5. Binary objects labeling

5.1. Introduction

In this laboratory an object labeling algorithm which allows you to label distinct objects from a binary (black&white) image is presented. This algorithm is useful for the separation of distinct objects for further analyses/measurements applied on each individual object.

5.2. Theoretical considerations

In order to extract specific features of the objects from a digital image, it is necessary to perform a segmentation process to the original image. As a result of the segmentation operation, the obtained image will contain well-differentiated objects. The purpose of the labeling process is to assign to each distinct object a unique label (integer number). In the following, a fast labeling algorithm (which scans the image pixels of a binary image only once) will be presented.

Algorithm steps

1. Labeling the pixels from the source image, using a single image scan, and establishing the equivalent label pairs.
2. Establishing equivalence labeling classes.
3. Updating operation needed to replace each pixel's label with the label of its equivalence class.

Step 1. Labeling the pixels from the source image and establishing the equivalent label pairs.

The labeling process needs first to define the type of connectivity used. Then the image is scanned (line by line, in top-down and left-right order) and connected pixels are labeled with the same label. The presented algorithm uses a 5-conectivity (Fig. 5.1), in which the current pixel is denoted with 'X':

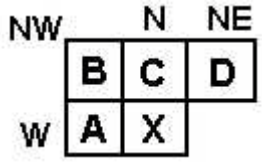

Fig. 5.1 The 5-conectivity used.

The algorithm for the general case is presented in Fig. 5.2. We consider pixels having a value different from "0" belonging to the background (in 8 bits/pixels bitmap images, object points are black – intensity value =0). In figure 2 the following notations are used:

- X_Label, A_Label, B_Label, C_Label, D_Label are the labels assigned to pixels X, A, B, C and D respectively (see Fig. 5.1);
- NewPair(Label1,Label2) is a function which appends pair (Label1, Label2) in the list of the equivalent label pairs.
- NewLabel is a function used to generate a new label (increments the last assigned label with 1 (the first generated new label is 1)).

Observation: The pixel labels of the source image are kept in an integer matrix having the same dimensions as the source image. In the labels' matrix, the background pixels are labeled with "0".

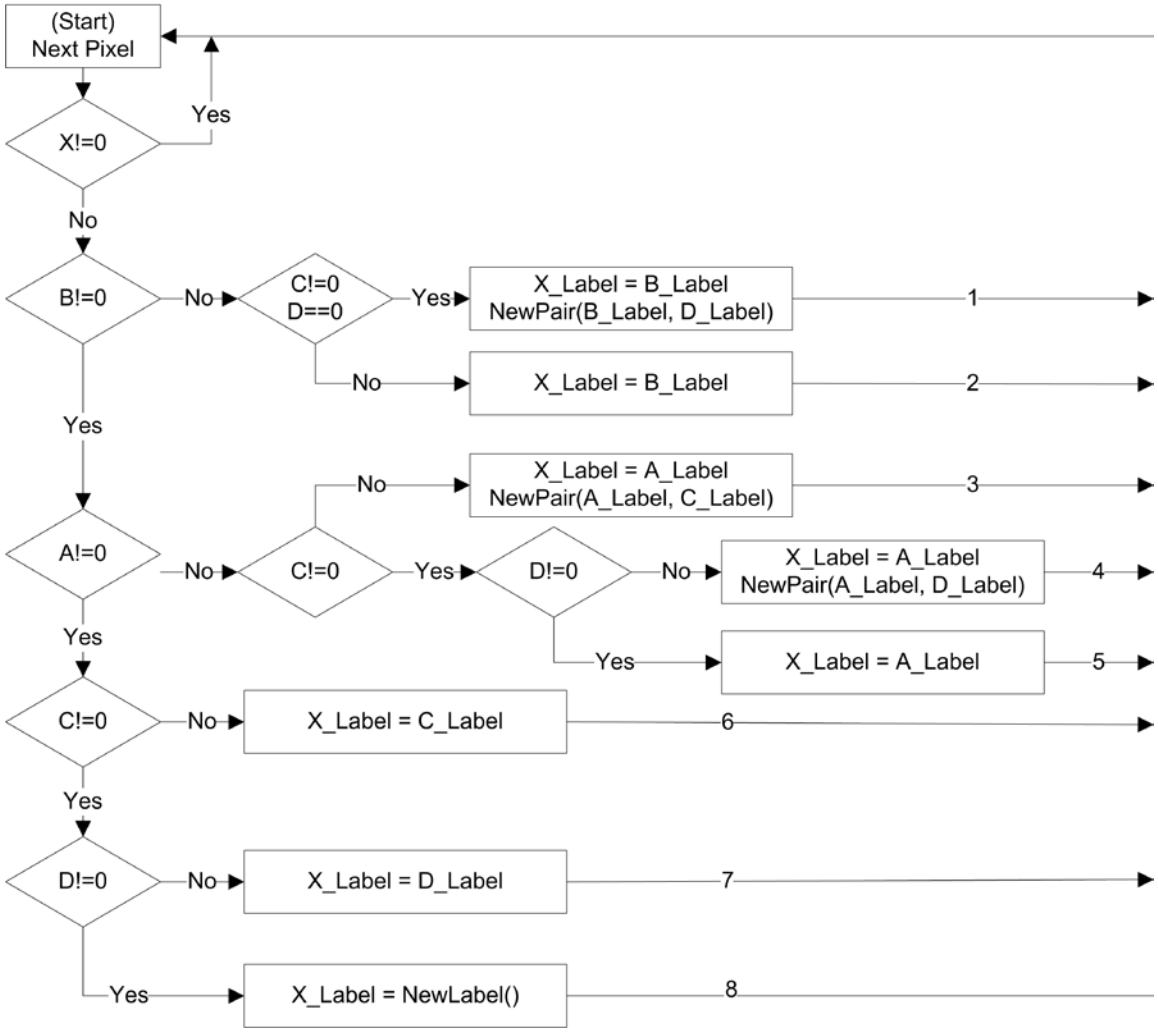

Fig. 5.2 Block diagram of the labeling algorithm

The labeling algorithm is applied for each individual pixel by scanning the pixels in a top-down left-right manner. **Observation:** in order to use the 5 pixel connectivity (Fig. 5.1) the image scanning must start from the second line from the top and second column from the left and will finish on the second to last line and second to last column. This is because the four neighbors of the labeled pixel W, NW, N, NE (pixels A, B, C, D, Fig. 5.1) must be located inside the image; otherwise the memory buffer in which the image is located may be exceeded.

The pixel labeling and equivalent label generation algorithm is described next for the following cases:

a) If the current pixel belongs to the background then we skip to the next pixel;
b) If the current pixel belongs to an object, then it must be labeled, and the following cases may occur:

- If the neighbors from NW and NE are object pixels and the neighbor from the N direction is not an object pixel, then the label of the current pixel will be the same as NW neighbor's label and an equivalent pair will be added for the NW and NE directions (branch 1, Fig. 5.2);
- If the neighbor from NW is an object pixel and either the neighbor from the N is an object pixel or the neighbor from NE is not an object pixel, then the label of the current pixel will be the same as the label of the pixel from NW (branch 2, Fig. 5.2);

- If the neighbor from NW is not a pixel object and the neighbors from W and N are object pixels then the label of the pixel will be the same as the label of the W pixel and the equivalent pair will be added for the W and N neighbors (branch 3, Fig. 5.2);
- If the neighbors from the NW and N are not object pixels and the neighbors from W and NE are object pixels, then the label of the pixel will be made the same as the label of the W pixel and an equivalent pair is added for the W and NE pixel labels (branch 4, Fig. 5.2);
- If the neighbor from the W direction is an object pixel and the neighbors from NW, N and NE are not object pixels, then the label of the current pixel will be equal to the label of the pixel from the W direction (branch 5, Fig. 5.2);
- If the neighbors from the NW and W directions are not object pixels and the neighbor from the N direction is an object pixel, then the label of the current pixel will be the same as the label of the N pixel (branch 6, Fig. 5.2);
- If the neighbors from NW, W and N are not object pixels and the neighbor from the NE direction is an object pixel, then the label of the current pixel will be the same as the label of the NE pixel (branch 7, Fig. 5.2);
- If none of the pixel's neighbors (NW, W, N and NE) are object pixels, then a new label will be generated for the current pixel (branch 8, fig. Fig. 5.2).

Multiple labeling of an object appears in the case of sequential scanning, in the case of "J" shape objects, as you can see in figure 3. The labeling process begins to label two "different" objects with labels "1" and "2", until the pixel "X" is reached, where we found that object 1 and 2 are connected. At this stage the two labels are considered equivalent and are appended to the list of equivalent pairs (NewPair(1,2)).

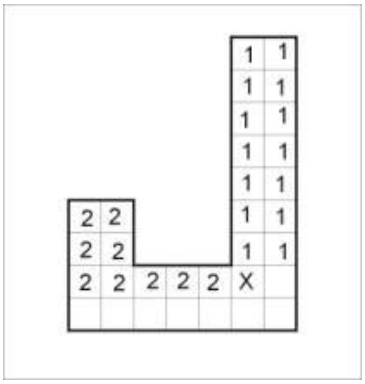

Fig. 5.3 The case of multiple labeled objects

Step 2. Establishing the labeling equivalence classes

After all the pixels were labeled and the equivalence pairs were established, the equivalence classes must be found. The equivalence pairs are binary relations stored in the list. To find the equivalence classes a graph can be used in which the nodes are the labels, and the arches are the equivalence binary relations. The problem of finding the equivalence classes is to find the connected sub graphs (the transitive closure of the equivalence relationships).

To find a connected sub-graph a breadth first search can be used, starting from any graph node. All the neighbors of the starting node are put in a queue and are marked as visited. (in order to track the nodes that were already considered as part of the current

connected sub-graph); when a node is extracted from the queue, its neighbors are added (if they were not previously visited) and are marked) and are marked as visited. The process is repeated for each node in the queue, until the queue becomes empty. All the nodes that went through the queue will bear the same label which will be associated to the equivalence class Next, an unvisited node is found and its equivalence class is built. This process continues until all the graph's nodes have been visited. An isolated node represents an equivalent class containing a single element.

For relabeling, an integer array may be used. The new label for the old label *i* will be located at index *i*. All array's elements are initialized with 0. This array may also be used for keeping track of the labels that were not previously visited: a 0 value marks that the label (graph node) was not previously visited.

One must go through all the equivalent pair list in order to search for one node's neighbors (the node is the first element of the equivalence pair and the neighbor is the other). This adjacency list graph representation has the disadvantage of long search times. A adjacency matrix can be used, but it would require a lot of memory. A sparse matrix representation can be used instead.

Step 3. Re-labeling all the labels with the labels corresponding to the equivalence classes.

The label image is scanned, and each label is replaced by the new label corresponding to its equivalence class. This operation can be accomplished by using the array described above.

5.3. Labeling examples

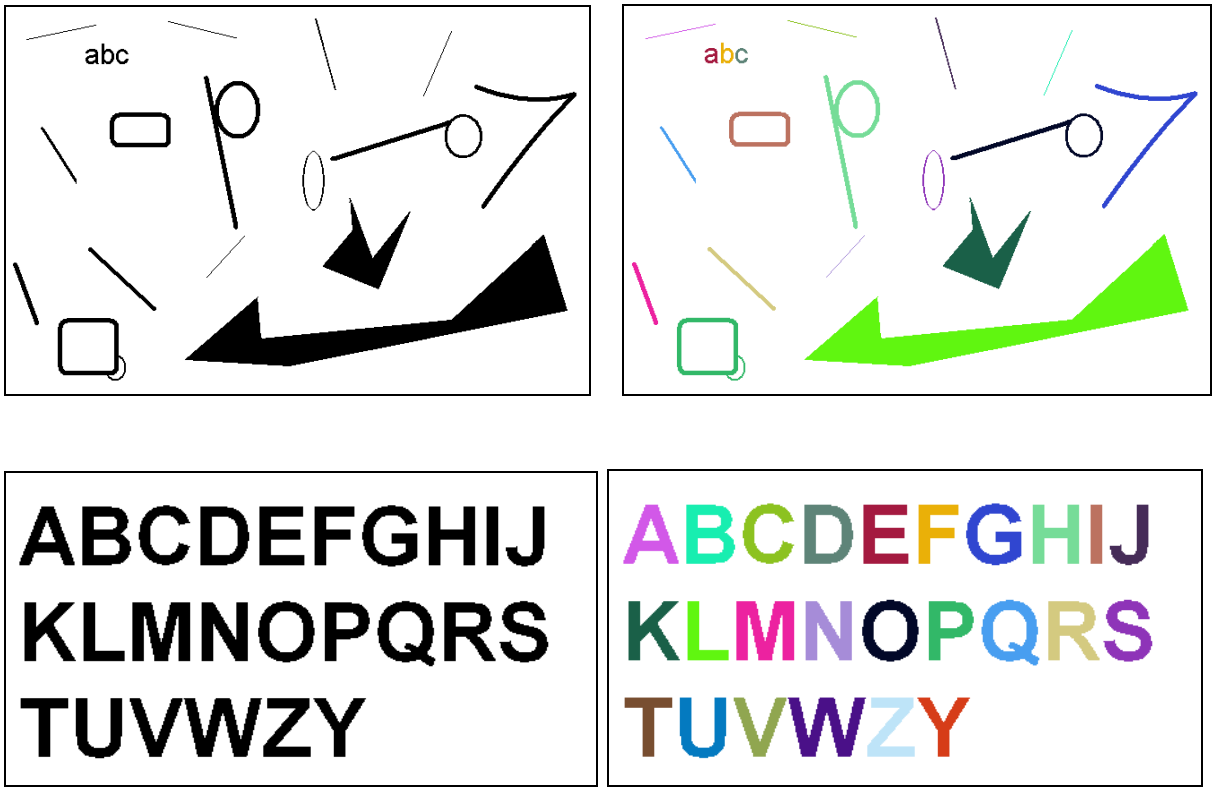

Fig. 5.4 Labeling examples

5.4. Implementation hints

For labeling, the source image must be scanned in a top-down, left-right manner. Because of the five pixel connectivity used (Fig. 5.1), the scanning will proceed from the second line and second column from the top and will end at the second to last line and second to last column (see step 1, section 5.2).

```
for (int i=dwHeight-2;i>=0;i--)
     for (int j=1;j<dwWidth-1;j++)
          if (lpSrc[i*w+j]==0)
          {
               // if the current pixel is black
               // the labeling algorithm is applied
               // the value is stored in the label image
               // the equivalence pairs are also stored
          }
```

Observations:

1. The bitmap image is vertically flipped in memory (line 0 from the memory corresponds to the bottom line of the image).
2. For the label image (matrix) an *int* array having the same size as the image will be allocated.

After the object pixels were labeled, the equivalence classes are determined based on the pairs on the set of equivalent labels stored previously (Step 2, section 5.2).

The last step is to re-label the pixels with the label of the equivalence class to which their old label belongs. Displaying the labeled objects is made with different (highly contrasting colors) can be done by modifying the 1..254 LUT entries (the 0 and 255 are not modified because the 0 value is a marker of an inexistent equivalence class and the value 255 is reserved for the white background). This works if the resulting equivalence classes have labels from 1 to 254.

```
// modifying the LUT for displaying with different colors
     for (int k=1;k<=254;k++)
     {
          // generarate a random color for index k1≤k≤254
          bmiColorsDst[k].rgbRed = randomValueRed;
          bmiColorsDst[k].rgbGreen = randomValueGreen;
          bmiColorsDst[k].rgbBlue = randomValueBlue;
     }
```

Observations:

1. The binary image used for labeling must be an 8bpp image with a sorted LUT which contains only black pixels (index 0 – for objects) and white pixels (value 255 – for background)
2. Using the previously presented coloring method with different colors for all labeled objects allows only 254 different colors for displaying the labeled objects.

5.5. Practical Work

1. Implement the steps the binary object labeling algorithm.
2. Add a processing function to the DIBLook framework for executing this algorithm. Display the labeled objects using different colors.
3. **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
[2]. Robert M. Haralick, Linda G. Shapiro, *Computer and Robot Vision*, Addison-Wesley Publishing Company, 1993.