



AGH University of Science and Technology

Department of Telecommunication

Morphological Operations on Images

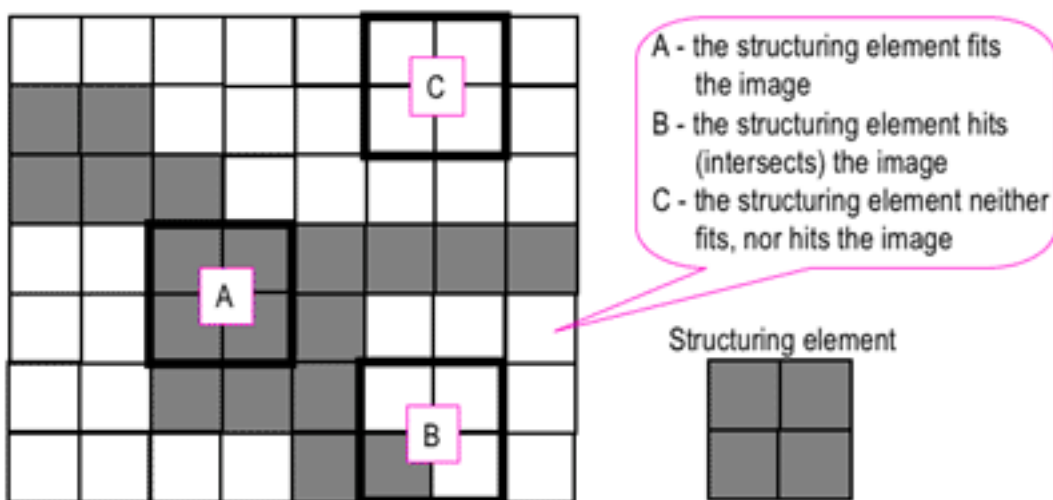
Introduction to image processing using Matlab

Theoretical fundamentals

1. Morphological Operations

Morphology is a broad set of image processing operations that process images based on shapes. Morphological operations apply a structuring element to an input image, creating an output image of the same size. In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. By choosing the size and shape of the neighborhood, you can construct a morphological operation that is sensitive to specific shapes in the input image.

The most basic morphological operations are dilation and erosion. Dilation adds pixels to the boundaries of objects in an image, while erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the image. In the morphological dilation and erosion operations, the state of any given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image. The rule used to process the pixels defines the operation as a dilation or an erosion.



Probing of an image with a structuring element
(white -zero, grey pixels have non-zero values).

1.1 Erosion

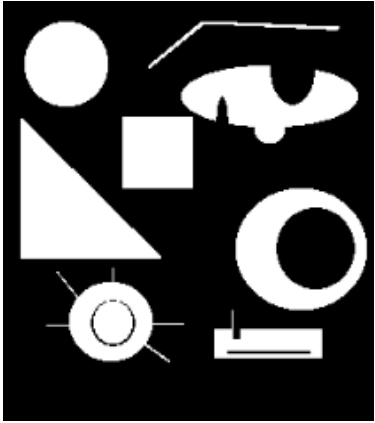
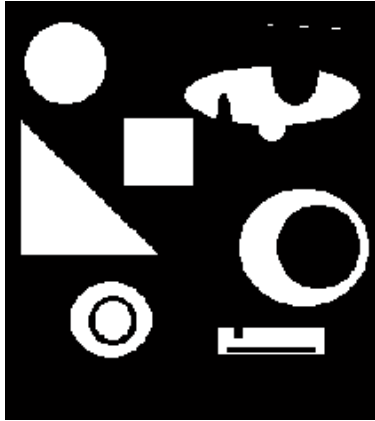
Let E be a Euclidean space or an integer grid, and A a binary image in E .

$$A \ominus B = \{z \in E \mid B_z \subseteq A\}$$

where B_z is the translation of B by the vector z , i.e. $B_z = \{b + z \mid b \in B\}$

Erosion is one of the two basic operators in the area of mathematical morphology, the other being dilation. It is typically applied to binary images, but there are versions that work on grayscale images. The basic effect of the operator on a binary image is to erode away the boundaries of regions of foreground pixels (*i.e.* white pixels, typically). Thus areas of foreground pixels shrink in size, and holes within those areas become larger.

Example:

Input	Structuring element	Output									
	<table border="1" data-bbox="651 1115 924 1332"><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1	1	
1	1	1									
1	1	1									
1	1	1									

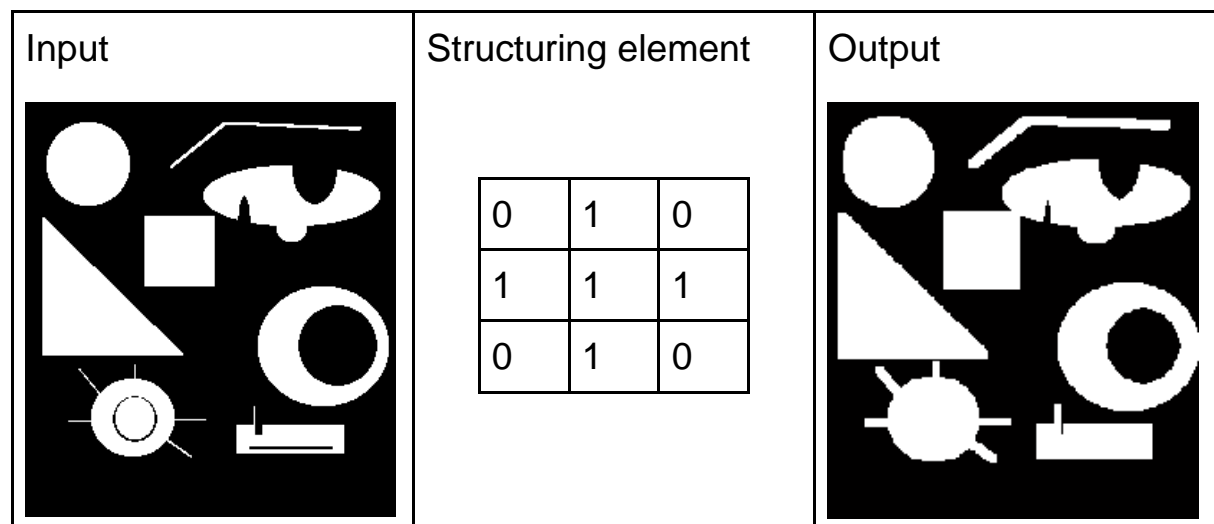
1.2 Dilation

The dilation of A by the structuring element B is defined by:

$$A \oplus B = \bigcup_{b \in B} A_b$$

Dilation is one of the two basic operators in the area of mathematical morphology . It is typically applied to binary images, but there are versions that work on grayscale images. The basic effect of the operator on a binary image is to gradually enlarge the boundaries of regions of foreground pixels. Thus areas of foreground pixels grow in size while holes within those regions become smaller.

Example:



2. List of functions used in the exercise and their descriptions.

<code>imread(filename)</code>	reads the image from the file specified by filename, inferring the format of the file from its contents.
<code>imshow(X)</code>	displays image X in a Handle Graphics® figure, where I is a grayscale, RGB (truecolor), or binary image.
<code>im2bw(X)</code>	converts the grayscale image X to a binary image. The output image replaces all pixels in the input image with luminance greater than level with the value 1 (white) and replaces all other pixels with the value 0 (black).
<code>getnhood(K)</code>	returns the neighborhood(array) associated with the structuring element K(STREL object).
<code>size()</code>	returns the size.
<code>floor(X)</code>	rounds each element of X to the nearest integer less than or equal to that element.
<code>strel(shape, parameters)</code>	creates a structuring element, K, of the type specified by shape. Depending on shape, strel can take additional parameters.
<code>padarray(A, padsize)</code>	pads array A with 0's. padsize is a vector of nonnegative integers that specifies both the amount of padding to add and the dimension along which to add it. The value of an element in the vector specifies the amount of padding to add. The order of the element in the vector specifies the dimension along which to add the padding.
<code>false(n)</code>	is an n-by-n array of logical zeros.
<code>max(A)</code>	returns the largest elements of A.
<code>min(A)</code>	returns the smallest elements of A.
<code>subplot(m, n, p)</code>	divides the current figure into an m-by-n grid and creates an axes for a subplot in the position specified by p. MATLAB® numbers its subplots by row, such that the first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If the axes already exists, then the command <code>subplot(m,n,p)</code> makes the subplot in position p the current axes.

3. Example of erosion/dilation in Matlab.

Here you can see how inbuilt matlab code for dilate and erosion works.

```
A=imread('file_name.jpg');
subplot(1,3,1), imshow(A);

s=strel('disk',2);
B=imerode(A,s);
x=strel('square',2);
C=imdilate(A,x);

subplot(1,3,2), imshow(B);
subplot(1,3,3), imshow(C);
```

Morphological Operations on Images

(Basic morphological operations on images: erosion and dilate)

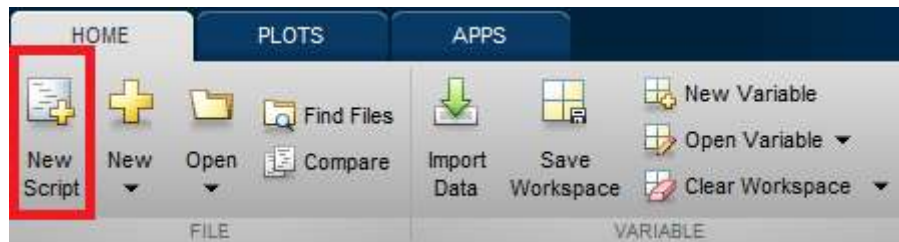
Instruction

Explanations are in Theory.pdf.

- Create a folder on your desktop and name it "Morph".
- Copy all the images from CD into this folder.
- Start MATLAB and set your workspace to mentioned before folder.



- Create new script and follow instructions



1. Image erosion

- In the first line we want to create our application using:

```
function [] = program_name()
```

- Next thing we want to do, is to load our image into matrix. To do so you need to create a variable 'A' and load into it image (we will be using 'img8.png') using function `imread('img8.png')`.
- We can see if the image is loaded properly using `imshow()` function which displays our image. So we could see all the images on the same chart let's use this part `subplot(1,3,1)`. We also want to put a title to our picture. In the next line write this part of code `title('here_description')`. You can create a description under every image you are going to show.

Our application will be operating on the black and white images using their matrix binary form. 1's stands for white color and 0's stands for black color.

- To convert our image into matrix containing 1's and 0's we need to use function `im2bw()` on the same variable we created before
- To see it on the same figure as the image before let's paste this part of the code `subplot(1,3,2)`,
- Next step is creating our structuring element. To do it use this line:

```
B=getnhood(strel('disk', 2));
```

What it does is returns the neighborhood associated with the structuring element(`getnhood`) which is array filled with 1's and 0's in the form of flat disk with radius value of 2(created with `strel` command). If you change the value of radius you can see that it affects the output image.

- Next are those two lines:

```
m=floor(size(B,1)/2);  
n=floor(size(B,2)/2);
```

Basically what it does is returns a number of lines and columns in our structuring element divided by 2 and rounded to integer value.

- Next we need to create a space for our dilated image. Create variable 'D' and use function `false()` to create matrix of 0's with the size of our original image
- Next is a part of code that we need to use to move around our image

```
for i=1:size(A,1)-(2*m)  
    for j=1:size(A,2)-(2*n)  
  
        X=A(i:i+(2*m),j:j+(2*n));  
        D(i,j)=min(min(X-B));  
  
    end  
end
```

We have two loops because we are moving around our image using lines and columns (i - stands for lines and j - stands for columns). Both lines

and columns are shorter because of the size of our structuring element. We don't want this element to go outside of the image.

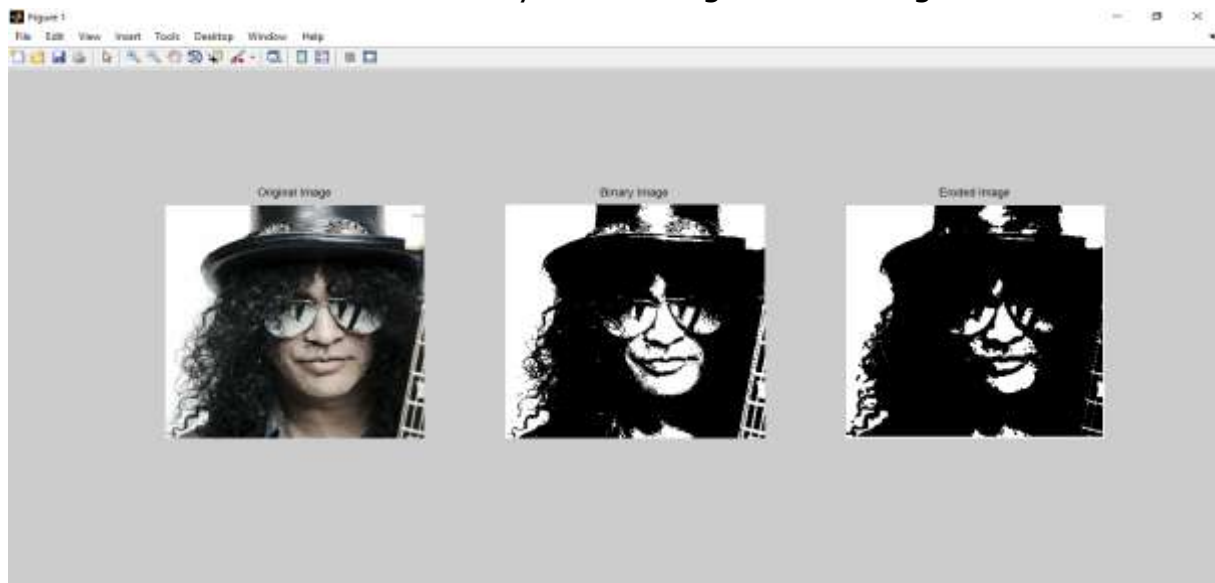
Variable X contain a part of our image that will be at the moment compared with our structuring element. So It has to be the same size as structuring element.

Variable D with every loop fill the position (i,j) of our image with a result of operation. Gray-scale erosion is equivalent to a local-minimum operator.

- Last step is to again show our image, but this time we need to flip the colors

```
subplot(1,3,2)
imshow(~D);
```

As a result of the code you should get something like this



2. Image dilate

There is not much of a difference between application for erosion and dilatation. All you need to do is to go through those few simple steps.

- First we need to change our structuring element to (for example):

```
B=[1 1 1 1 1 1 1];
```


- Next is our main part. It should look like this.

```
for i=1:size(A,1)
    for j=1:size(A,2)-6
        D(i,j)=sum(B&A(i,j:j+6));
    end
end
```

We have two loops because we are moving around our image using lines and columns (i - stands for lines and j - stands for columns). We start by moving our structuring element from position 1,1(then 1,2 | 1,3 | 1,4 | ...) on our matrix.

Variable 'D' contain a result of logical AND operation of our structuring element and image. If all the values are zero then our D matrix is updated on position (i,j) with 0. If the of operation is different, then our D matrix is updated on position (i,j) with 1.

- Last but not least is showing our image.

```
subplot(1,3,3),
imshow(D);
```

As a result of the code you should get something like this

