4. Geometrical features of binary objects

4.1. Introduction

This lab work presents some important geometric properties of binary images and the algorithms used for computing them. The properties described are: the area, the center of mass, the elongation axis, the perimeter, the thinness ratio, the aspect ratio and the projections of the binary image.

4.2. Theoretical considerations

After applying segmentation and labeling algorithms to images we obtain a new image in which each object can be referenced separately.
An object 'i' is described by the function:

$$I_i(r,c) = \begin{cases} 1, & if\ I(r,c) \in \text{object labeled 'i'} \\ 0 & \text{otherwise} \end{cases}$$

The geometric properties of objects can be classified into two categories:

- Position and orientation properties: the center of mass, the area, the perimeter, the elongation axis
- Shape properties: aspect ratio, thinness ratio, Euler's number, the projections, the Feret diameters of the objects

4.2.1. Area

$$A_i = \sum_{r=0}^{N-1}\sum_{c=0}^{N-1} I_i(r,c)$$

The area A_i is measured in pixels and it indicates the relative size of the object.

4.2.2. The center of mass

$$\bar{r}_i = \frac{1}{A_i}\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} rI_i(r,c)$$

$$\bar{c}_i = \frac{1}{A_i}\sum_{r=0}^{N-1}\sum_{c=0}^{N-1} cI_i(r,c)$$

The equations above correspond to the row and column where the center of mass is located. This attribute helps us locate the object in a bi-dimensional image.

4.2.3. The axis of elongation (The axis of least second order moment)

$$\tan(2\phi_i) = 2\frac{\sum_{r=0}^{N-1}\sum_{c=0}^{N-1}(r-\bar{r}_i)(c-\bar{c}_i)I_i(r,c)}{\sum_{r=0}^{N-1}\sum_{c=0}^{N-1}(c-\bar{c}_i)^2 I_{i(r,c)} - \sum_{r=0}^{N-1}\sum_{c=0}^{N-1}(r-\bar{r}_i)^2 I_i(r,c)}$$

If both the nominator and the denominator of the above equation are equal to zero, than the object has a circular symmetry, and any line that passes through the center of mass is a symmetry axis.

For finding the direction of the line (the angle) one must apply the arctangent function. The arctangent is defined on the interval $(-\infty, +\infty)$ and it takes values in the interval $(-\pi/2, \pi/2)$. The evaluation of the arctangent becomes unstable when the denominator of the fraction tends to zero.

The signs of the numerator and of the denominator are important for determining the right quadrant in which the result lays. The arctangent function does not make the difference between directions that are opposed. For this reason the usage of the function "atan2" is suggested. The "atan2" function has as arguments the numerator and the denominator of such fraction, and it returns a result in the interval $(-\pi, \pi)$.

The axis of elongation gives information about how the object is positioned in the field of view, that is its orientation. The axis corresponds to the direction in which the object (seen as a plane surface of constant width) can rotate most easily (has a minimum kinetic moment).

4.2.4. The perimeter

The perimeter of the object helps us determine the position of the object in space and it also gives information about the shape of the object. The perimeter can be computed by counting the number of pixels on the contour (pixels of value 1 and having at least one neighbor pixel of value 0).

A first approach to contour detection is the scanning of the image, line by line and counting the number of pixels in the object that satisfy the condition mentioned above. A main disadvantage of this method is that we cannot distinguish the exterior contour from the interior contours (if they exist they are generated by the holes in the object). As the pixels of digital images represent distributions on a rectangular raster, the length of curves and oblique lines in the image cannot be correctly estimated by counting the pixels. A first correction is given by the multiplication by $\pi/4$ of the perimeter that resulted in the previous algorithm. There are other methods for length correction. These methods take into account the type of neighborhood used (4 neighbors, 8 neighbors etc).

Another method for detecting the contour of an object involves the usage of an existing algorithm for edge detection, the thinning of the edges until they become 1 pixel thick and in the end the counting of the resulted edge pixels.

Methods of type "chain-codes" represent complex methods for contour detection and offer a high accuracy.

4.2.5. The thinness ratio (circularity)

$$T = 4\pi\left(\frac{A}{P^2}\right)$$

The function above has the maximum value equal to 1, and for this value we obtain a circle. The thinness ratio is used for determining how "round" an object is. If the value of T is close to 1, the object tends to be round.

The value of the thinness ratio also offers information on how regular an object is. The objects that have a regular contour have a greater value of T than the objects of irregular contours. The value 1/T is called irregularity factor of the object (or compactness factor).

4.2.6. The aspect ratio

This property is found by scanning the image and keeping the minimum and maximum values of the lines and columns that form the rectangle circumscribed to the object.

$$R = \frac{c_{\max} - c_{\min} + 1}{r_{\max} - r_{\min} + 1}$$

4.2.7. The projections of the binary object

The projections give information about the shape of the object. The horizontal projection equals the sum of pixels computed on each line of the image, and the vertical projection is given by the sum of the pixels on the columns.

$$h_i(r) = \sum_{c=0}^{N-1} I_i(r,c)$$

$$v_i(c) = \sum_{r=0}^{N-1} I_i(r,c)$$

The projections are used in applications of text recognition in which the interest object can be normalized.

4.3. Implementation details

In order to distinguish between the various objects present in an image, we will suppose that each one of them is painted using a different color. These colors may be the result of a previous labeling step, or may be generated manually (see Fig. 4.1).

There are various approaches for implementing the geometrical properties extractors. A simple approach is to compute them for all objects in an image at once. There are at most 255 objects plus background in the image format described above, each one having a different color index.

The second approach is to use the mouse to select an object. The user should position the mouse pointer over a pixel belonging to the desired object and double-click on it. In response to this action, you should display a message box containing the values of the desired features.

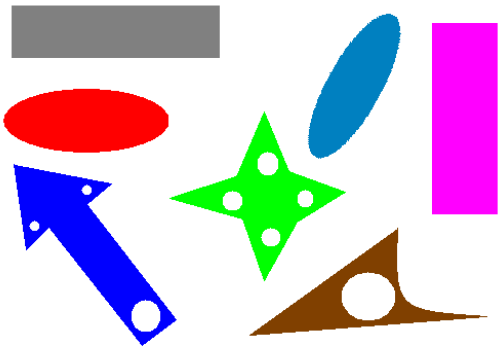

Fig. 4.1 Example of a labeled image on which the algorithms described here could be tested

In order to add a handler for the double click event you must follow these steps (see Fig. 4.2):

1. On the workspace window select the *Class View* tab.
2. Right click on the *CDibView* class and choose *Properties…*
3. In the *Properties* window choose the *Messages* section.
4. On the *WM_LBUTTONDBLCLK* add the method *OnLButtonDblClk* (*<Add> OnLButtonDblClk*) by using the combo-box. The wizard will generate a message handler method, receiving as a parameter the coordinates of the mouse pointer, relative to the view's at the time when the double-click occurred.

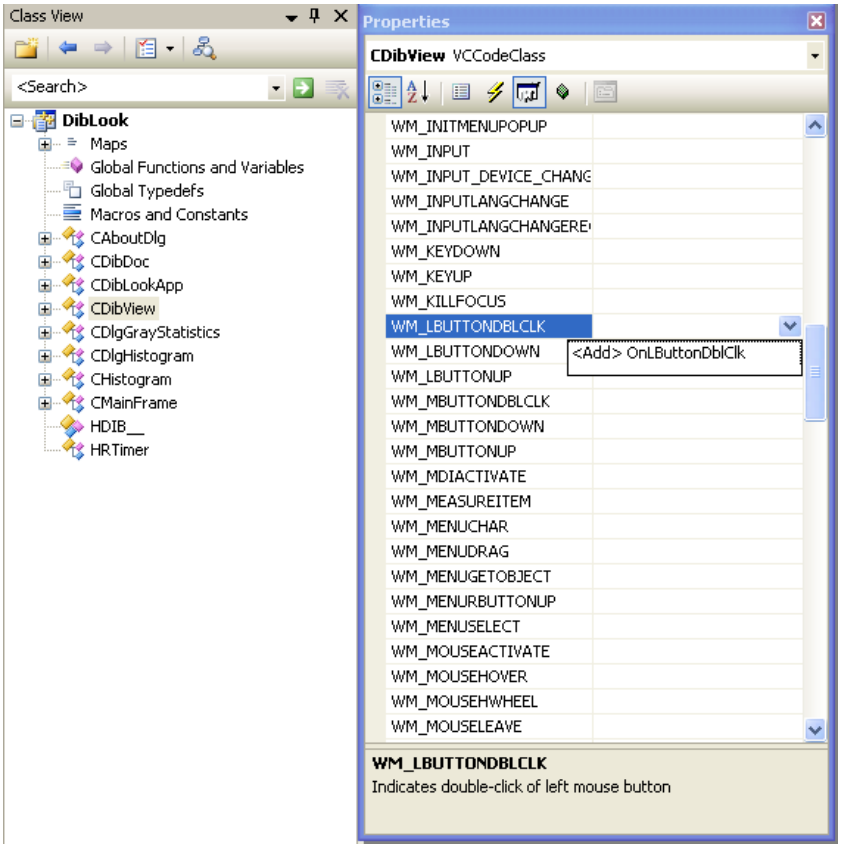

Fig. 4.2 Adding a new message handler for the double-click using the left mouse button in *CDibView* class

The following code presents a stub mouse event handler that computes the coordinates of the mouse click in the image, and displays a message box with the coordinates and the color index of the pixel over which the click occurred.

```
void CDibView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
     BEGIN_SOURCE_PROCESSING;

     //obtain the scroll position (because of scroll bars' positions
     //the coordinates may be shifted) and adjust the position
     CPoint pos = GetScrollPosition()+point;

     //point contains the window's client area coordinates
     //the y axis is inverted because of the way bitmaps
     //are represented in memory
     int x = pos.x;
     int y = dwHeight-pos.y-1;
```

```
     //test if the position is inside the image
     if (x>0 && x<dwWidth && y>0 && y<dwHeight)
     {
          //prepare a CString for formating the output message
          CString info;
          info.Format("x=%d, y=%d, color=%d", x, y, lpSrc[y*w+x]);
          AfxMessageBox(info);
     }

     END_SOURCE_PROCESSING;

     //call the superclass' method
     CScrollView::OnLButtonDblClk(nFlags, point);
}
```

A third approach is to either select an individual object or compute the features for all objects and display the results directly on the destination image. An easy way to display text and graphics on an image is to use the Windows GDI (Graphics Device Interface) functions.

Each graphical operation in Windows must be accomplished through a DC (Device Context) object. This object holds such data as the device driver that performs the drawing (the driver for the graphics card, printer, memory device), the surface on which the drawing is performed (the main display surface, a back surface, a surface located in main memory), the current drawing pen (color, line width) the current brush (for filling surfaces) and so on.

In order to draw on a bitmap, the bitmap must be "selected" in the DC, so that all subsequent drawing will be done over the image. The device independent bitmaps (DIBs) used in DIBView cannot be selected directly in a DC. In order to cope with this problem, a device dependent bitmap (DDB) must be created, and the data from the source image copied to it. Next, the DDB is selected in a memory DC and drawing is performed. Finally, the pixels in the DDB are copied back to the destination DIB. The following code performs all the above steps. It also displays a line and a text at the coordinates where a mouse double click occurred.

```
void CDibView::OnLButtonDblClk(UINT nFlags, CPoint point)
{
     BEGIN_PROCESSING();

     CDC dc; //memory DC
     dc.CreateCompatibleDC(0); //create it compatible with the screen

     CBitmap ddBitmap;//to hold a device dependent bitmap compatible with
                      //the screen

     //create a DDB, compatible with the screen
     //and initialize it with the data from the source DIB
     HBITMAP hDDBitmap =
     CreateDIBitmap(::GetDC(0), &((LPBITMAPINFO)lpS)->bmiHeader, CBM_INIT,
     lpSrc, (LPBITMAPINFO)lpS, DIB_RGB_COLORS);

     //attach the handle to the CBitmap object
     ddBitmap.Attach(hDDBitmap);

     //select the DDB into the memory DC
     //so that all drawing will be performed on the DDB
     CBitmap* pTempBmp = dc.SelectObject(&ddBitmap);

     //from this point onward, all drawing done using the DC object
     //will be made on the DDB
```

```
    //obtain the scroll position (because of scroll bars' positions
    //the coordinates may be shifted) and adjust the position
    CPoint pos = GetScrollPosition()+point;

    //create a green pen for drawing
    CPen pen(PS_SOLID, 1, RGB(0,255,0));

    //select the pen on the device context
    CPen *pTempPen = dc.SelectObject(&pen);
    //draw a text
    dc.TextOut(pos.x,pos.y, "test");
    //and a line
    dc.MoveTo(pos.x,pos.y);
    dc.LineTo(pos.x, pos.y-20);

    //select back the old pen
    dc.SelectObject(pTempPen);
    //and the old bitmap
    dc.SelectObject(pTempBmp);

    //copy the pixel data from the device dependent bitmap
    //to the destination DIB
    GetDIBits(dc.m_hDC, ddBitmap, 0, dwHeight, lpDst, (LPBITMAPINFO)lpD,
    DIB_RGB_COLORS);

    END_PROCESSING("line");
}
```

4.4. Practical work

1. For each individual object in a labeled image compute the object's area, center of mass, axis of elongation, thinness ratio and aspect ratio. For displaying the results, you can chose among the following 2 options (presented in section 4.3):
    a. display the geometrical features of all objects in a dialog box
    b. display the geometrical features of each object in a message box when the area of the object is double-clicked
2. Compute and display the object's projections
3. Display objects' axes of elongation on the destination image using the GDI drawing functions.
4. Display mass centers and areas on the destination image using the GDI drawing functions
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!!!**

Bibliography

[1]. Umbaugh Scot E, *Computer Vision and Image Processing*, Prentice Hall, NJ, 1998, ISBN 0-13-264599-8