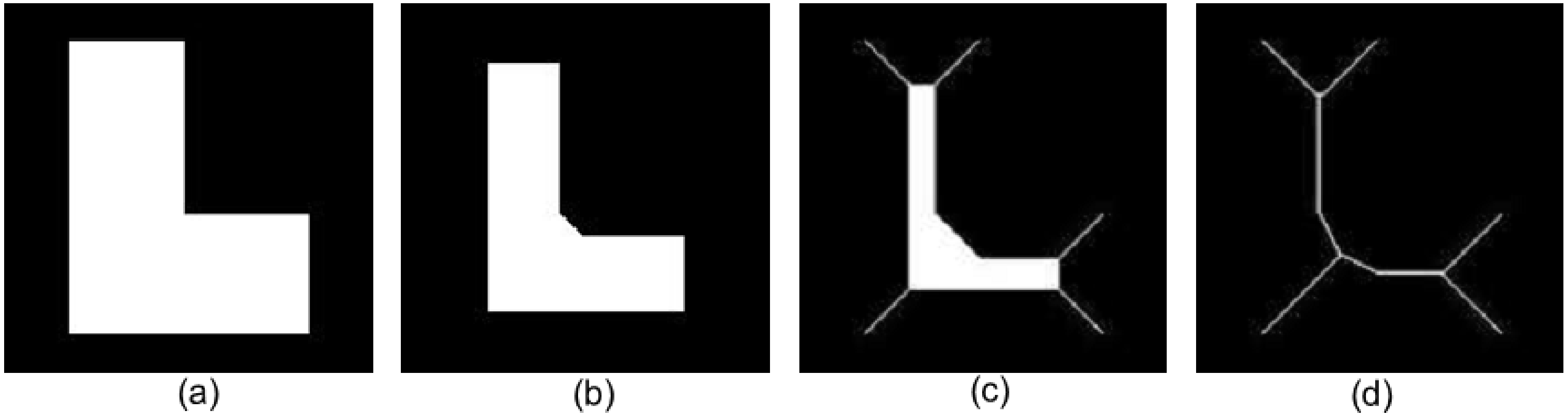
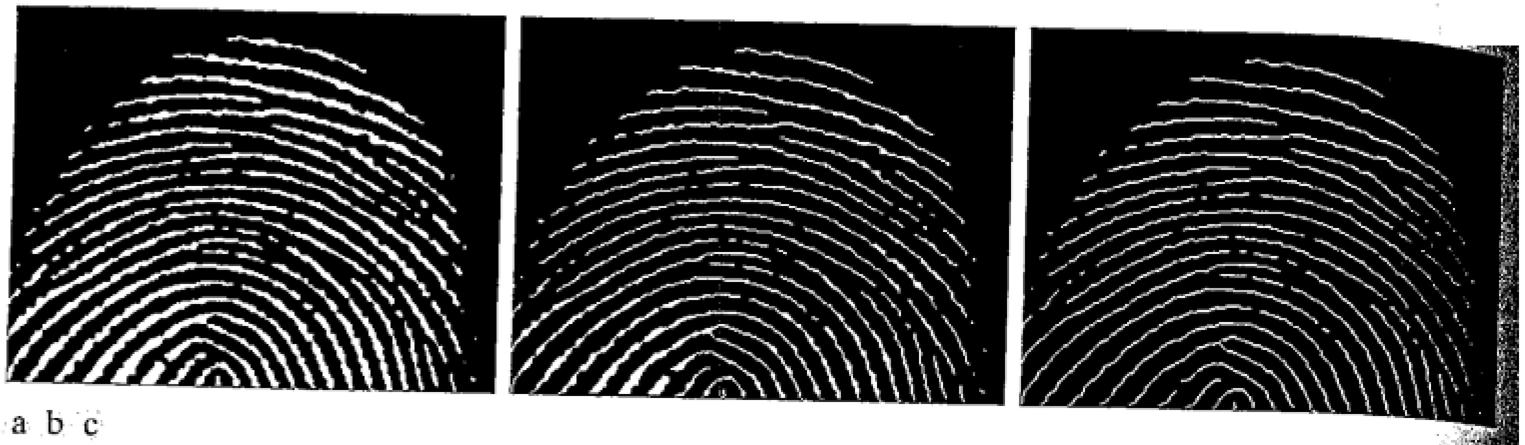
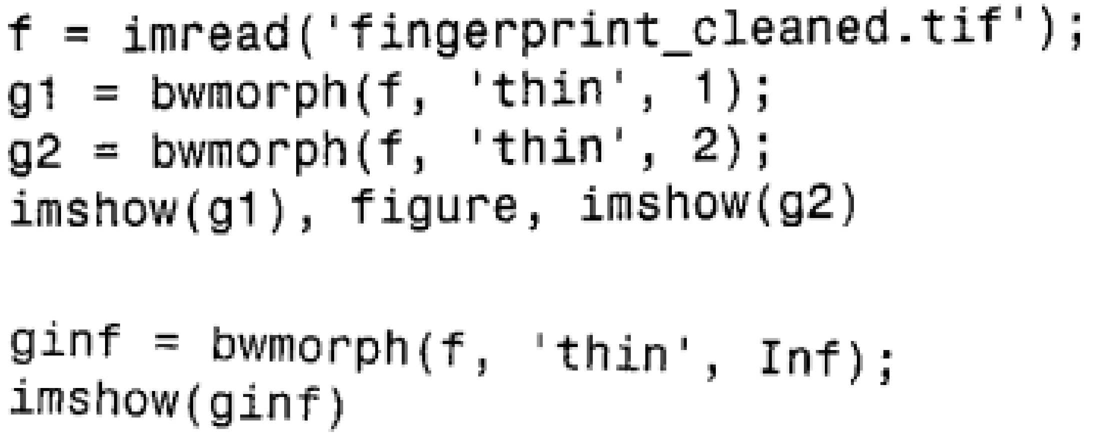


Illustration of thinning for line detection. (a) A binary original image; (b) After 10 iterations of thinning; (c) After 20 iterations of thinning; (d) The thinning process is repeated until no further change occurs, e.g. convergence.



(a) Fingerprint image from Fig. 9.11(c) thinned once. (b) Image thinned twice. (c) Image thinned until stability.



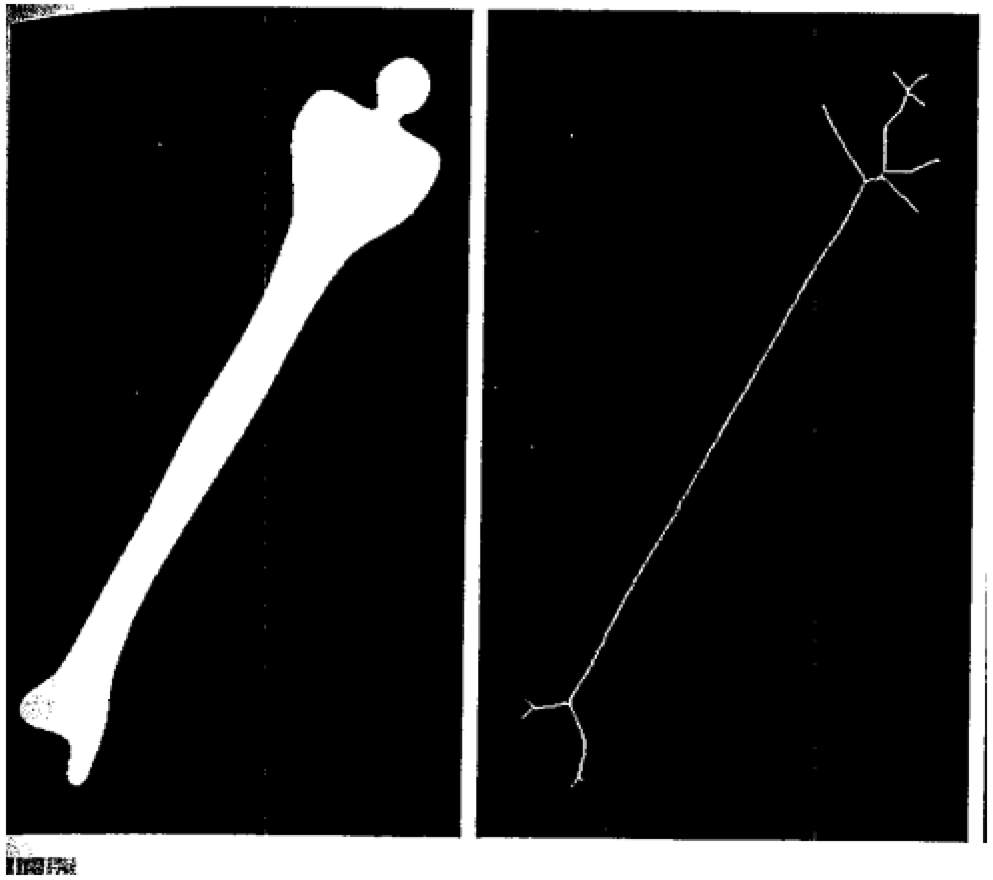
```
f = imread('fingerprint_cleaned.tif');
g1 = bwmorph(f, 'thin', 1);
g2 = bwmorph(f, 'thin', 2);
imshow(g1), figure, imshow(g2)

ginf = bwmorph(f, 'thin', Inf);
imshow(ginf)
```

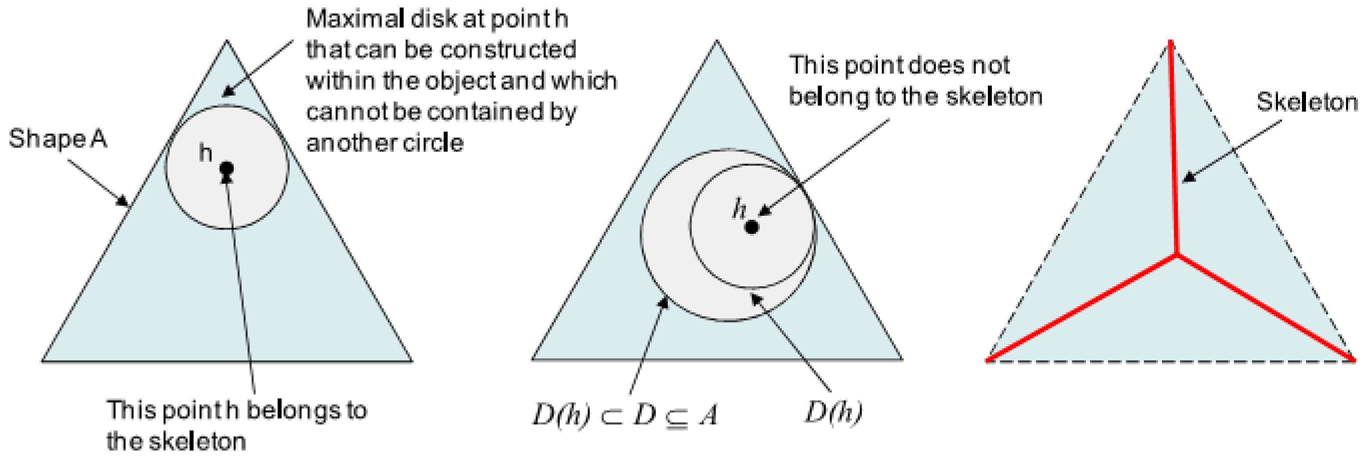
- Thicken làm dầy ảnh có được bằng cách làm thinning ảnh bù

TẠO KHUNG SKELETON

- Làm mỏng ảnh thành các đường dầy một pixel giữ hình dạng của vật



```
>> fs = bwmorph(f, 'skel', Inf);
>> imshow(f), figure, imshow(fs)
```


Maximal disk at point h that can be constructed within the object and which cannot be contained by another circle
Shape A
h
This point h belongs to the skeleton
This point does not belong to the skeleton
h
$D(h) \subset D \subseteq A$
$D(h)$
Skeleton

Illustrating the definition of the skeleton of an object: construction of the Euclidean skeleton for a triangular shape.

To define the skeleton of a shape *A*, for each point $h \in A$, let *D(h)* denote the largest disk centred at *h* such that $D(h) \subseteq A$. Then, the point *h* is a point on the skeleton of *A* if there does not exist a disk *D*, such that $D(h) \subset D \subseteq A$. In this case, *D(h)* is called the maximal disk located at point *h*. If, in addition to the skeleton, the radii of the *maximal disks* located at all points *h* on the skeleton of a shape *A* are known, then *A* can be uniquely reconstructed from this information as the union of all such maximal disks. Therefore, the skeleton, together with the radius information associated with the maximal disks, contains enough information to uniquely reconstruct the original shape. An example of skeletonization is illustrated

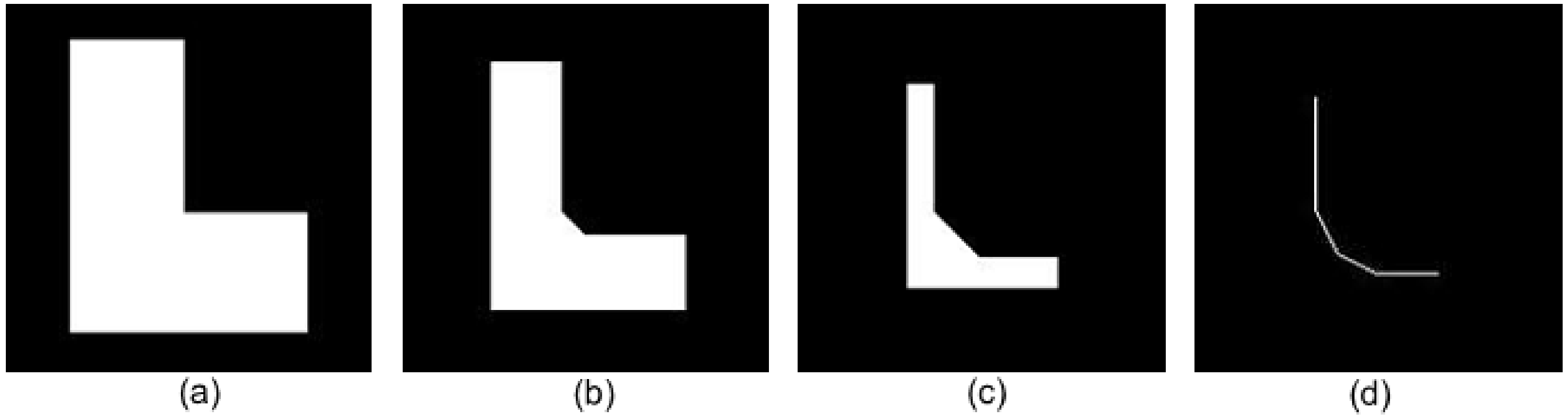
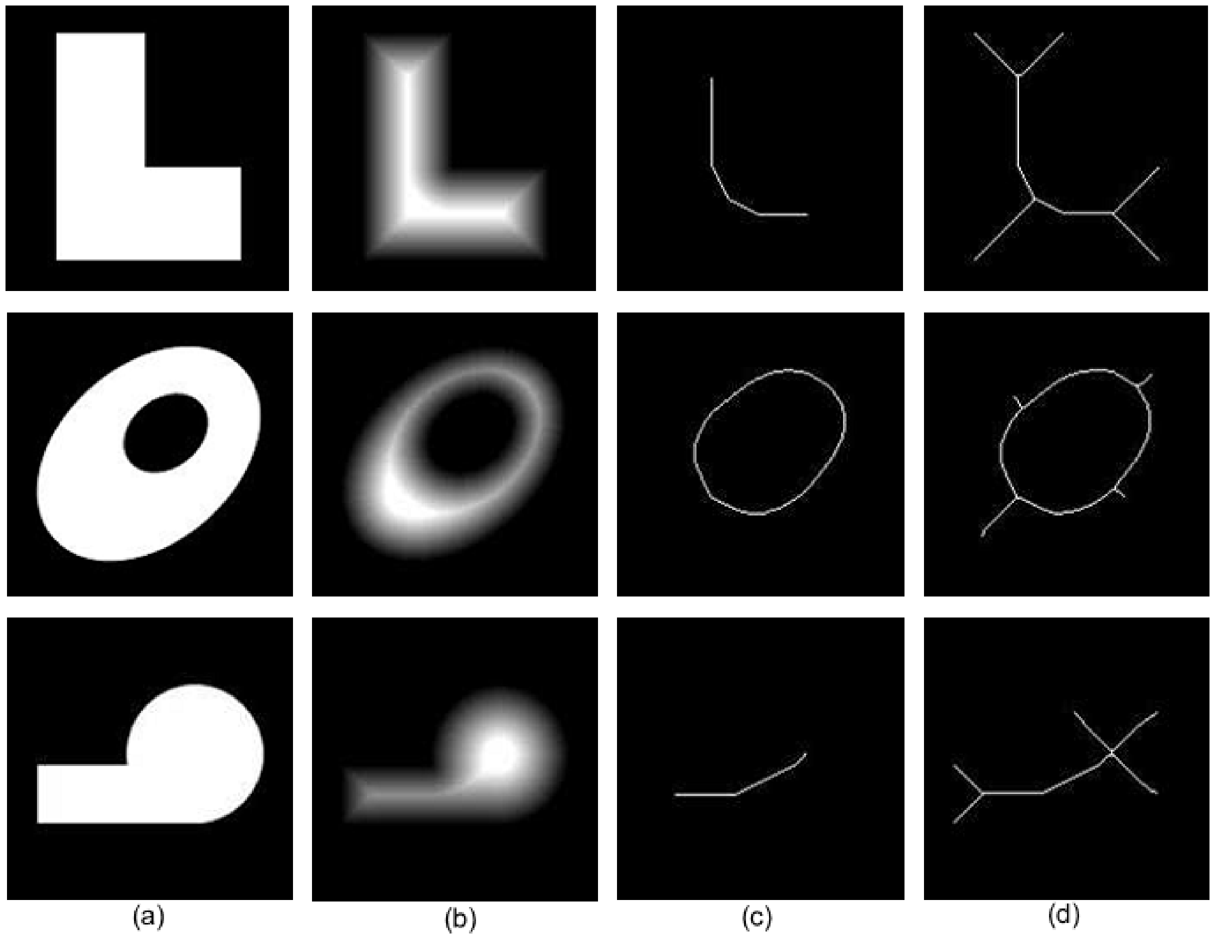
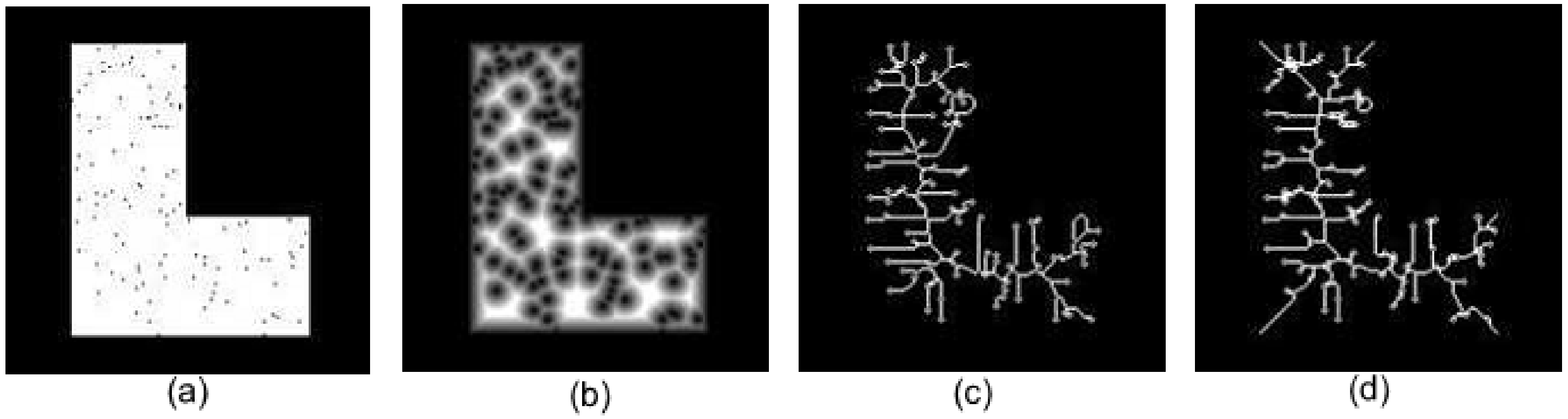


Illustration of skeleton for line detection. (a) A L-shaped original binary image; (b) After 10 iterations of skeletonization; (c) After 20 iterations of skeletonization; (d) The skeletonization process is repeated until no further change occurs, e.g. convergence.



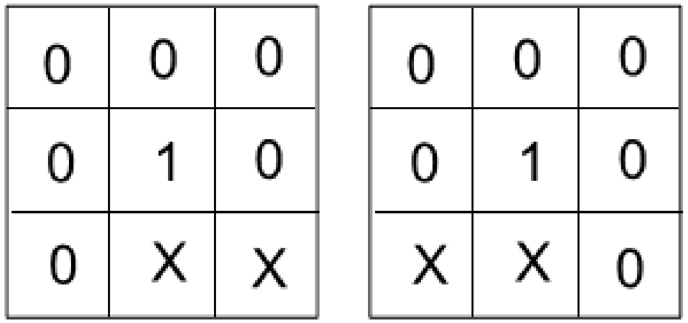
Skeleton vs. thinning. (a) Original binary images with various shapes; (b) Distance transformation of image (a); (c) Skeleton images; (d) Thinning images.



Both Skeleton and thinning are sensitive to noise. (a) Original L-shaped binary images with added pepper noise; (b) Distance transformation of image (a); (c) Skeleton image; (d) Thinning image.

PRUNING (TiẢ)

- Thinning và Skeleton có thể tạo các cựa gà spur ở các điểm nhọn của ảnh
- Xóa các cựa này bằng Pruning dùng các SE



0	0	0
0	1	0
0	X	X

(a)

0	0	0
0	1	0
X	X	0

(b)

Structure elements used for pruning. At each iteration, each element must be used in each of its four 90°.

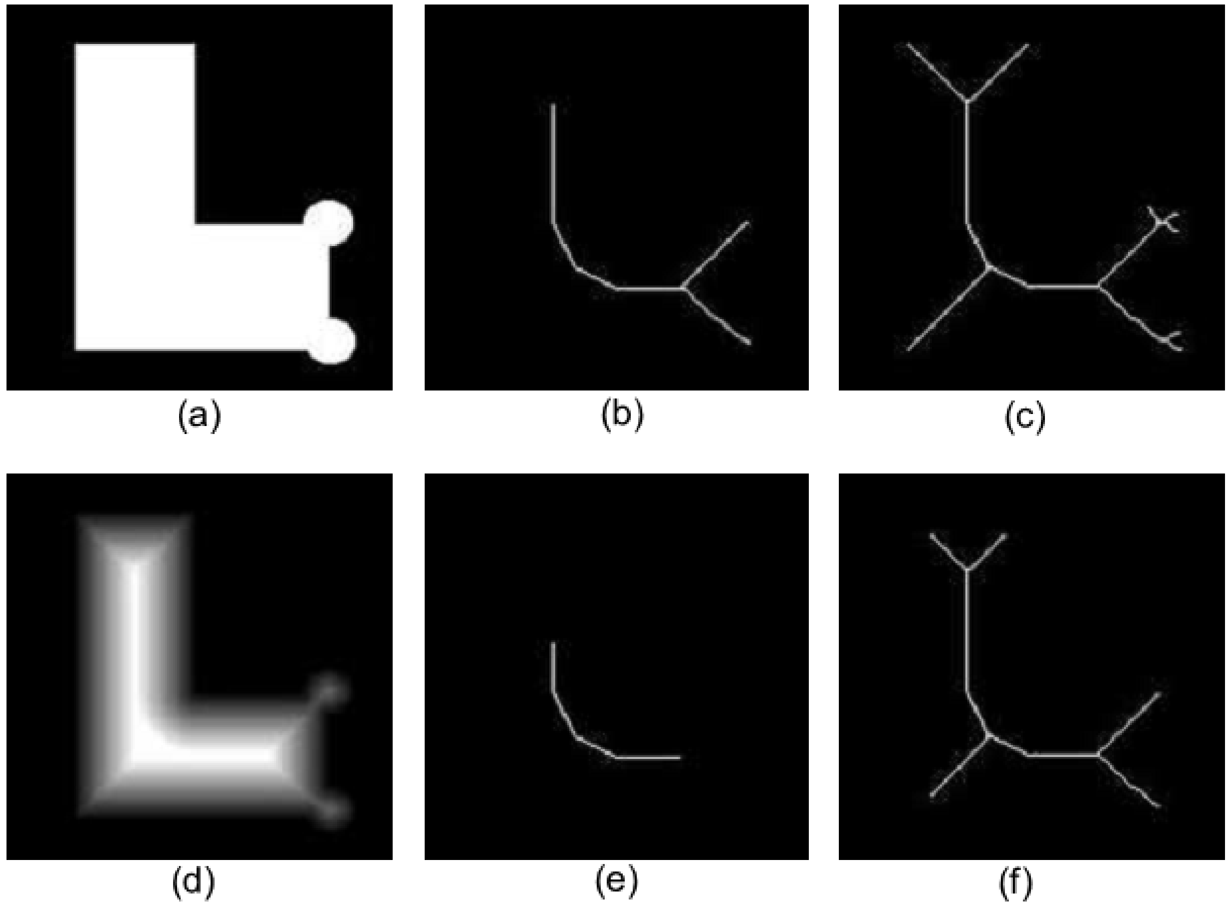
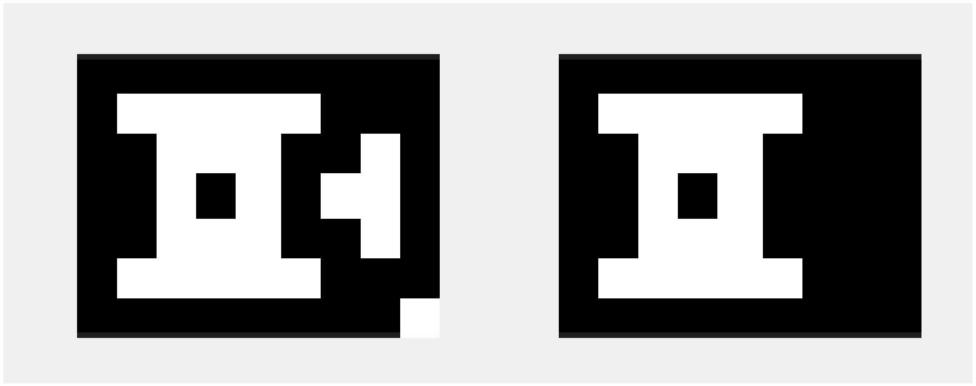


Illustration of pruning. (a) A L-shaped original binary image with some boundary distortion on the bottom right of L; (b) Skeletonization of image (a); (c) Thinning of image (a); (d) Distance transformation of image (a); (e) After 10 iterations of pruning on image (b); (f) After 30 iterations of pruning on image (c).

REMOVE SMALL BLOB

Dùng hàm BW2 = bwareaopen(BW, npixel) để loại bỏ các đốm nhỏ có số pixel<npixel

```
im=[0 0 0 0 0 0 0 0 0;
    0 1 1 1 1 1 0 0 0;
    0 0 1 1 1 0 0 1 0;
    0 0 1 0 1 0 1 1 0;
    0 0 1 1 1 0 0 1 0;
    0 1 1 1 1 1 0 0 0;
    0 0 0 0 0 0 0 0 1];
im1=bwareaopen(im,5);
subplot(1,2,1);imshow(im);
subplot(1,2,2);imshow(im1);
```



Detecting a Cell Using Image Segmentation

This example shows how to detect a cell using edge detection and basic morphology. An object can be easily detected in an image if the object has sufficient contrast from the background.

```
I = imread('cell.tif');
figure, imshow(I), title('original image');
%Detect Entire Cell
[~, threshold] = edge(I, 'sobel');
fudgeFactor = .5;
BWs = edge(I,'sobel', threshold * fudgeFactor);
figure, imshow(BWs), title('binary gradient mask');
%Dilate the Image
se90 = strel('line', 3, 90);
se0 = strel('line', 3, 0);
```

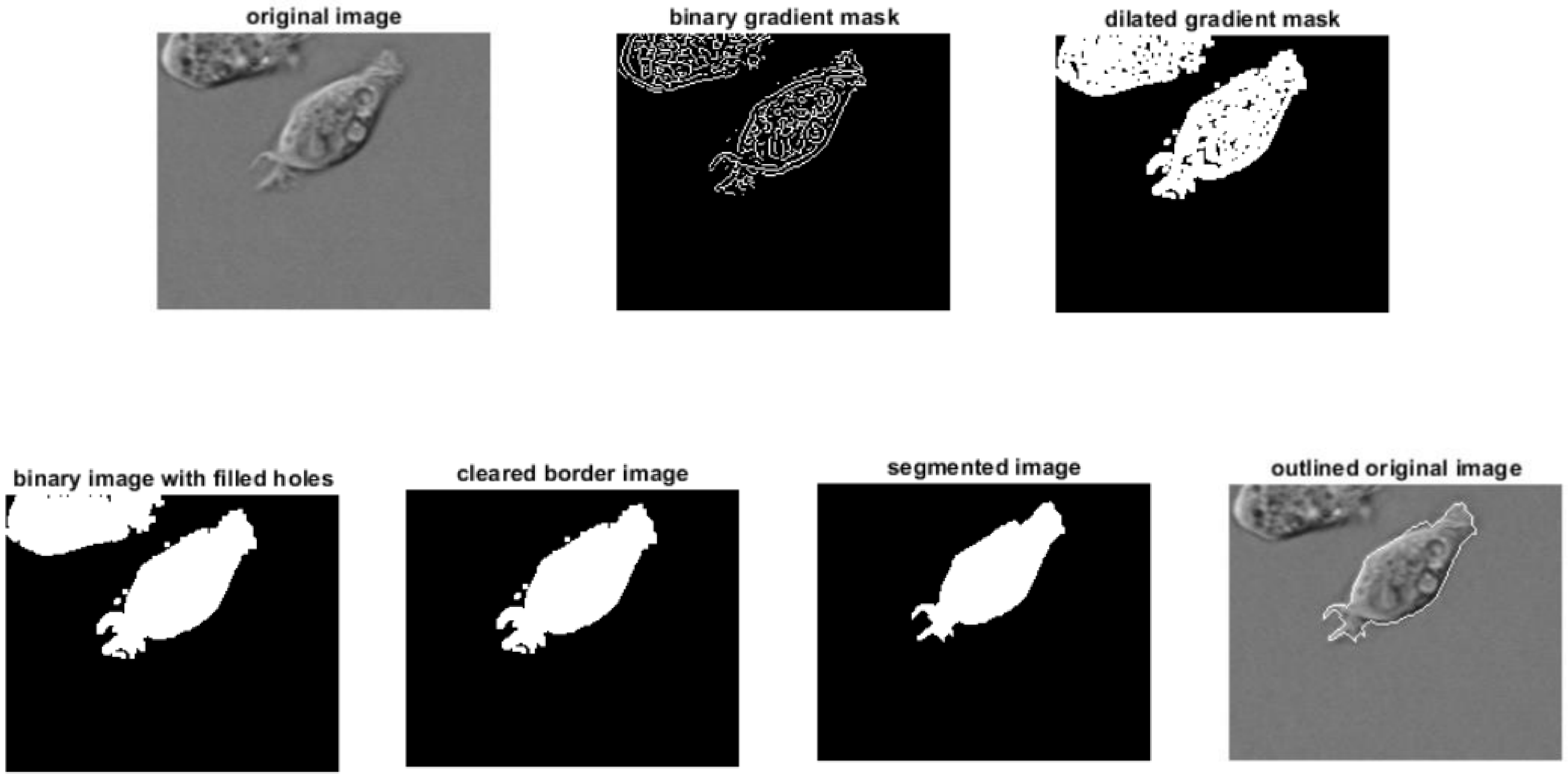
Detecting a Cell Using Image Segmentation

```
BWsdil = imdilate(BWs, [se90 se0]);
figure, imshow(BWsdil), title('dilated gradient mask');
%Fill Interior Gaps
BWdfill = imfill(BWsdil, 'holes');
figure, imshow(BWdfill);
title('binary image with filled holes');
%Remove Connected Objects on Border
BWnobord = imclearborder(BWdfill, 4);
figure, imshow(BWnobord), title('cleared border image');
```

Detecting a Cell Using Image Segmentation

```
% Smoothen the Object
seD = strel('diamond',1);
BWfinal = imerode(BWnobord,seD);
BWfinal = imerode(BWfinal,seD);
figure, imshow(BWfinal), title('segmented image');
%Draw Outline
BWoutline = bwperim(BWfinal);
Segout = I;
Segout(BWoutline) = 255;
figure, imshow(Segout), title('outlined original image');
```

Detecting a Cell Using Image Segmentation



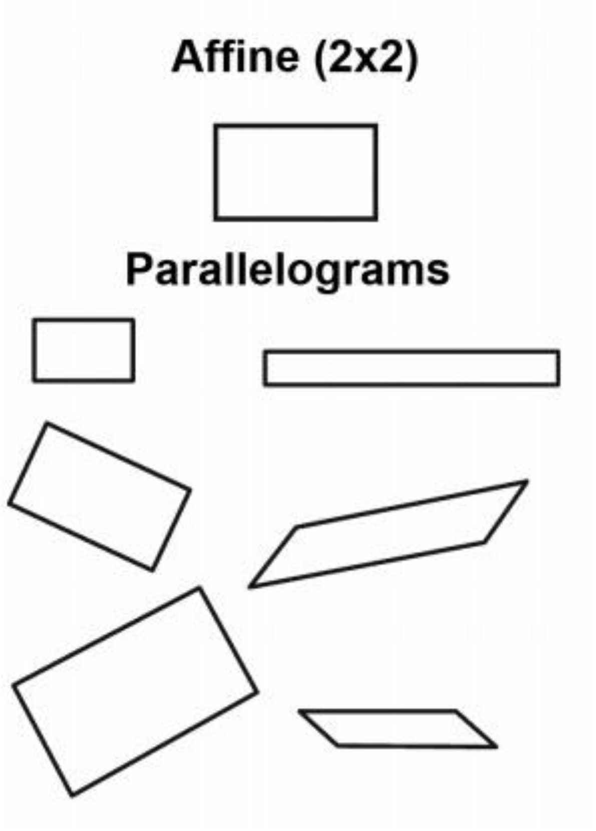
BiẾN ĐỔI HÌNH HỌC

- Thực hiện các phép biến đổi như stretch, shrink, warp, rotate
- Biến đổi affine: cho điểm X(x;y) ma trận A 2*2 và vectơ b 2*1, Điểm mới là X’(x’;y’)

  X’=AX+b, M=[A b]

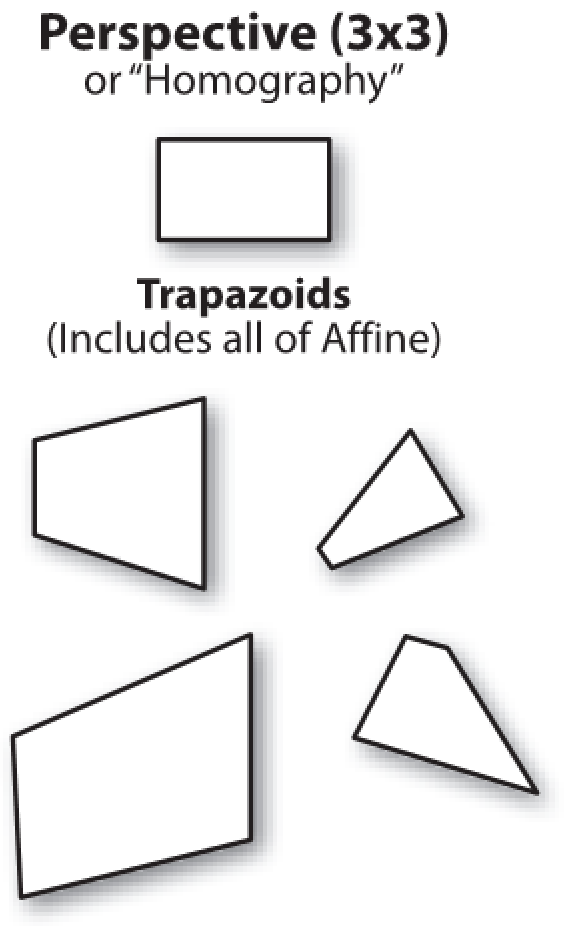
  X’=M*[x y 1]’

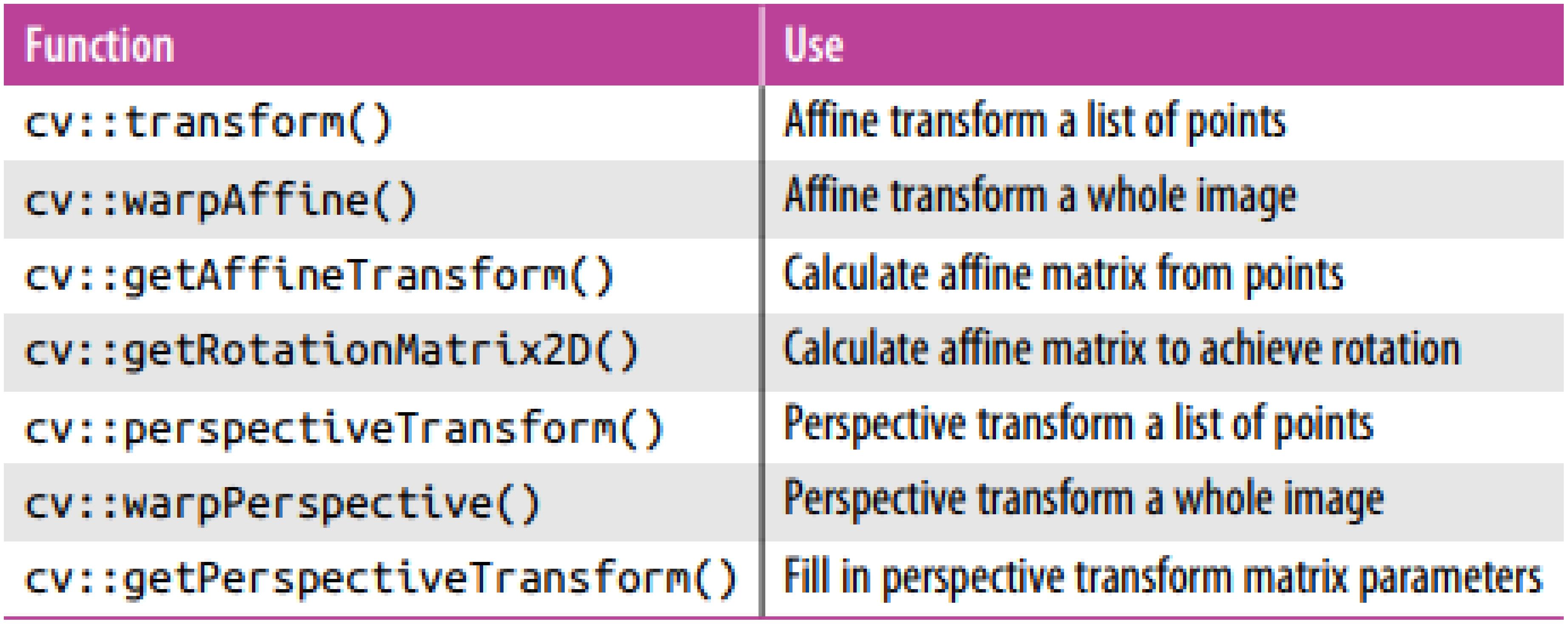
  M là ma trận 2*3
- Biến đổi affine nén dãn, quay ảnh, có thể biến đổi hình chữ nhật thành hình bình hành, hình bình hành thành hình bình hành.



BiẾN ĐỔI HÌNH HỌC

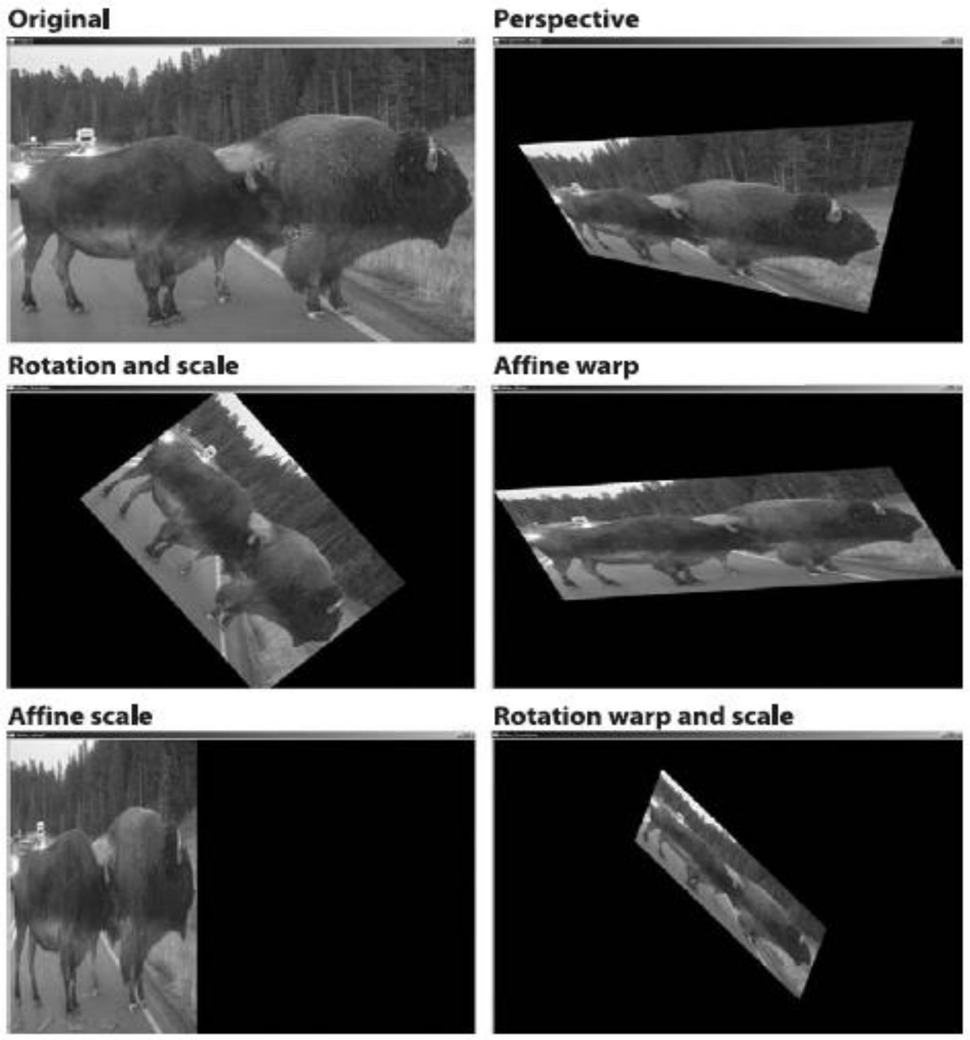
- Biến đổi phối cảnh perspective ta thay vectơ tọa độ X (x,y) bởi vectơ X'(x,y,1), M là ma trận 3*3 , vectơ biến đổi phối cảnh là MX'. Hình chữ nhật sau biến đổi phối cảnh có thể biến thành hình thang





Function	Use
`cv::transform()`	Affine transform a list of points
`cv::warpAffine()`	Affine transform a whole image
`cv::getAffineTransform()`	Calculate affine matrix from points
`cv::getRotationMatrix2D()`	Calculate affine matrix to achieve rotation
`cv::perspectiveTransform()`	Perspective transform a list of points
`cv::warpPerspective()`	Perspective transform a whole image
`cv::getPerspectiveTransform()`	Fill in perspective transform matrix parameters

BiẾN ĐỔI HÌNH HỌC



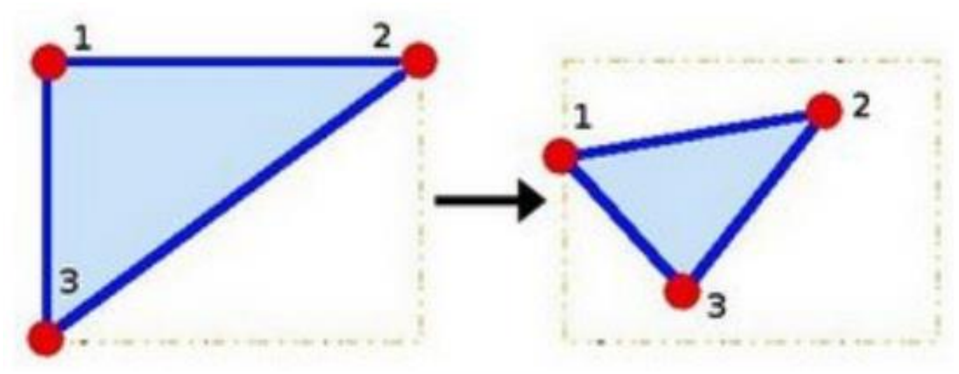
BiẾN ĐỔI HÌNH HỌC

```
void cv::warpAffine(
  cv::InputArray    src,                                 // Input image
  cv::OutputArray   dst,                                 // Result image
  cv::InputArray    M,                                   // 2-by-3 transform mtx
  cv::Size          dsize,                               // Destination image size
  int               flags       = cv::INTER_LINEAR,      // Interpolation, inverse
  int               borderMode  = cv::BORDER_CONSTANT,   // Pixel extrapolation
  const cv::Scalar& borderValue = cv::Scalar()           // For constant borders
);
```

$$dst(x, y) = src(M_{00}x + M_{01}y + M_{02}, M_{10}x + M_{11}y + M_{12})$$

Ma trận M được tính từ 3 điểm ảnh gốc và 3 điểm ảnh đích

```
cv::Mat cv::getAffineTransform(         // Return 2-by-3 matrix
  const cv::Point2f* src,               // Coordinates *three* of vertices
  const cv::Point2f* dst                // Target coords, three vertices
);
```



```
cv::Mat cv::getRotationMatrix2D(                    // Return 2-by-3 matrix
  cv::Point2f  center                               // Center of rotation
  double       angle,                               // Angle of rotation
  double       scale                                // Rescale after rotation
);
```

AFFINE TRANSFORM

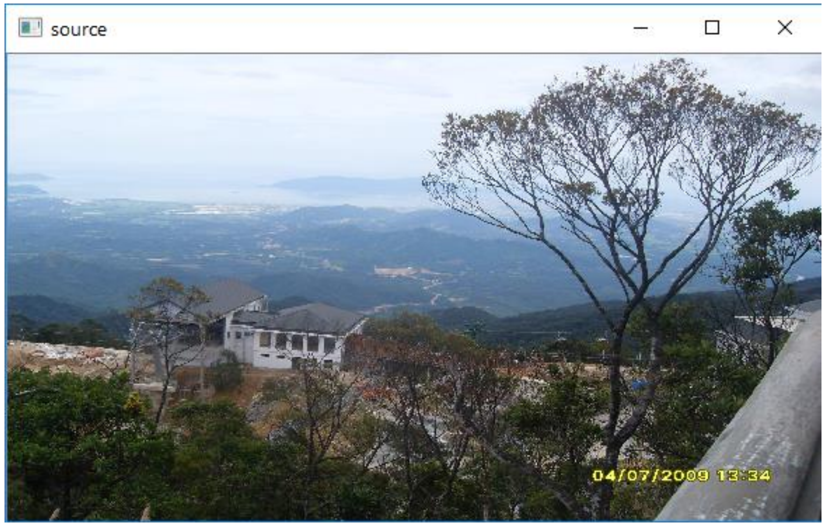
```
int main( int argc, char** argv )
 {Point2f srcTri[3];
   Point2f dstTri[3];
   Mat rot_mat( 2, 3, CV_32FC1 );
   Mat warp_mat( 2, 3, CV_32FC1 );
   Mat src, warp_dst, warp_rotate_dst;
src = imread( argv[1], 1 );
namedWindow(“source”, 0);
imshow( “source”, src );
   // Set the dst image the same type and size as src
   warp_dst = Mat::zeros( src.rows, src.cols, src.type() );
   // Set your 3 points to calculate the  Affine Transform
```

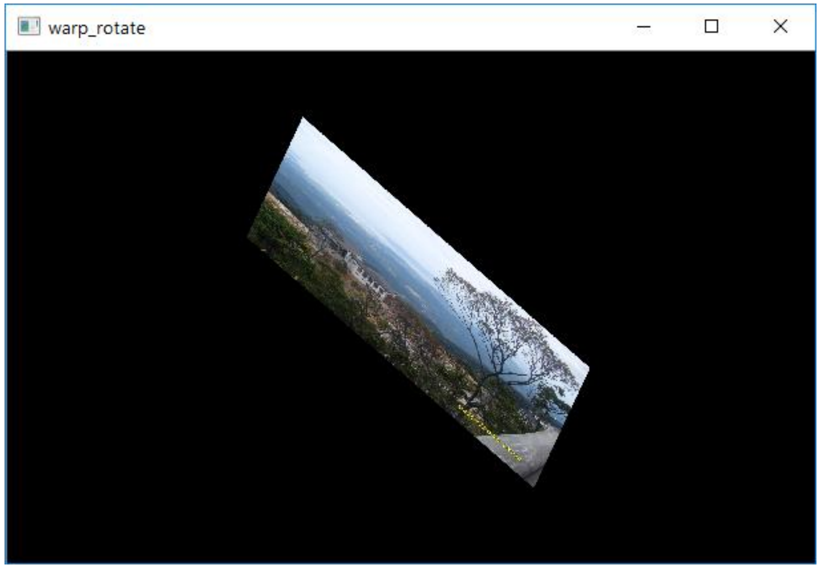
AFFINE TRANSFORM

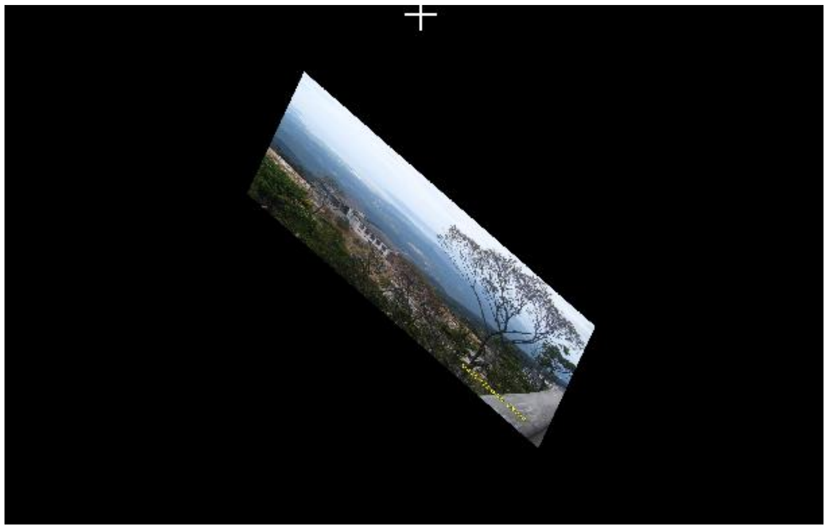
```
 srcTri[0] = Point2f( 0,0 ); //Top left
 srcTri[1] = Point2f( src.cols - 1, 0 );//Top right
  srcTri[2] = Point2f( 0, src.rows - 1 );
  dstTri[0] = Point2f( src.cols*0.0, src.rows*0.33 );
  dstTri[1] = Point2f( src.cols*0.85, src.rows*0.25 );
  dstTri[2] = Point2f( src.cols*0.15, src.rows*0.7 );
  /// Get the Affine Transform
  warp_mat = getAffineTransform( srcTri, dstTri );
  /// Apply the Affine Transform just found to the src image
  warpAffine( src, warp_dst, warp_mat, warp_dst.size() );
/// Compute a rotation matrix with respect to the center of the image
  Point center = Point( warp_dst.cols/2, warp_dst.rows/2 );
  double angle = -50.0;
  double scale = 0.6;
```

AFFINE TRANSFORM

```
// Get the rotation matrix with the specifications above
   rot_mat = getRotationMatrix2D( center, angle, scale );
   //Rotate the warped image
warpAffine( warp_dst, warp_rotate_dst, rot_mat, warp_dst.size() );
namedWindow(“warp”, 0);
imshow( “warp”, warp_dst );
namedWindow(“warp_rotate”, 0);
imshow( “warp_rotate”, warp_rotate_dst );
waitKey(0);
return 0;
  }
```


source
04/07/2009 13:34


warp_rotate



PERSPECTIVE TRANSFORM

```
void cv::warpPerspective(
  cv::InputArray    src,                                  // Input image
  cv::OutputArray   dst,                                  // Result image
  cv::InputArray    M,                                    // 3-by-3 transform mtx
  cv::Size          dsize,                                // Destination image size
  int               flags       = cv::INTER_LINEAR,       // Interpolation, inverse
  int               borderMode  = cv::BORDER_CONSTANT,    // Extrapolation method
  const cv::Scalar& borderValue = cv::Scalar()            // For constant borders
);
```

$$dst(x, y) = src\left(\frac{M_{00}x + M_{01}y + M_{02}}{M_{20}x + M_{21}y + M_{22}}, \frac{M_{10}x + M_{11}y + M_{12}}{M_{20}x + M_{21}y + M_{22}}\right)$$

```
cv::Mat cv::getPerspectiveTransform(        // Return 3-by-3 matrix
  const cv::Point2f* src,                   // Coordinates of *four* vertices
  const cv::Point2f* dst                    // Target coords, four vertices
);
```

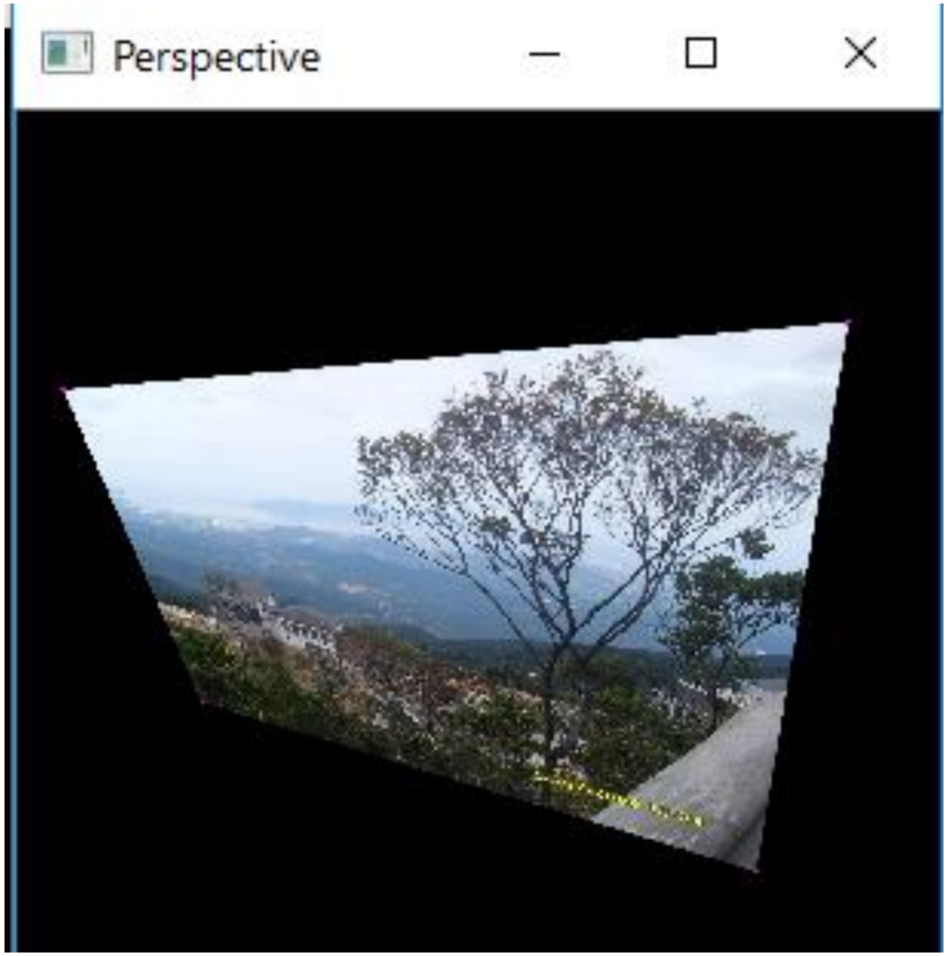
PERSPECTIVE TRANSFORM

```
int main(int argc, char** argv) {
Point2f srcQuad[] = {
Point2f(0, 0), // src Top left
Point2f(src.cols-1, 0), // src Top right
Point2f(src.cols-1, src.rows-1), // src Bottom right
Point2f(0, src.rows-1) // src Bottom left
};
Point2f dstQuad[] = {
Point2f(src.cols*0.05f, src.rows*0.33f),
Point2f(src.cols*0.9f, src.rows*0.25f),
Point2f(src.cols*0.8f, src.rows*0.9f),
Point2f(src.cols*0.2f, src.rows*0.7f)
};
```

PERSPECTIVE TRANSFORM

```
Mat warp_mat = cv::getPerspectiveTransform(srcQuad, dstQuad);
Mat dst;
warpPerspective(src, dst, warp_mat, src.size(), cv::INTER_LINEAR);
for( int i = 0; i < 4; i++ )
circle(dst, dstQuad[i], 5, cv::Scalar(255, 0, 255));
namedWindow(("Perspective", 0);
imshow("Perspective", dst);
cv::waitKey();
return 0;
}
```

PERSPECTIVE TRANSFORM


Perspective