7. Morphological operations on binary images

7.1. Introduction

Morphological operations are affecting the form, structure or shape of an object. Applied on binary images (black & white images – Images with only 2 colors: *black* and *white*). They are used in pre or post processing (filtering, thinning, and pruning) or for getting a representation or description of the shape of objects/regions (boundaries, skeletons convex hulls).

7.2. Theoretical considerations

The two principal morphological operations are *dilation* and *erosion* [1]. Dilation allows objects to expand, thus potentially filling in small holes and connecting disjoint objects. Erosion shrinks objects by etching away (eroding) their boundaries. These operations can be customized for an application by the proper selection of the structuring element, which determines exactly how the objects will be dilated or eroded.

Notations:

black pixel: in grayscale values for a 8 bits/pixel indexed image its value will be 0
white pixel: in grayscale values for a 8 bits/pixel indexed image its value will be 255

7.2.1. The dilation

The *dilation* process is performed by laying the structuring element **B** on the image **A** and sliding it across the image in a manner similar to convolution (will be presented in a next laboratory). The difference is in the operation performed. It is best described in a sequence of steps:

1. If the origin of the structuring element coincides with a 'white' pixel in the image, there is no change; move to the next pixel.
2. If the origin of the structuring element coincides with a 'black' in the image, make black all pixels from the image covered by the structuring element.

Notation:

$A \oplus B$

The structuring element can have any shape. Typical shapes are presented bellow:

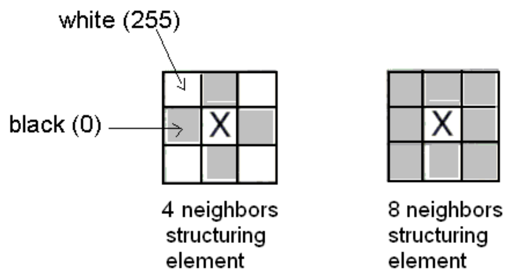

white (255)
black (0)
X
X
4 neighbors structuring element
8 neighbors structuring element

Fig. 7.1 Typical shapes of the structuring elements (B)

An example is shown in Fig. 7.2. Note that with a dilation operation, all the 'black' pixels in the original image will be retained, any boundaries will be expanded, and small holes will be filled.

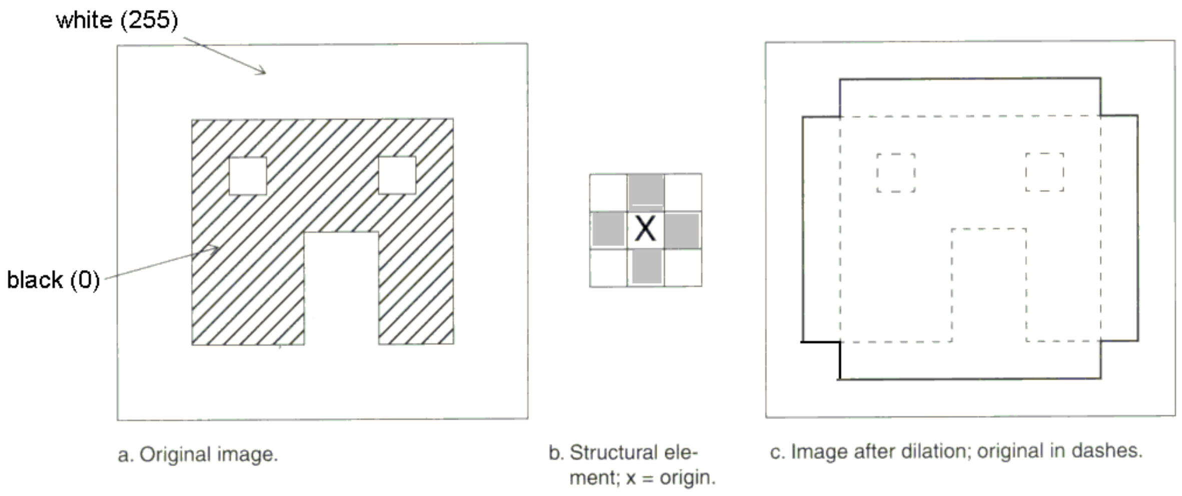

white (255)
black (0)
X
a. Original image.
b. Structural element; x = origin.
c. Image after dilation; original in dashes.

Fig. 7.2 Illustration of the dilatation process

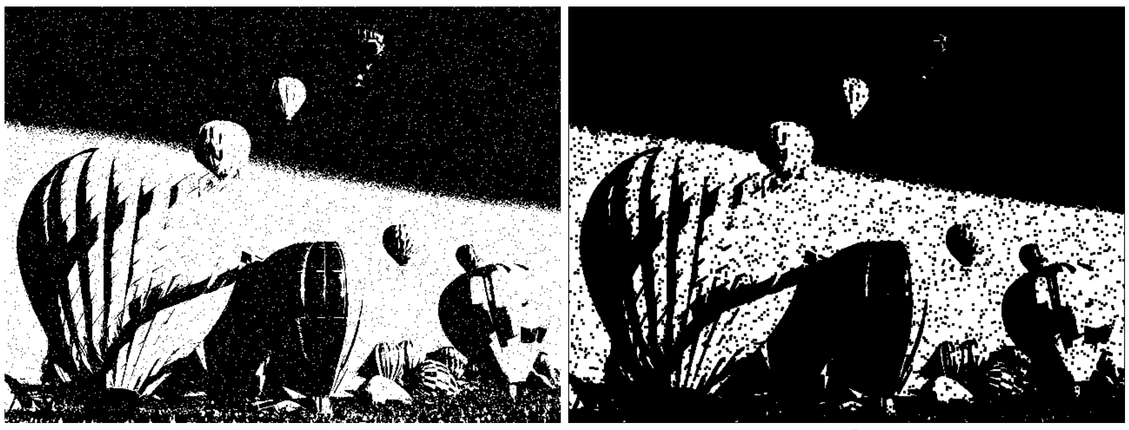


Fig. 7.3 Example of the dilation: a. Original image **A**; b. The result image: **A** $\oplus$ **B**.

7.2.2. The erosion

The *erosion* process is similar to dilation, but we turn pixels to 'white', not 'black'. As before, slide the structuring element across the image and then follow these steps:

1. If the origin of the structuring element coincides with a 'white' pixel in the image, there is no change; move to the next pixel.

2. If the origin of the structuring element coincides with a 'black' pixel in the image, and any of the 'black' pixels in the structuring element extend beyond the object in the image (with ‘black' pixels) in the image, then change the 'black' pixel in the image to a 'white'.

Notation:

$$A \Theta B$$

In Fig. 7.4, the only remaining pixels are those that coincide to the origin of the structuring element where the entire structuring element was contained in the existing object. Because the structuring element is 3 pixels wide, the 2-pixel-wide right leg of the image object was eroded away, but the 3-pixel-wide left leg retained some of its center pixels.

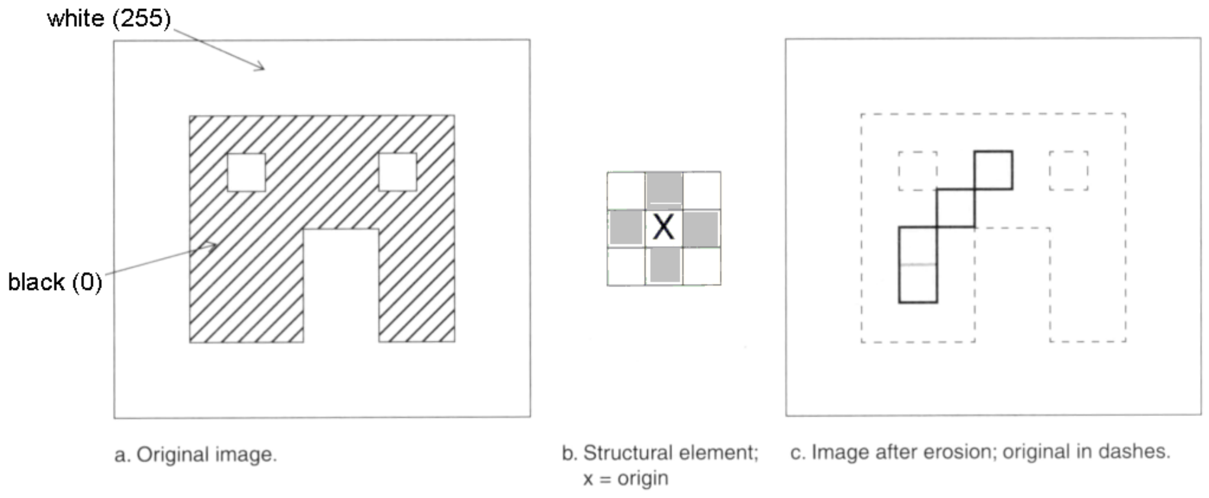

white (255)
black (0)
X
a. Original image.
b. Structural element; x = origin
c. Image after erosion; original in dashes.

Fig. 7.4 Illustration of the erosion process

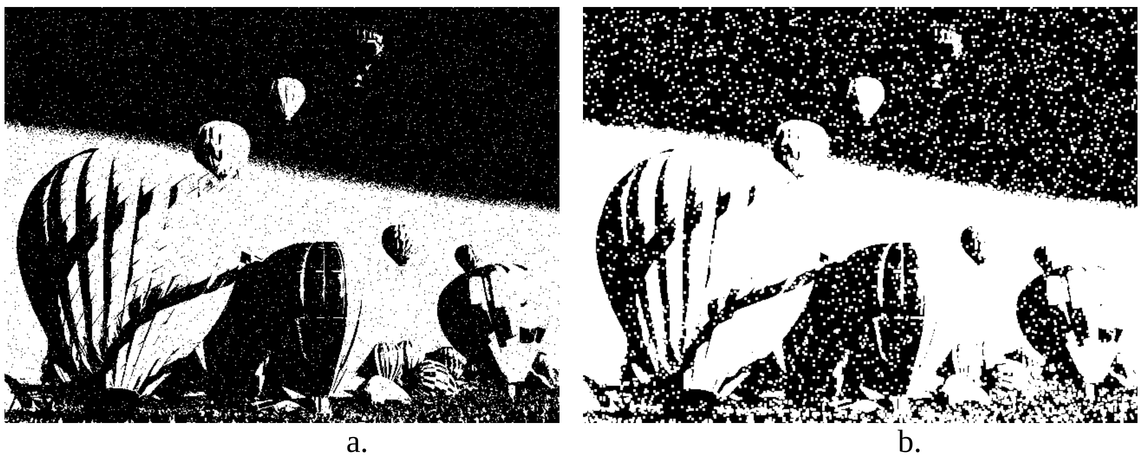


Fig. 7.5 Example of the erosion: a. Original image **A**; b. The result image: **A Θ B**.

7.2.3. Opening and closing

These two basic operations, dilation and erosion, can be combined into more complex sequences. The most useful of these for morphological filtering are called opening and closing [1]. *Opening* consists of an erosion followed by a dilation and can be used to eliminate all pixels in regions that are too small to contain the structuring element. In this case the structuring element is often called a probe, because it is probing the image looking for small objects to filter out of the image. See Fig. 7.6 for the illustration of the opening process.

Notation:

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

Closing consists of a dilation followed by erosion and can be used to fill in holes and small gaps. In Fig. 7.7 we see that the closing operation has the effect of filling in holes and closing gaps. Comparing the left and right images from Fig. 7.8, we see that the order of operation is important. Closing and opening will generate different results even though both consist of erosion and dilation.

Notation:

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

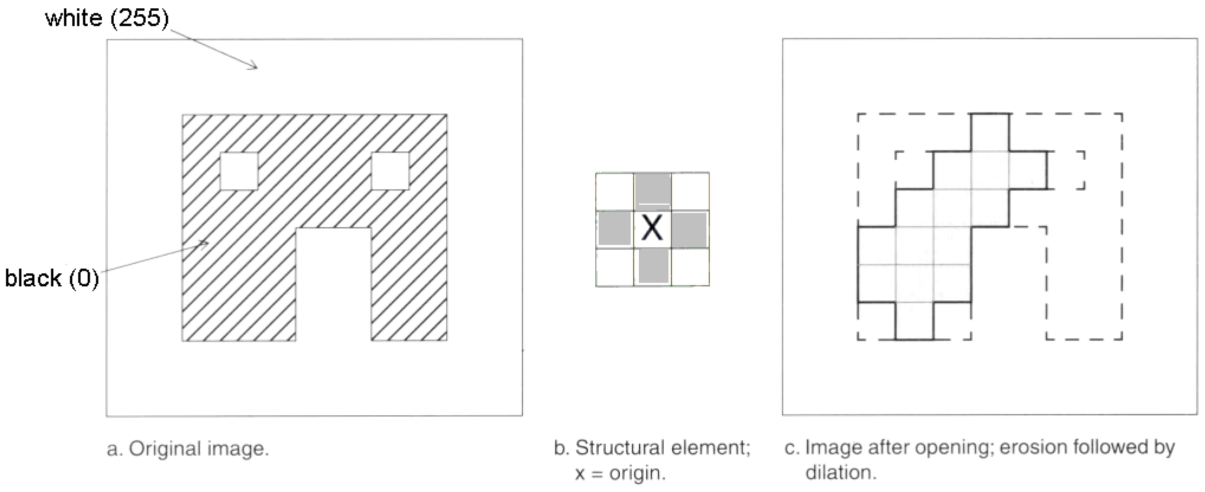


Fig. 7.6 Illustration of the opening process

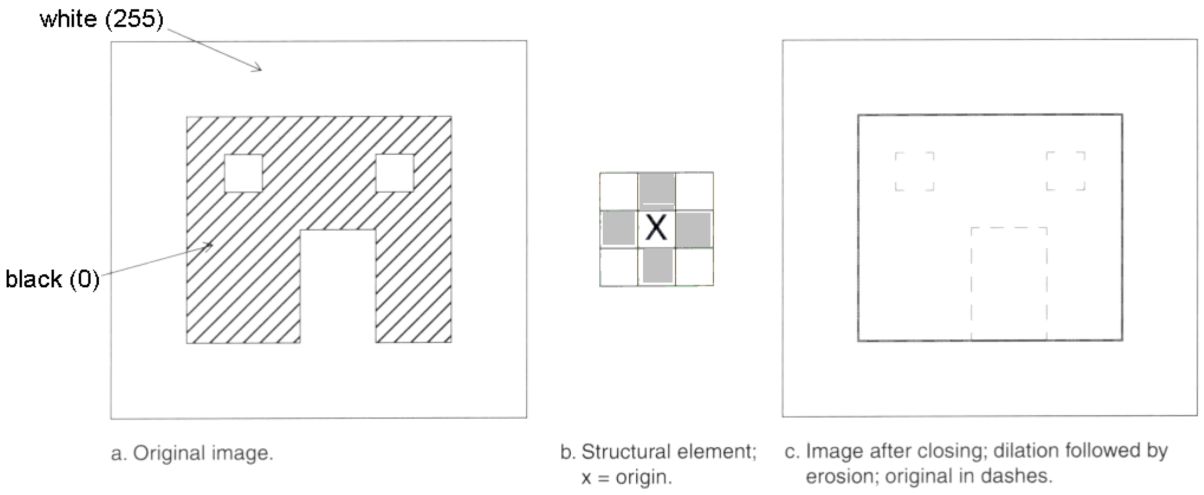
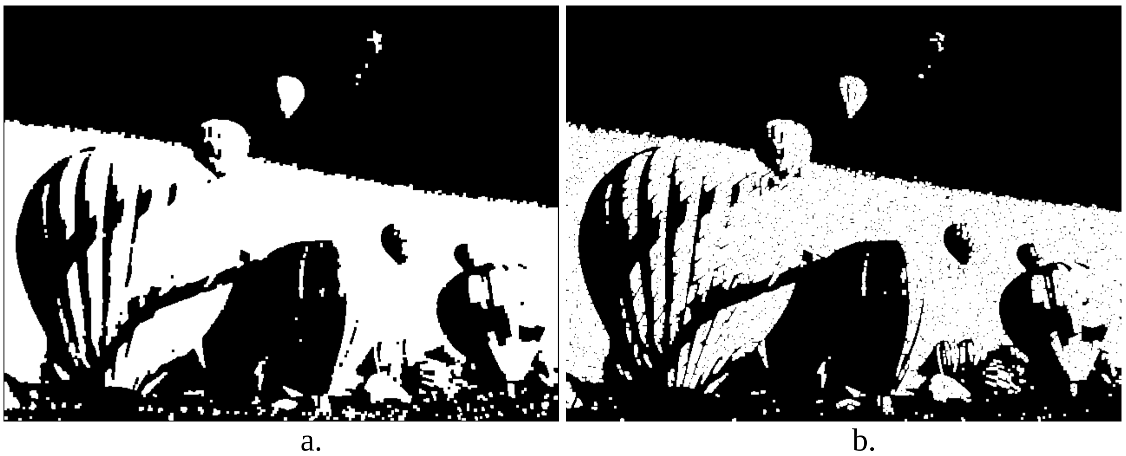

white (255)
black (0)
a. Original image.
b. Structural element;
x = origin.
X
c. Image after closing; dilation followed by erosion; original in dashes.

Fig. 7.7 Illustration of the closing process



a. b.

Fig. 7.8 Results of the opening (a) and closing (b) operations applied on the original image from Fig. 7.5a.

7.2.4. Some basic morphological algorithms [2]

7.2.4.1. Boundary extraction

The boundary of a set *A*, denoted by $\beta(A)$, can be obtained by first eroding *A* by *B* and then performing the set differences between *A* and its erosion. That is,

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

where

B is a suitable structuring element.

'–' is the difference operation on sets (illustrated in Fig. 7.10)

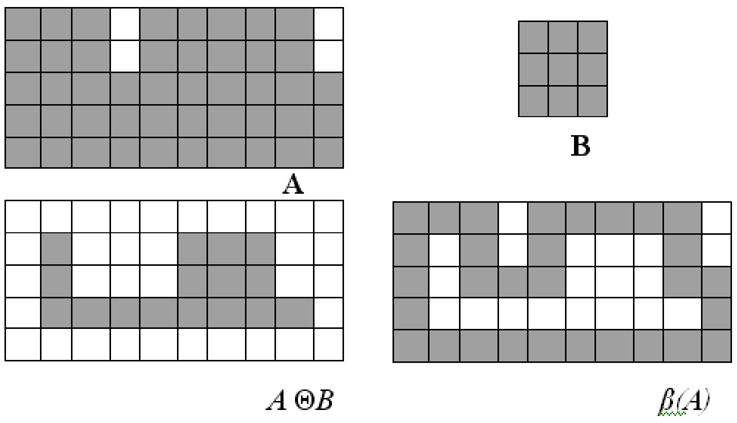


Fig. 7.9 Illustration of the boundary extraction algorithm

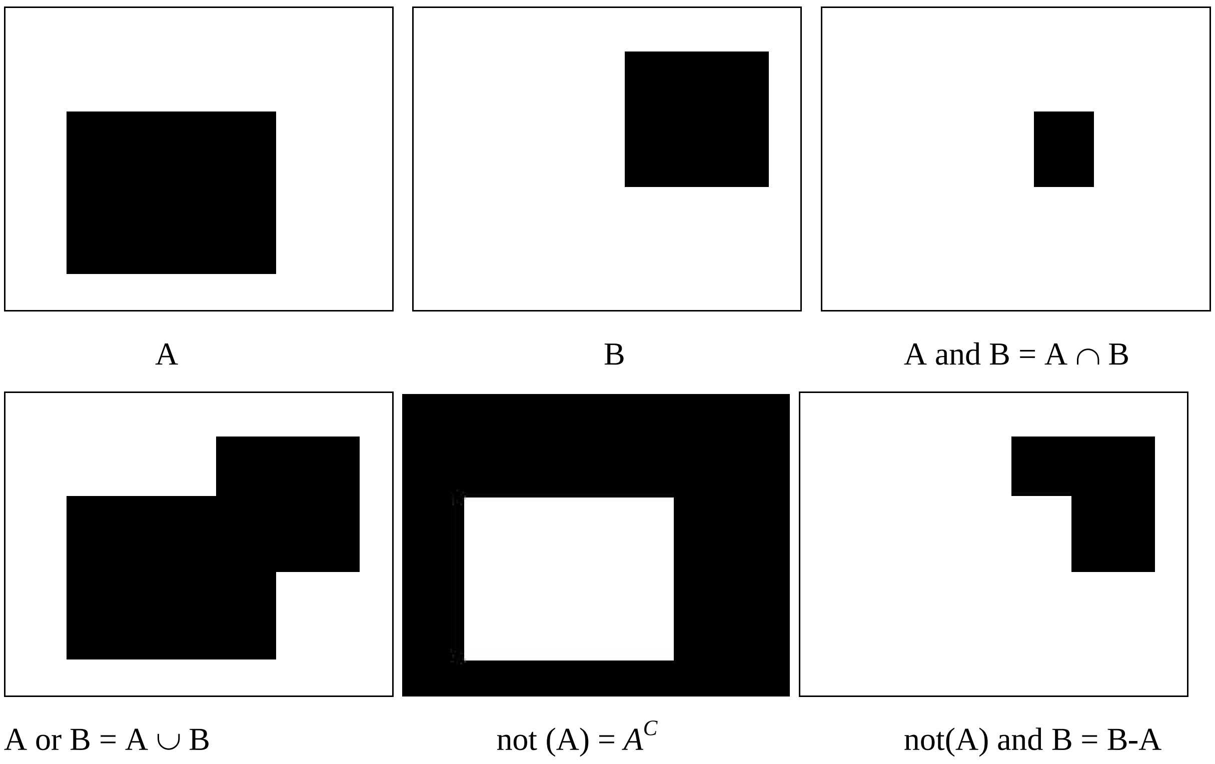


Fig. 7.10 Illustration of the main operations on sets

7.2.4.2. Region filling

Next we develop a simple algorithm for region filling based on set dilations, complementation, and intersections.

Beginning with a point p inside the boundary, the objective is to fill the entire region with ‘black’. If we adopt the convention that all non-boundary (background) points are labeled ‘white’, then we assign a value of ‘black’ to p to begin. The following procedure then fills the region with ‘black’:

$$X_k=(X_{k-1} \oplus B) \cap A^C \quad k=1,2,3,$$

where

$X_0=p$,
B is the symmetric structuring element
$\cap$ - is the intersection operator (see Fig. 7.10)
A^C – is the complement of set A (see Fig. 7.10)

The algorithm terminates at iteration step *k* if $X_k= X_{k-1}$. The set union of X_k and A contains the filled set and its boundary.

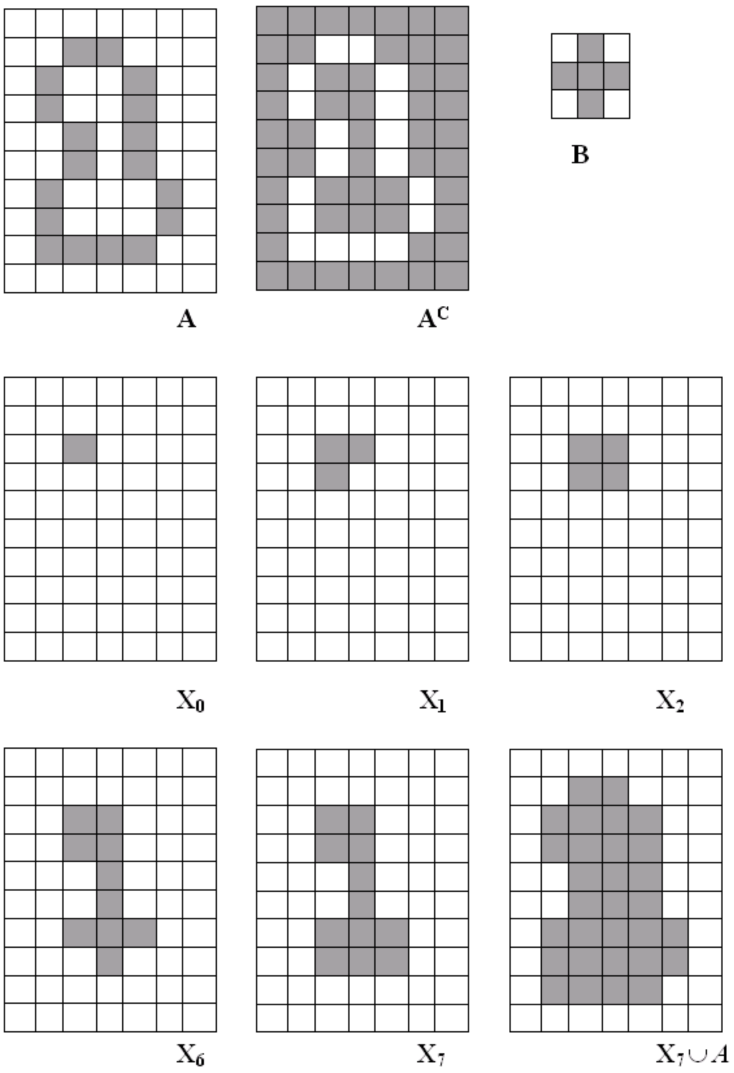


Fig. 7.11 Illustration of the region filling algorithm

7.3. Implementation hints

7.3.1. Using a supplementary image buffer for chain processing

The results of the basic morphological operations (dilation and erosion) should be applied in the following manner:

Destination image = *Source image* (operator) *Structuring element*

The source image shouldn't be affected in any way!

For the implementation of the combined morphological operations (opening and closing) or of the repeated operations (for example: *n* consecutive erosions) in a single processing function a supplementary image buffer should be created used. This can be done as follows (the code exemplifies how to perform two dilations, using an auxiliary temporary buffer):

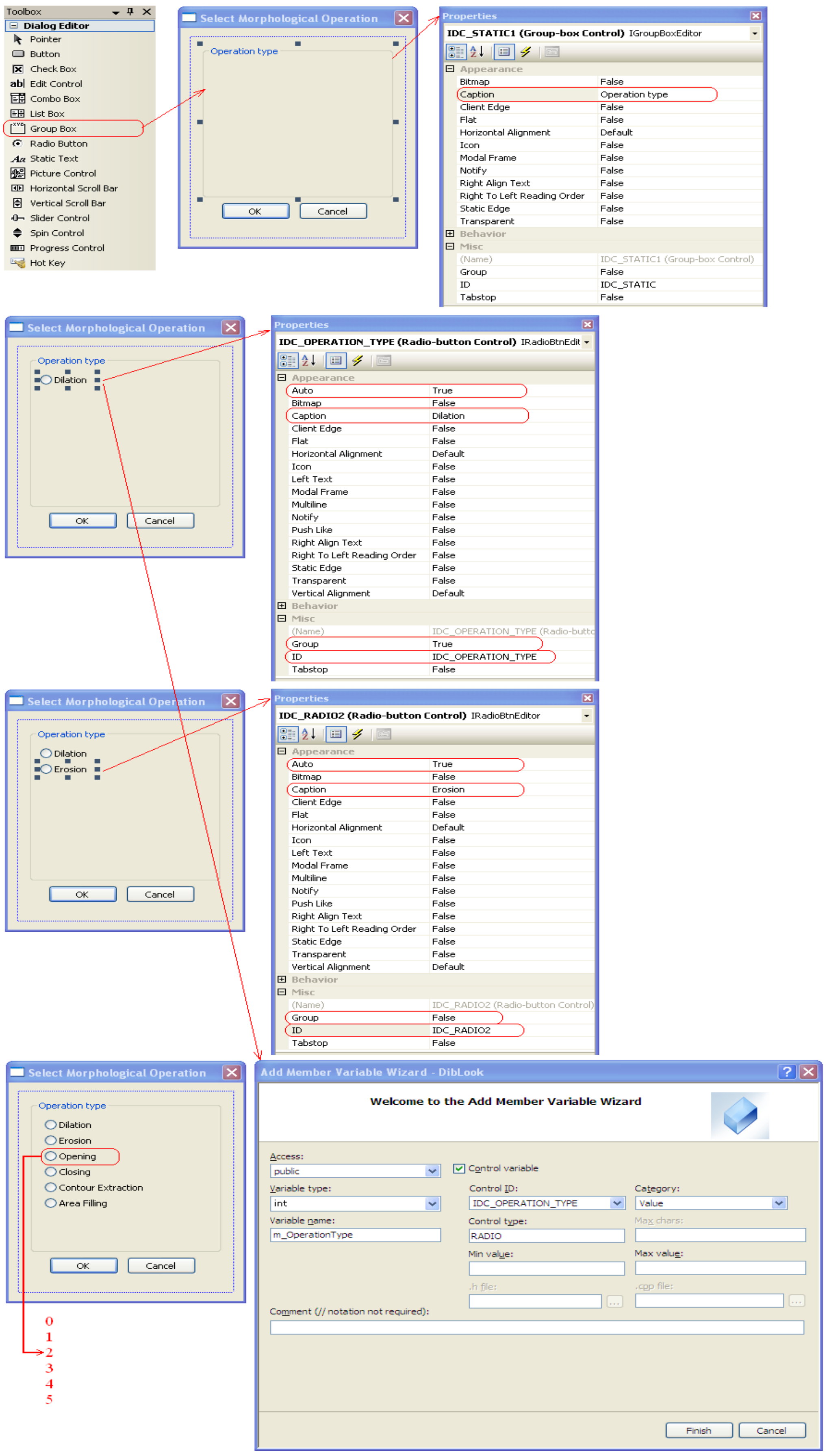
```
//allocate a temporary buffer.
//its dimensions should be w and dwHeight (same as
//the pixel array in the source and destination bitmap
unsigned char *lpTemp = new unsigned char [w*dwHeight];
//perform 1st dilation, and write the result
//in the temporary buffer
//perform 2nd dilation, and write the result
//in the destination image
//free the buffer! C++ doesn't have garbage collection
//use the delete[] operator, not delete!!
delete[] lpTemp;
```

7.3.2. Additional hints for designing an input dialog box

If you want select the type of the morphological operation which you want to apply and the number of its repetitions, you can use a dialog box as input. Creating a dialog box and using edit controls as inputs was presented in Laboratory 2.

The following example illustrates the implementation of a single selection from multiple options (for example the type the morphological operation, dilation, erosion, opening, closing, contour extraction, region filling) using radio buttons. A radio button is similar to a check box, with the difference that, in a group of radio buttons, only a single radio button may be selected at any time. In order to create a group of radio buttons, allow user interaction with it and obtain the selected button you must perform the following steps (see Fig. 7.12):

1. The first step is to create a new Dialog box and to add the corresponding class *CDlgSelectMorphologicalOperation* as explained in Laboratory 2.
2. In order to visually group the various radio buttons add a Group Box control on the dialog. Give the group box a suggestive name, such as "Operation Type"
3. Add the first radio button on the group box. Give it suggestive ID, such as IDC_OPERATION_TYPE. Make sure you select the "Group" option!! Also, for this and all subsequent radio buttons, make sure that the option "Auto" in the "Appearance" section is selected!!
4. Add the other radio buttons. It is not necessary to change their IDs. Make sure that their Group checkbox is NOT selected!!
5. Associate an integer member variable with the first control (the one with the group box selected.
6. The associated member variable holds the index of the selected control. The index is 0-based, i.e. if the first button is selected then the index is 0, if the second radio button is selected then the index is 1 and so on.



Properties

IDC_STATIC1 (Group-box Control) IGroupBoxEditor

Appearance	
Bitmap	False
Caption	Operation type
Client Edge	False
Flat	False
Horizontal Alignment	Default
Icon	False
Modal Frame	False
Notify	False
Right Align Text	False
Right To Left Reading Order	False
Static Edge	False
Transparent	False
Behavior	
Misc	
(Name)	IDC_STATIC1 (Group-box Control)
Group	False
ID	IDC_STATIC
Tabstop	False

Properties

IDC_OPERATION_TYPE (Radio-button Control) IRadioBtnEdit

Appearance	
Auto	True
Bitmap	False
Caption	Dilation
Client Edge	False
Flat	False
Horizontal Alignment	Default
Icon	False
Left Text	False
Modal Frame	False
Multiline	False
Notify	False
Push Like	False
Right Align Text	False
Right To Left Reading Order	False
Static Edge	False
Transparent	False
Vertical Alignment	Default
Behavior	
Misc	
(Name)	IDC_OPERATION_TYPE (Radio-butt
Group	True
ID	IDC_OPERATION_TYPE
Tabstop	False

Properties

IDC_RADIO2 (Radio-button Control) IRadioBtnEditor

Appearance	
Auto	True
Bitmap	False
Caption	Erosion
Client Edge	False
Flat	False
Horizontal Alignment	Default
Icon	False
Left Text	False
Modal Frame	False
Multiline	False
Notify	False
Push Like	False
Right Align Text	False
Right To Left Reading Order	False
Static Edge	False
Transparent	False
Vertical Alignment	Default
Behavior	
Misc	
(Name)	IDC_RADIO2 (Radio-button Control)
Group	False
ID	IDC_RADIO2
Tabstop	False

Add Member Variable Wizard - DibLook

Welcome to the Add Member Variable Wizard

Access: public	[✓] Control variable	
Variable type: int	Control ID: IDC_OPERATION_TYPE	Category: Value
Variable name: m_OperationType	Control type: RADIO	Max chars:
	Min value:	Max value:
	.h file:	.cpp file:

Comment (// notation not required):

Fig. 7.12 Adding a group box, radio buttons, and associating the first radio button control with an index integer meber variable

A clean way to obtain the desired operation selection from a group of radio buttons is:

1. Define an enumeration in your dialog class (*CDlgSelectMorphologicalOperation* in this example) holding the desired options. You should add this in the public section of your class, in the corresponding header file (*DlgSelectMorphologicalOperation.h* in this example):

```
class CDlgSelectMorphologicalOperation : public CDialog {
public:
   //enumeration holding the operation type
   //(enumerations are by default zero based)
   enum EOperationType {
         Dilation,
         Erosion,
         Opening,
         Closing,
         ContourExtraction,
         AreaFilling,
  };
    ………
    };
```

2. In your processing method (in *dibview.cpp*) add the following code to instantiate the dialog and obtain the selected operation:

```
//instantiate the dialog
CDlgSelectMorphologicalOperation dlgSelect;
//set the default selection to the first operation
dlgSelect.m_OperationType = 0;
//show the dialog in modal mode
dlgSelect.DoModal();
//obtain the selection
switch(dlgSelect.m_OperationType) {
case CDlgSelectMorphologicalOperation::Dilation:
      …… //code for dilation
      break; //do not forget to put break after each case
case CDlgSelectMorphologicalOperation::Erosion:
      …… //code for erosion
      break;
case CDlgSelectMorphologicalOperation::Opening:
      …… //code for opening
      break;
case CDlgSelectMorphologicalOperation::Closing:
      …… //code for closing
      break;
case CDlgSelectMorphologicalOperation::ContourExtraction:
      …… //code for contour extraction
      break;
case CDlgSelectMorphologicalOperation::AreaFilling:
      …… //code for region filling
      break;
}
```

7.4. Practical work

1. Add to the DIBLook framework processing functions which implement the basic morphological operations.
2. Add the facility to apply the morphological operations repeatedly (*n* times). For that purpose use a dialog box to input the number of repetitions *n* (through an Edit control) and to select the type of the morphological operation (through radio buttons).
3. Implement the boundary extraction algorithm.
4. Implement the region filling algorithm.

5. **Save your work. Use the same application in the next laboratories. At the end of the image processing laboratory you should present your own application with the implemented algorithms!!!**

References

[1]. Umbaugh Scot E, *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8
[2] R.C.Gonzales, R.E.Woods, *Digital Image Processing. 2-nd Edition*, Prentice Hall, 2002.