(continue from DIP1.DOC)

(CONTINUE FROM DIP1.DOC) 46

4 BINARY IMAGE PROCESSING 47

4.1 BOUNDARY EXTRACTION 47
4.2 DISTANCE TRANSFORM 48
4.3 SKELETONIZING 51
4.4 COMPONENT LABELING 53
4.5 HALFTONING 53
4.6 MORPHOLOGICAL OPERATIONS 56

5 COLOR IMAGE PROCESSING 61

5.1 COLOR QUANTIZATION OF IMAGES 61
5.2 PSEUDO-COLORING 63

6 APPLICATIONS 66

6.1 RASTER TO VECTOR CONVERSION 66
6.2 OPTICAL CHARACTER RECOGNITION 66
6.3 CELL PARTICLE MOTION ANALYSIS 69
6.4 MAMMOGRAPHY CALCIFICATION DETECTION 71

LITERATURE 73

4 Binary image processing

4.1 Boundary extraction

A similar algorithm to edge following is known as *boundary tracing*. It is assumed that the objects consist of black pixels only, and the white pixels belong to the background. Let us first define the properties called *4-connectivity* and *8-connectivity*. Two pixels are said to be 4-connected to each other if they are neighbors via any of the four cardinal directions (N, E, S, W). Two pixels are said to be 8-connected if they are neighbors via any of the eight directions (N, NE, E, SE, S, SW, W, NW). Moreover, an object is said to be 4-connected if any of its pixels can be reached from any other pixel of the same object by traversing via 4-connected pixel pairs, see Figure 4.1. An 8-connected object is defined in a similar manner.

The boundary tracing algorithm starts with any boundary pixel of the object. The algorithm takes advantage of the property, that if an object is 8-connected, the set of white pixels surrounding the object is then 4-connected. For example, the white pixels marked 1, 2, and 3 in Figure 4.2 form part of the exterior boundary of the object, and they are 4-connected. The boundary is followed, not by traversing the boundary pixels, but by traversing the white pixels just outside the object. It is quicker to identify a 4-connected boundary than an 8-connected one, since fewer tests are involved.

Algorithm for boundary tracing:

1. SET *dir* ← east.
2. REPEAT UNTIL back at starting pixel:
   (a) IF pixel(turnright(*dir*)) is white, THEN
   SET *dir* ← turnright(*dir*)
   ELSE IF pixel(*dir*) is white, THEN
   leave *dir* unchanged
   ELSE IF pixel(turnleft(*dir*)) is white, THEN
   SET *dir* ← turnleft(*dir*)
   ELSE
   SET *dir* ← reverse(*dir*).
   (b) Go one step forward in direction *dir*.
   (c) Adjust bounding box parameters if necessary.

Let us examine how the algorithm works for the object given in Figure 4.2. The top-leftmost pixel (marked as current) is chosen as the seed of the algorithm. The algorithm starts with the white pixel at the top of the seed pixel, and the starting direction is east. After the second pixel the direction changes to south, which is followed until the fifth pixel is reached. At this stage the direction is reversed to north, and the third and fourth pixels are traversed again, this time in the opposite direction. The first 15 pixels of the exterior boundary are illustrated in Figure 4.3. The corresponding traversing directions are (E, S, S, S, N, N, N, E, E, E, E, S, S, W, W).

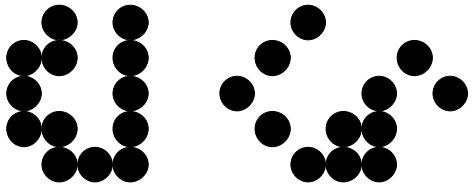


Figure 4.1: Example of a 4-connected object (left), and an 8-connected object (right).

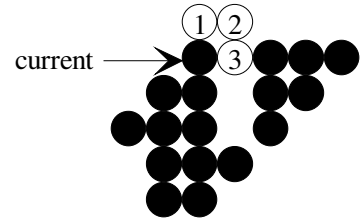


Figure 4.2: Example of boundary tracing.

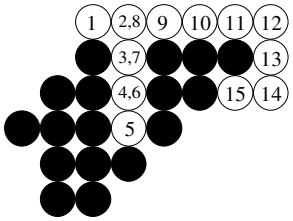


Figure 4.3: The 15 first steps of the boundary tracing algorithm.

4.2 Distance transform

Consider two points $p_1=(x_1, y_1)$, and $p_2=(x_2, y_2)$. The distance between these points is usually measured by one of the three *distance metrics* given below. See Figure 4.4 for an example of these distance metrics.

Euclidean distance:

$$D_e(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.1}$$

City block distance:

$$D_4(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2| \tag{4.2}$$

Chessboard distance:

$$D_8(p_1, p_2) = \max(|x_1 - x_2|, |y_1 - y_2|) \tag{4.3}$$

Let's define that objects consist of black pixels (1), and white pixels (0) are background. A *distance transform* is an operation where the distance between a pixel and the nearest background pixel is calculated for every object pixel. The *8-distance transform* can be calculated by a two-pass algorithm using the local pixel windows of Figure 4.5. It is assumed that the image pixels are already labeled so that the object pixels have been labeled as 1 and background pixels labeled as 0.

The *forward pass* operates the image in *row-major order* (from top to down, and from left to right). Each pixel is re-labeled as the minimum of its neighbors plus one. At the *backward pass* the image is processed in reverse order from bottom to up, and from right to left. Each pixel is re-labeled as the minimum of itself, and of its neighbors plus one. See Figures 4.6 and Figure 4.7 as examples.

The *4-distance transform* can be calculated by a similar algorithm, but using windows with only two neighbors (W and N in forward pass; E and S in backward pass where the NW and NE pixels).

Algorithm for 8-distance transform:

1. Initialize the image pixels:
   Label the pixels $d_{x,y} = \begin{cases} 1, & \text{if pixel } x,y \text{ is } \textit{black} \\ 0, & \text{if pixel } x,y \text{ is } \textit{white} \end{cases}$
2. Forward pass:
   For each object pixel: $d_{x,y} = \min\left(d_W, d_{NW}, d_N, d_{NE}\right) + 1$
3. Backward pass:
   For each object pixel: $d_{x,y} = \min\left(d_{x,y}, d_E + 1, d_{SE} + 1, d_S + 1, d_{SW} + 1\right)$

Euclidean:

√8	√5	2	√5	√8
√5	√2	1	√2	√5
2	1	0	1	2
√5	√2	1	√2	√5
√8	√5	2	√5	√8

City block:

4	3	2	3	4
3	2	1	2	3
2	1	0	1	2
3	2	1	2	3
4	3	2	3	4

Chessboard:

2	2	2	2	2
2	1	1	1	2
2	1	0	1	2
2	1	1	1	2
2	2	2	2	2

Figure 4.4: Distances from the center pixel.

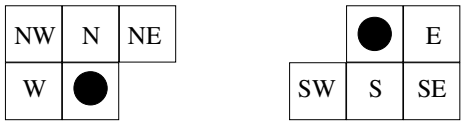


Figure 4.5: The window for forward pass (left), and for backward pass (right).

Original image:

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

After forward pass:

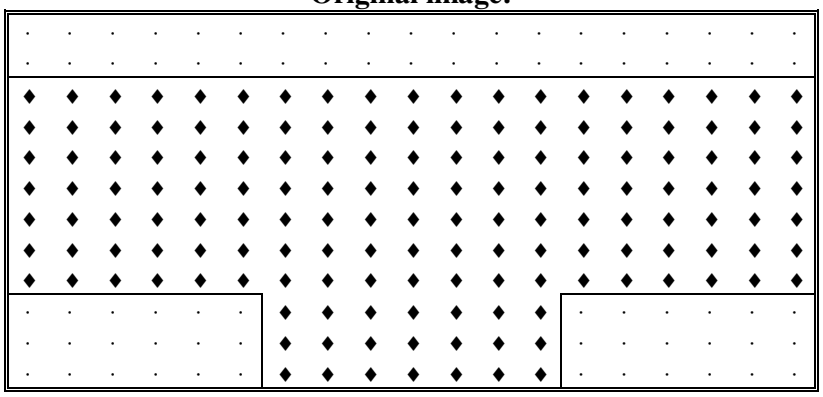
0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	2	2	2	2	1	0
0	1	2	3	3	0	0	0
0	1	2	3	1	0	0	0
0	0	0	0	0	0	0	0

After backward pass:

0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	2	2	1	1	1	0
0	1	2	2	1	0	0	0
0	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0

Figure 4.6: 8-distance transform: image after forward pass (left);
image after backward pass (right).

Original image:



4-distance transformed image:

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	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	**4**	**4**	**4**	**4**	**4**	4	4	4	4	4	4	4	**4**	**4**	**4**	**4**	**4**	**4**
3	3	3	3	3	3	4	**5**	5	5	5	**5**	4	3	3	3	3	3	3
2	2	2	2	2	2	3	4	5	**6**	5	4	3	2	2	2	2	2	2
1	1	1	1	1	1	2	3	4	5	4	3	2	1	1	1	1	1	1
0	0	0	0	0	0	1	2	3	4	3	2	1	0	0	0	0	0	0
0	0	0	0	0	0	1	2	3	**4**	3	2	1	0	0	0	0	0	0
0	0	0	0	0	0	1	2	3	**4**	3	2	1	0	0	0	0	0	0

8-distance transformed image:

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	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
3	3	3	3	3	3	3	3	3	**4**	3	3	3	3	3	3	3	3	3
2	2	2	2	2	2	2	2	3	**4**	3	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	2	3	**4**	3	2	1	1	1	1	1	1	1
0	0	0	0	0	0	1	2	3	**4**	3	2	1	0	0	0	0	0	0
0	0	0	0	0	0	1	2	3	**4**	3	2	1	0	0	0	0	0	0
0	0	0	0	0	0	1	2	3	**4**	3	2	1	0	0	0	0	0	0

Figure 4.7: Examples of the 4-distance and the 8-distance transforms. Skeletal pixels (see Section 4.3) have been superimposed.

4.3 Skeletonizing

Binary image objects can be described by a unit width skeleton. The skeleton is placed in the medial region of the object, has the same topology, and allows the evaluation of the spatial dimensions as well as of the orientation of the object and its subsets. *Skeletonizing* (or *thinning*) usually consists of two main steps:

1. Distance transform
2. Detection of the skeletal points

The first stage is usually either the 4-distance or the 8-distance transform. A simple algorithm for detecting the skeletal points operates the 4-distance transform image in row-major order applying a local window operation. For each pixel, its 8 nearest neighbors are examined. If the pixel is a local maximum, i.e. the distance value of the pixel is greater than or equal to all of its neighbors, it is detected as a skeletal point. The algorithm, however, does not preserve continuity of the skeleton, see Figure 4.8. Therefore additional processing is needed to restore the continuity. Another approach for the skeletal detection is given by Arcelli and Baja:

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	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**	**4**
3	3	3	3	3	3	3	3	3	**4**	3	3	3	3	3	3	3	3	3
2	2	2	2	2	2	2	2	2	3	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	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	**1**	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

Figure 4.8: Skeletal pixels found by the simple thinning algorithm. Underlined pixels should also be included to preserve the continuity of the skeleton.

Arcelli-Baja algorithm

The algorithm operates on 4-distance transformed images. The image pixels are classified to *object pixels* (A) and to *background pixels* ($\overline{A}$). The object pixels are further classified either to *inner pixels* (I), or to *contour pixels* (C). A pixel belongs to C if its distance value equals to unit; the rest of the object pixels with $d_4 \geq 2$ belong to I. A contour pixel is defined as *multiple* if at least one of the two *multiplicity conditions* is satisfied in the 3×3 neighborhood:

1. Neither the N-S (north-south), nor the W-E (west-east) neighbor pairs are such that one belongs to I and another to $\overline{A}$.

2. Any of the triples in the set { N-NE-E, E-SE,S, S-SW-W, W-NW-N } are such that the diagonal neighbor belongs to C, while the remaining two belong to $\overline{A}$.

See Figure 4.9 for the classification of the pixels. Contour pixels satisfying one of the multiplicity conditions are underlined.

The algorithm starts by examining each of the contour pixels (C). The pixels are checked against the multiplicity conditions, and all the pixels found to be multiple are marked as skeletal pixels. The rest of the contour pixels are then removed from the contour (C) to the set of background pixels ($\overline{A}$). The skeletal pixels just found remain in the class of contour pixels.

The algorithm is then repeated to the *2nd layer* of the pixels, having d_4=2. These pixels are moved from I to C. At the kth step of the algorithm, pixels of the kth layer are considered. See Figure 4.10 for the 4th step of the iterated process, where the bold labels constitute the pixels of the current layer. Circled 3's represent pixels found to be skeletal pixels in the previous step of the algorithm. The circled 4's indicate the currently identified multiple pixels. The algorithm is iterated, contour after contour, until on the current contour all the pixels are multiple. The multiple pixels together constitute the skeleton of the object.

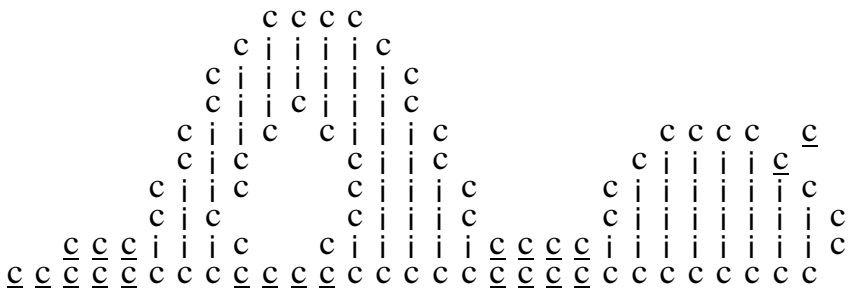


Figure 4.9: Classification of the pixels.

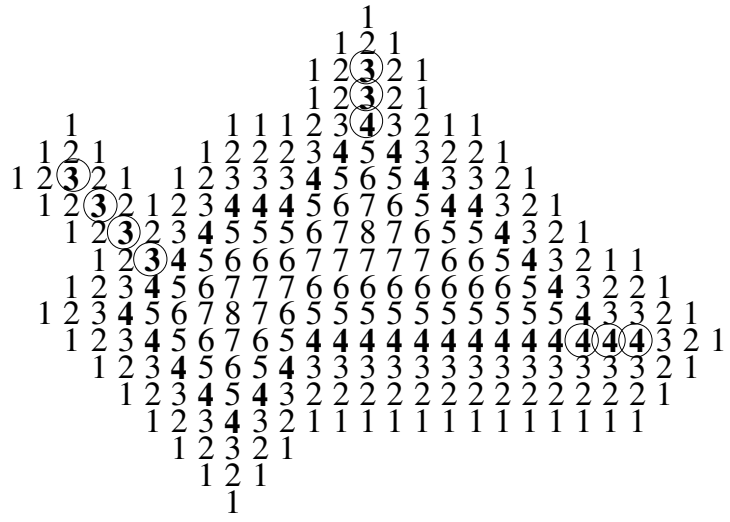


Figure 4.10: Arcelli-Baja algorithm in the 4th phase.

4.4 Component labeling

Component labeling is an operation where each 8-connected object in the image is assigned its own label. The labeling can be performed by a forward pass over the image, in which only the object pixels are considered. The four neighboring pixels within a local window (see Figure 4.5) are examined. If any of the neighboring pixels have been labeled earlier, the same label is then assigned to the current pixel. Otherwise a new label is created. When two neighboring pixels have different labels (two branches are meeting), the pixels with these labels should be combined later. This is the case with the labels 1 and 3 of the example in Figure 4.11.

-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	-	-	2	2	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	3	3	-	-	1	1	-	-	2	2	-	-
-	4	4	4	4	4	4	4	4	-	-	-	-	3	3	-	-	1	1	-	-	2	2	-	-
-	4	4	-	-	-	-	4	4	-	-	-	-	3	3	3	3	3	3	3	3	3	3	-	-
-	4	4	-	5	5	-	4	4	-	-	-	-	3	3	3	3	3	3	3	3	3	3	-	-
-	4	4	-	5	5	-	4	4	-	-	-	-	-	-	-	-	3	3	-	-	3	3	-	-
-	4	4	-	5	5	-	4	4	-	-	-	-	-	-	-	-	3	3	-	-	3	3	-	-
-	4	4	-	-	-	-	4	4	-	-	-	-	-	-	-	-	3	3	-	-	3	3	-	-
-	4	4	4	4	4	4	4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	6	6	6	6	6	6	6	6	6	6	6	6	6	6	-	-	-	-
-	-	-	-	-	-	-	6	6	6	6	6	6	6	6	6	6	6	6	6	6	-	-	-	-
-	-	-	-	-	-	-	6	6	-	-	-	-	-	-	-	-	-	-	6	6	-	-	-	-
-	-	-	-	-	-	-	6	6	-	-	-	-	-	-	-	-	-	-	6	6	-	-	-	-
-	-	-	-	-	-	-	6	6	-	-	-	-	7	7	7	-	-	-	6	6	-	-	-	-
-	-	-	-	-	-	-	6	6	-	-	-	-	7	7	7	-	-	-	6	6	-	-	-	-
-	-	-	-	-	-	-	6	6	-	-	-	-	-	-	-	-	-	-	6	6	-	-	-	-
-	-	-	-	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	-	-	-	-
-	-	-	-	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	-	-	-	-
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 4.11: The principle of component labeling.

4.5 Halftoning

Halftoning (or *binarization*) represents a class of methods for converting gray-scale images to binary, black and white images. Binarization can be trivially performed by global thresholding, however, it is usually desired that the gray-scale information is retained in some way.

Rasterization

Rasterization is widely used in newspaper technology in the printing phase, where the digital image is transferred back to analog form. The gray-scale pixels are reproduced as different

sized black dots whose diameter depends on the brightness of the pixel. The darker the pixel, the larger the dot and vice versa, see Figure 4.12 for an example.

Figure 4.12: Rasterized gray-scale values from white to black.

Dithering

Dither coding (or *dithering*) methods convert gray-scale images to binary images while trying to retain the average grayness of the image without increasing the resolution. The main idea of dithering is to apply a variable threshold in the binarization, see Figure 4.13. The two most common dithering methods are *ordered dithering* and *error diffusion*.

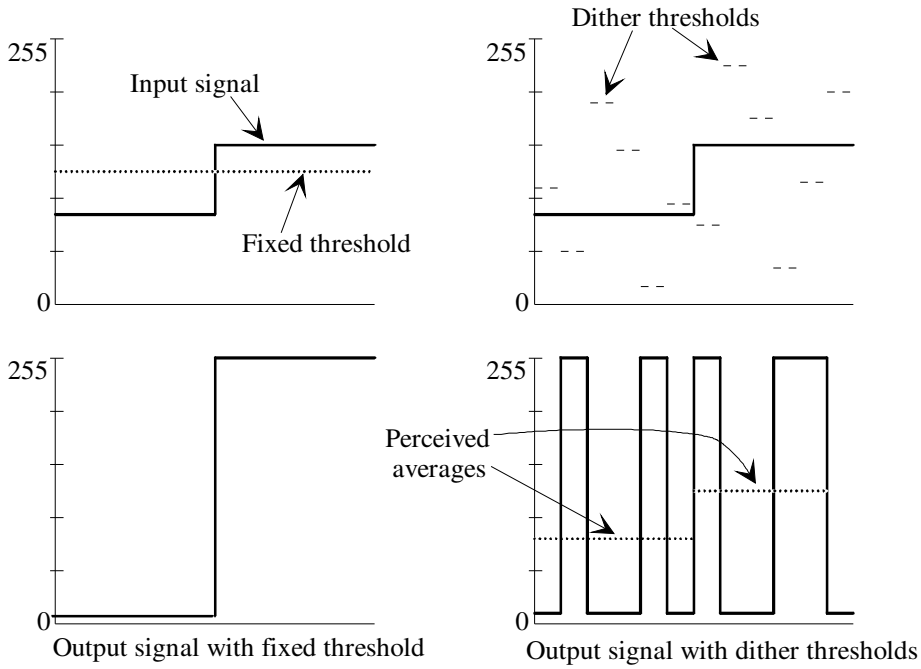


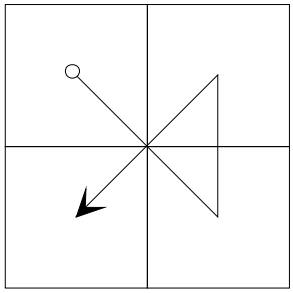
Figure 4.13: The principle of dithering.

Ordered dither

In ordered dithering the image is first divided into $n \times n$ blocks. Each block is then separately processed. The thresholds of the pixels are given by the pseudo-random generator of Figure 4.14. If the pixel values are in the range $[0, x_{max}]$, the threshold value $th_{i,j}$ is calculated as:

$$th_{i,j} = \frac{x_{\max} + 1}{n^2} \cdot \left(k_{i,j} + 0.5 \right) \tag{4.4}$$

where n is the size of the dither matrix, and $k_{i,j}$ is the corresponding pseudo-random value given by the matrix. Consider a block of pixels with constant gray-value of 4 (in the scale [0, 15]). The resulting dithered block consists of 4 white and 12 black pixels having the average gray value of 4, see Figure 4.15. The advantage of the ordered dither is that it can be applied in parallel, for each pixel at the same.



0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

Figure 4.14: General orientation of the ordered dither matrix (left);
Threshold matrix of 4×4 ordered dithering (right).

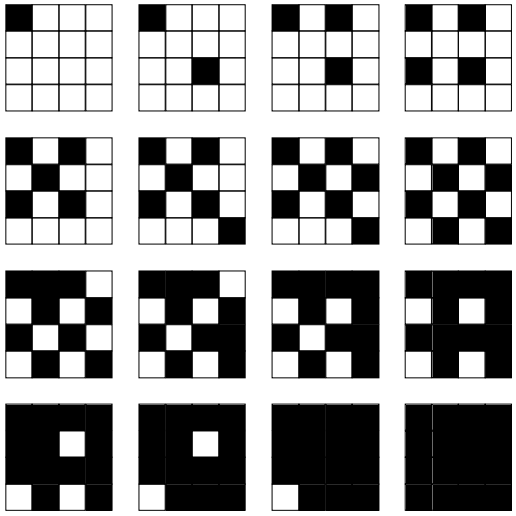


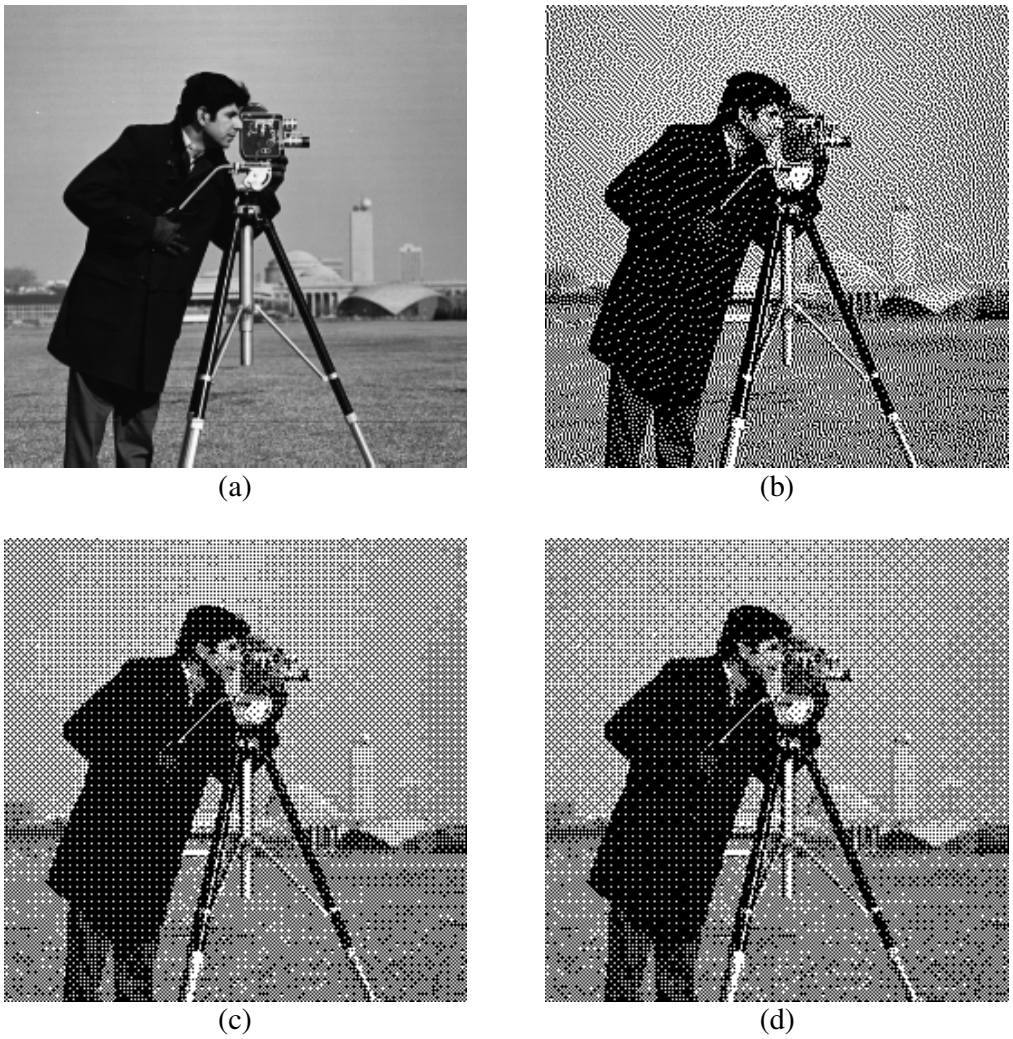
Figure 4.15: Resulting patterns of ordered dithering for blocks with constant gray-level values.

Error diffusion

Error diffusion processes the image in row-major order (top-down and left-to-right). Each pixel is examined and rounded to black or white. The error of the binarization is then compensated for the following pixels. For example, if a pixel value 191 is rounded up to 255, the image is then 64 units too bright. This is compensated by making the neighboring gray pixels a little darker (64 units darker in total) so that the sum of all the pixels' gray values is unchanged. Specifically, the pixels neighboring to east, south, southeast and southwest are adjusted. The error is not evenly distributed to these pixels, but a *weighting mask* is used. The best known error diffusion method, *Floyd-Steinberg dithering*, applies the weighting mask given in Figure 4.16.

	●	7/16
3/16	5/16	1/16

Figure 4.16: Weighting mask of the Floyd-Steinberg error diffusion method.



(a) (b)

(c) (d)

Figure 4.17: Example of dithering: (a) Original image; (b) Floyd-Steinberg dither; (c) Ordered dither with 4×4 matrix; (d) Ordered dither with 8×8 matrix.

4.6 Morphological operations

Mathematical morphology offers strong tools in the representation and description of region shape, such as boundaries and skeletons. We are also interested in morphological techniques for pre- and post-processing, such as morphological filtering, thinning, and pruning. The basic

operations in morphology are *dilation* and *erosion*: In dilation and erosion the image A is filtered by a local window B called *structuring element*, see Figure 4.18.

Dilation:

$$A \oplus B = \{a + b : a \in A, b \in B\} = \bigcup_{b \in B} A_b \tag{4.5}$$

Erosion:

$$A \ominus B = \left(A^c \oplus B\right)^c = \bigcap_{b \in B} A_b \tag{4.6}$$

In dilation, the *origin* of the structuring element B is placed on the top of each object pixel (black pixel) of the image A. Then each neighboring pixel within the local neighborhood (defined by the structuring element) is checked. Any white pixel found within the window are included in the object, i.e. changed to black, see Figure 4.19. The image object is enlarged as the result of dilation.

In erosion, the image is processed by the structuring element in similar manner. Now, if any neighboring pixel is white, the pixel in the origin is removed from the object, i.e. changed to white, see Figure 4.19. The resulting object after erosion is a subset of the original object. Note, that dilation and erosion are not dual operations. In fact, if the two operations are applied one after another with the same structuring element, two important morphological filtering operations, *opening* and *closing*, are obtained:

Opening:

$$A \circ B = (A \ominus B) \oplus B \tag{4.7}$$

Closing:

$$A \bullet B = (A \oplus B) \ominus B \tag{4.8}$$

The opening operation satisfies the following properties:

1. $A \circ B$ is a subset (subimage) of A.
2. If C is subset of D, then $C \circ B$ is a subset (subimage) of $D \circ B$.
3. $(A \circ B) \circ B = (A \circ B)$.

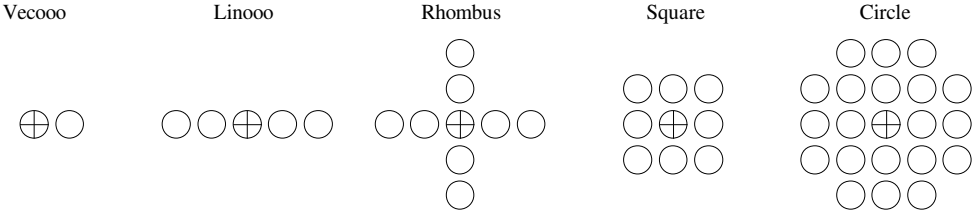


Figure 4.18: Example of general structuring elements (origin marked with +).

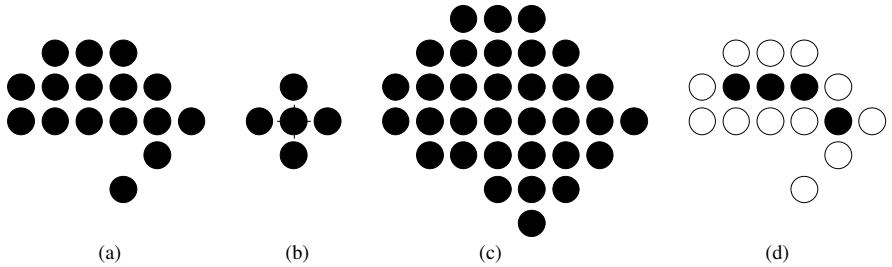


Figure 4.19: Example of dilation and erosion: (a) original object; (b) structuring element; (c) result of dilation; (d) result of erosion.

The closing operation satisfies the following properties:

1. A is a subset (subimage) of $A \bullet B$.
2. If C is subset of D, then $C \bullet B$ is a subset (subimage) of $D \bullet B$.
3. $(A \bullet B) \bullet B = (A \bullet B)$.

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 difference between A and its erosion:

$$\beta(A) = A - (A \ominus B) \tag{4.9}$$

Figure 4.20 illustrates the mechanics of boundary extraction.

Extraction of connected components

Extraction of connected components in a binary image is central to many automated image analysis applications. Consider A as an 8-connected object, and assume that a point p of A is known. Then, the following iterative expression yields all the elements of A:

$$X_k = (X_{k-1} \oplus B) \bigcap A \qquad k = 1, 2, 3, \ldots \tag{4.11}$$

where $X_0=p$, and B is the structuring element shown in Figure 4.20b. The algorithm terminates at iteration step k if $X_k=X_{k-1}$. The only difference compared to the region filling algorithm, is the use of A instead of its complement A^c.

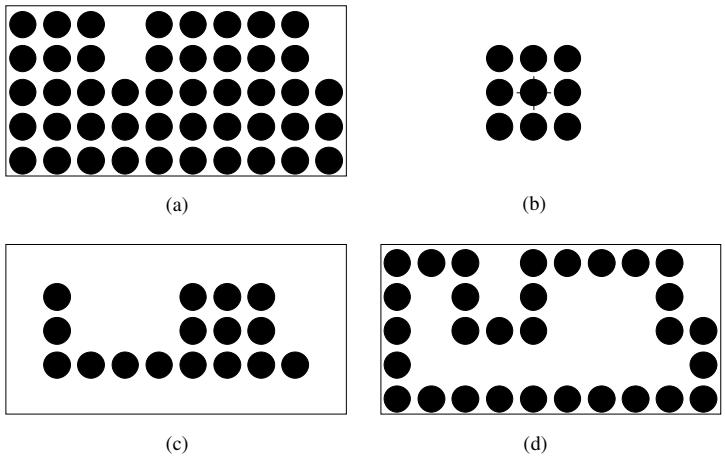


Figure 4.20: Boundary extraction: (a) original object; (b) structuring element; (c) A eroded by B; (d) the result of the boundary extraction.

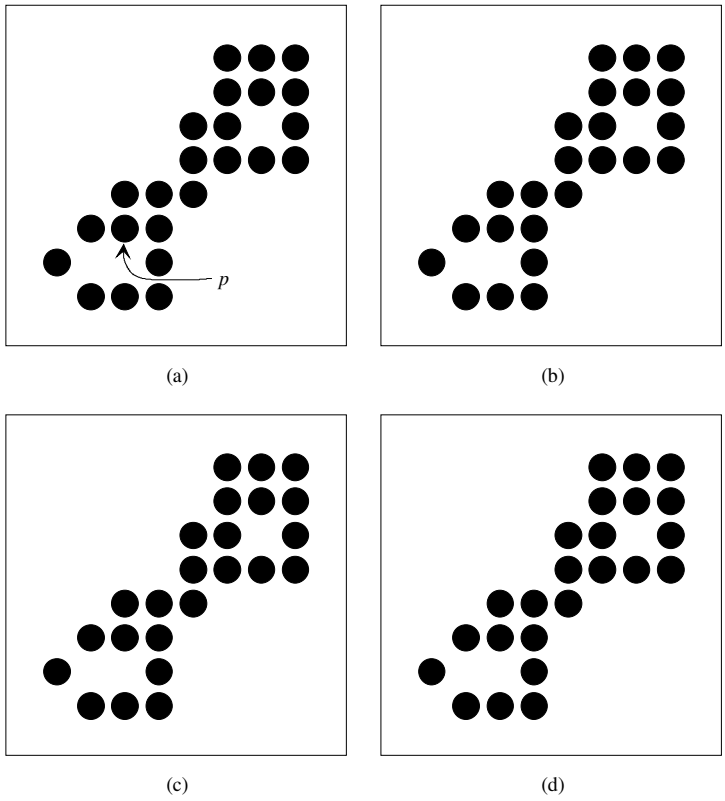


Figure 4.21: Extraction of connected components: (a) original object; (b) result of the first iteration step; (c) result of the second iteration step; (d) final result.

Region filling

Consider A as an 8-connected boundary of an object. The object region can be filled by a set of dilations, complementation, and intersections. Beginning with a point p inside the boundary, the object can be filled by the following procedure:

$$X_k = \left(X_{k-1} \oplus B\right) \bigcap A^c \qquad k = 1, 2, 3, \ldots \tag{4.10}$$

where $X_0=p$, and B is the symmetric structuring element shown in Figure 4.22(c). 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, see Figure 4.22.

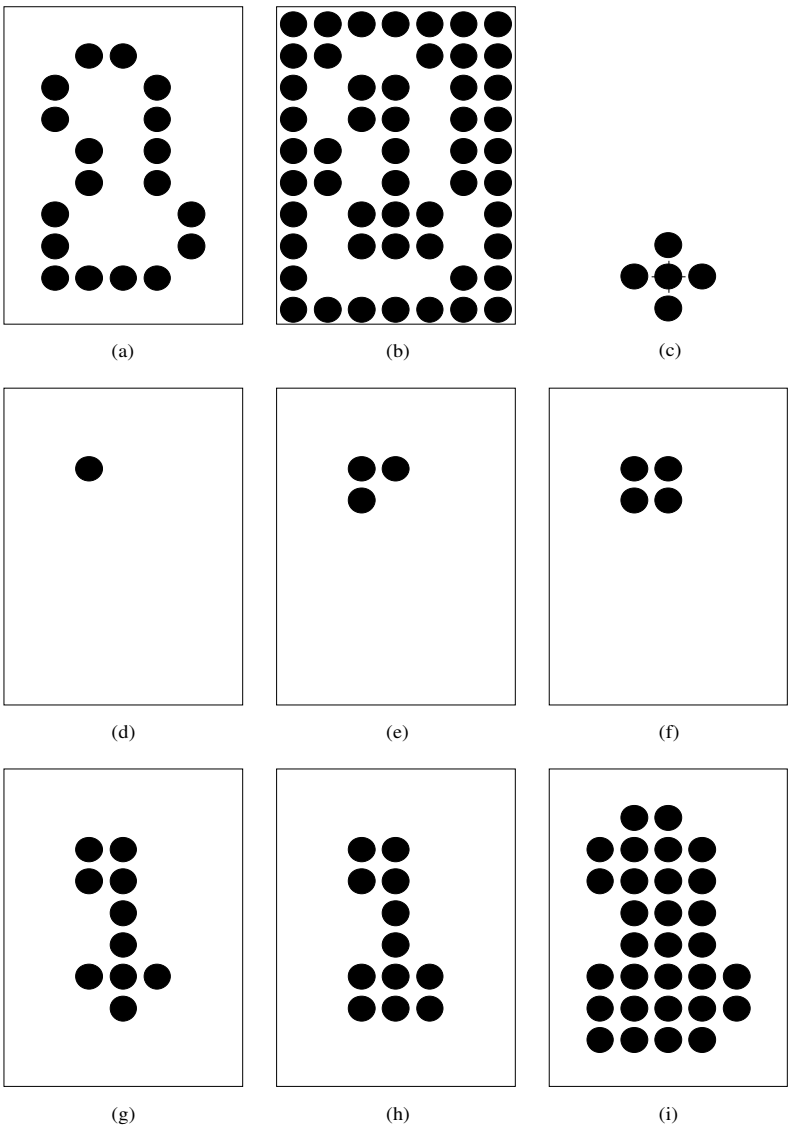


Figure 4.22: Region filling: (a) boundary of an object; (b) complement of the boundary; (c) structuring element; (d) initial point inside the boundary; (e)-(h) various steps of the algorithm; (i) final result, obtained by forming the set union of (a) and (h).

5 Color image processing

There is nothing much new in color image processing compared to gray-scale images. The same algorithms (histogram equalization, filtering, etc) can be separately applied to the color components of the RGB image, or to the components of whatever color model is in use. The processing can be directed to all of the components, or only one or two of them. The processing doesn't necessarily need to be equivalent to all of the components. In image compression, for example, the components of the YUV model are often treated so that the chrominance components U and V are represented less accurately than the luminance component Y.

5.1 Color quantization of images

An important application of color image quantization is the conversion of true color images to color-mapped images, see Section 1.3. Here the RGB color space is quantized to K colors. Gray-scale pixels are often quantized by using very simple algorithms, e.g. cutting off the least significant bits. The color images, on the other hand, consist of three components, and if these were separately quantized the result would not be satisfactory. A joint quantization of the color components can be performed by using *vector quantization* (VQ). The VQ algorithms will be discussed more detailed in the second part (Image Compression) of this course, since vector quantization has a major role in image compression. Meanwhile, let us present three basic techniques for color quantization.

The popularity algorithm:

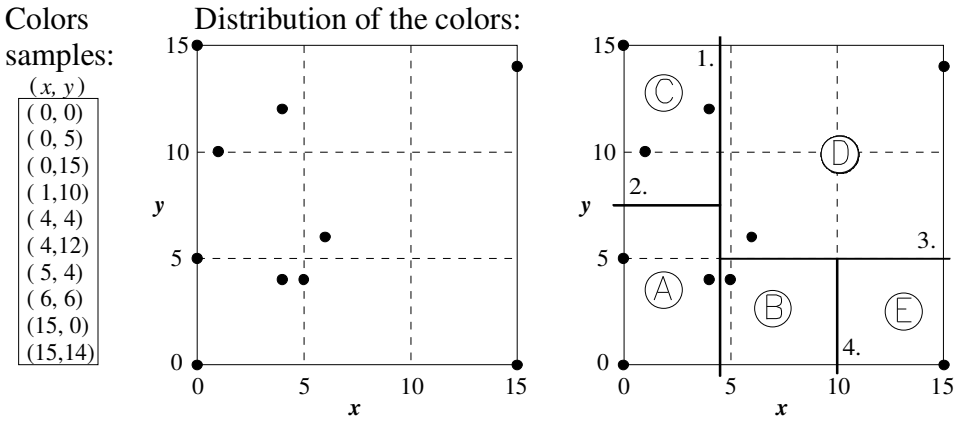
The assumption of this algorithm is that the color-map can be made by finding the most dense regions in the color distribution of the original image. The *popularity algorithm* simply chooses the K colors from the histogram with the highest frequencies, and uses these for the color-map. This can be done with a simple *selection sort*. The algorithm works well with most of the image, but performs poorly on images with a wide range of colors.

The median cut algorithm:

The median cut algorithm splits the color space iteratively into rectangular sub regions until it consists of K regions. It begins with only one region which is the full color space. At the i'th phase, the color space consist of i-1 regions. The regions are "shrunk" to fit tightly around the points (colors) it encloses, by finding minimum and maximum values of each of the color coordinates. The enclosed points are sorted along the longest dimension of the region. The region with the maximum dimension is then split into two halves at the medial point. Approximately equal numbers of points will fall on each side of the cutting plane. The step is recursively applied until K regions are generated. The representative for each box is computed by averaging the colors contained in each.

See Figure 5.1 for an example of the median cut algorithm. For simplicity let us consider a two color system (x, y) with the range of [0, 15], thus having 256 colors at maximum. Suppose that we have ten color samples of the image, and that the size of the color palette is predefined to five. At the first phase the color space is split at x=4.5. At the second phase the region A has the maximum dimension and is therefore split at y=7.5. At this moment we have three regions

consisting of five (region *A*), three (*B*), and two (*C*) color samples. At the next phase, the region *B* is split at *y*=7. Finally, the region A is further split at *x*=3. The resulting five regions are then assigned by the average colors of the samples in each region. The colors of the regions (*A*, *B*, *C*, *D*, *E*) are [(1,3), (5,4), (2,12), (11,10), (15,0)].



Stage:	Regions:	Maximum dimension:
Initial:	A=[0..15, 0..15]	**0..15**
1.	A=[0..4, 0..15]	**0..15**
	B=[5..15, 0..14]	0..14
2.	A=[0..4, 0..5]	0..5
	B=[5..15, 0..14]	**0..14**
	C=[0..4, 10..15]	10..15
3.	A=[0..4, 0..5]	0..5
	B=[5..15, 0..4]	**5..15**
	C=[0..4, 10..15]	10..15
	D=[6..15, 6..14]	6..15
4.	A=[0..4, 0..5]	0..5
	B=[5..5, 4..4]	5..5
	C=[0..4, 10..15]	10..15
	D=[6..15, 6..14]	6..15
	E=[15..15, 0..0]	15..15

Final color palette:

(1, 3)
(6, 4)
(2,12)
(11,10)
(15, 0)

Figure 5.1: Example of the median cut algorithm.

Heckbert quantization:

Heckbert quantization consists of three phases:

- Calculate the color histogram.
- Generate the color-map.
- Quantize the image using error diffusion technique.

Consider 24-bits RGB images consisting of over 16 million colors. A prequantization is performed in the first phase to preserve memory; each color component is quantized from 8 to 5 bits resulting in total 2^{15} = 32768 colors. This prequantization is done for the histogram only; the original image is still untouched.

At the second phase, an initial color-map is first generated by the median cut algorithm. The resulting color palette is then improved by an iterative algorithm known as the *Generalized Lloyd Algorithm* (GLA). Each iteration phase consists of two steps:

• Rearrange the color space regions by mapping each of the color pixel x to its nearest palette color y using the euclidean distance:

$$d(x,y) = \sqrt{\left(r_x - r_y\right)^2 + \left(g_x - g_y\right)^2 + \left(b_x - b_y\right)^2} \qquad (5.1)$$

• Calculate the *centroid* y' of each region (taking the average value of each color component). Then replace the palette color y by the centroid y'.

If the quality of the color palette is measured by *total square error* (TSE) it can be shown that the palette after each iteration is always better than or equal to the one before the iteration.

$$\mathrm{TSE} = \sum\left[\left(r_x - r_y\right)^2 + \left(g_x - g_y\right)^2 + \left(b_x - b_y\right)^2\right] \qquad (5.2)$$

The iterations are thus continued until no change is achieved. The algorithm doesn't necessarily reach the global optimum, but converges to a local minimum. An example of the iteration algorithm is given in Figure 5.2 starting with the color pixels given by the median cut algorithm. In the final phase, the image is quantized by the error diffusion technique.

5.2 Pseudo-coloring

Consider the image in Figure 5.3 which has been produced by an atom-force microscope. The sample pixels do not represent the reflectance of visible light of any wavelength, but the surface height of the object under examination. The sample image thus has no natural colors assigned to the pixels. It is, however, sometimes useful to enhance gray-scale images by using *pseudo-colors*. The eye finds it considerably more difficult to identify gradual changes in brightness compared to gradual changes in color. Here the pixels are artificially colored by a gray-scale→ RGB color-mapping. Let us consider three color mappings:

Color mapping:	**Calculation of the mapping:**
A - "From brown, via red and yellow, to white"	$r = \min[3\cdot x, 255]$ $g = \min[3\cdot(x\text{-}85), 255]$ $b = \min[3\cdot(x\text{-}171), 255]$
B - "Green"	$r = \min[3\cdot(x\text{-}171), 255]$ $g = x$ $b = 63$
C - "From red to blue"	$r = \min[1.5\cdot(x\text{-}85), 255]$ $g = 63$ $b = 255\text{-}\min[(1.5\cdot x), 255]$

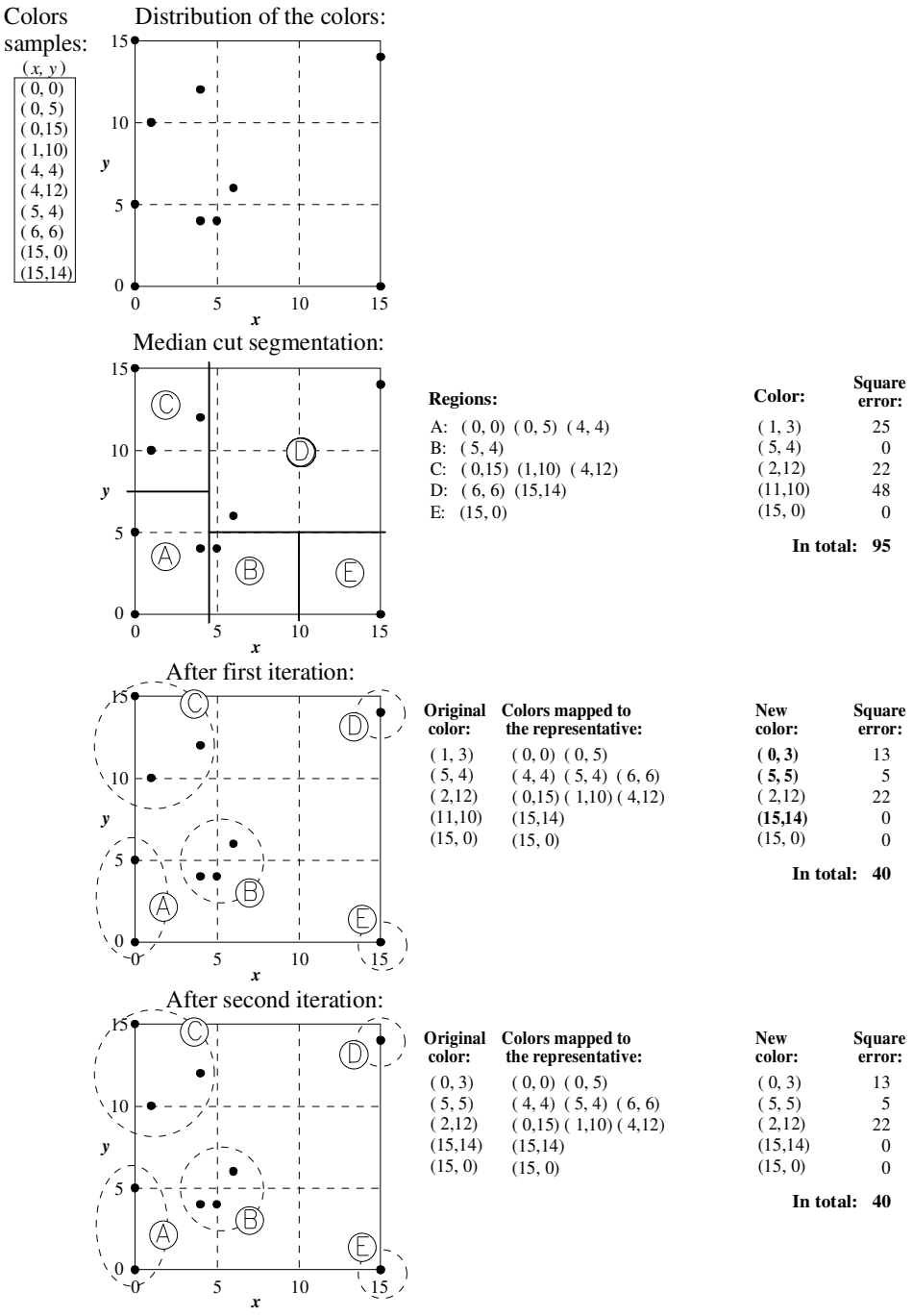


Figure 5.2: Example of the Heckbert quantization Lloyd algorithm.

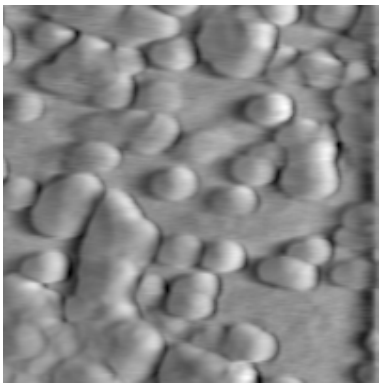


Figure 5.3: Image taken by an atom-force microscope (200×200×8).

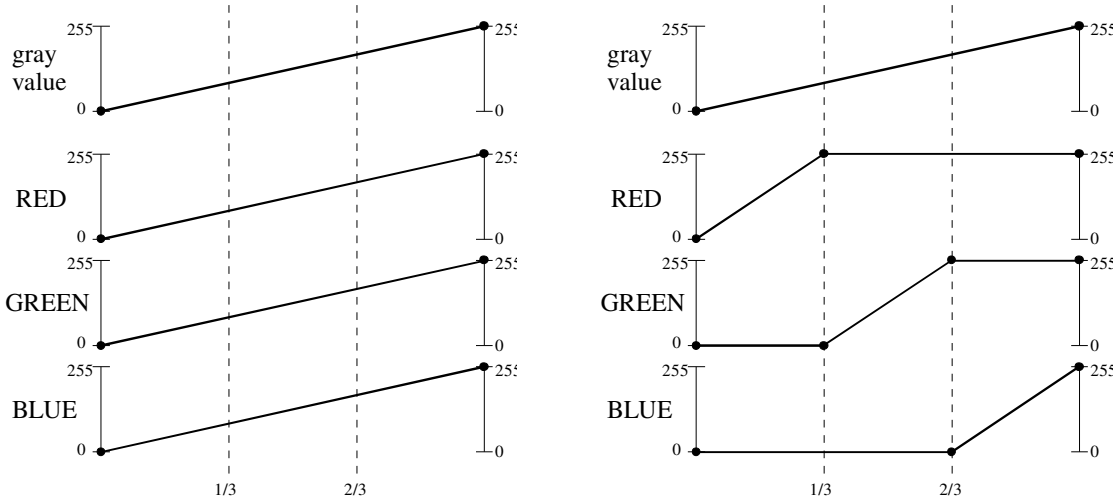


Figure 5.4: RGB-representation of the gray-scale (left); The color mapping A (right).

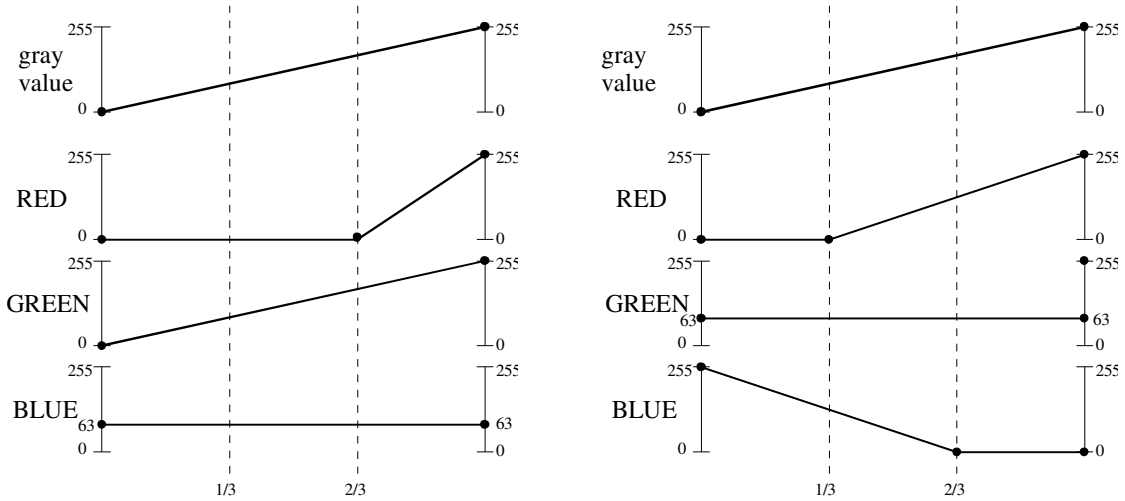


Figure 5.5: The color mapping B (left); the color mapping C (right).

6 Applications

Let us next have a brief look at some applications that are based on various combinations of the basic image processing operations. The following applications will be presented here:

- Raster to vector conversion
- Optical character recognition
- Cell particle motion analysis
- Mammography calcification detection

6.1 Raster to vector conversion

There are images (e.g. *engineering drawings*, *cartographic maps*, *schemes*) that are much more convenient to store in vector graphics format rather than in raster image format. Vector graphics have the following advantages over raster images:

- They can be easily edited by an appropriate drawing software.
- The quality of the image is independent of the scaling.

Despite these facts there are still drawings of these kind stored as raster images. They might have been transmitted through facsimile, printed and then digitized by optical scanner, or they are simply old hand-made engineering drawings prior the CAD era. There is a great need to convert these images to vector format, which however, is not an easy process. Let us next have a look at what kind of operations the conversion involves. The process of raster to vector conversion consists of the following phases:

1. Binarization: The original gray-scale image is first converted to a binary image by *local thresholding*, see Section 3.6.
2. Cleaning: The binary image is then "cleaned" by using *binary filtering* for strengthening weak thin lines.
3. Skeletonizing: The cleaned binary image is skeletonized by first performing *distance transform* and then applying the *Arcelli-Baja* algorithm, see Section 4.3.
4. Width-labeling: The skeleton image is transformed to so-called *width-labeled image* by a local window operation.
5. Pruning and cleaning: For deleting the false branches derived from the noise and cleaning the true branches *pruning* and *cleaning* are applied.
6. A piecewise linear segmentation: The feature points (such as *intersections*, *end-skels*, and *knot-cells*) of the skeleton are labeled, and finally a vector representation of the image is created.

6.2 Optical character recognition

A great number of black and white raster images, especially those transmitted via facsimile, consist mostly of typed or typeset text. These *textual images* are often desired to be converted back to some text format. The first part of the "image to text" conversion is to isolate and

extract the letters and marks from the image. At the second phase, these marks are matched against some predefined symbol library (e.g. ASCII figures of some font), or the library can be constructed on the basis of the marks found in the image.

Extracting symbols:

The image is scanned in row-major order, from left to right, top to down. The first nonwhite pixel is used as a seed to extract a symbol. Once a seed pixel is found, a boundary tracing algorithm is applied to locate the symbol, see Section 4.1. The extracted mark is then identified by *template matching*, and then removed from the image. The next symbol is located in the same manner. The location information (coordinates) of the symbols may be used to determine their order in the text file. Detailed description and more sophisticated algorithms for the symbol extraction is not covered here.

Template matching:

The matching procedure generally operates by examining an *error map*, which is the bitwise exclusive-OR between the new symbol and a library member. Before calculating the error map, the two symbols must be aligned appropriately in respect to each other. One way is to compute the *centroids* of the symbols (the average position of the black pixels in the bitmap) and align them. It is also possible to align the bitmaps in more than one position. For example, one might perform nine matches, corresponding to the centroids plus constant one-pixel displacements in the eight principal directions.

The starting point for template matching is to measure the difference between the two symbols by the total number of pixels that are set in the error map. However, it is best to weight the error pixels differently depending on the context in which the error occurs. Each error pixel contributes an amount that equals the number of error pixels in its 3×3 neighborhood. Consider the example in Figure 6.1. The total number of error pixels in the correct match of 'e' is 29, despite the number of error pixels in the mismatch of 'c'≠'o' is only 23. The corresponding total weighted errors are 75 for the match 'e', and 132 for the 'c'≠'o'.

A match is rejected if the weighted error exceed a predefined threshold. In order to make the scheme size-independent, the threshold should depend on the number of black pixels in the symbols, and may be chosen to be a linear function of it. The best way to set the threshold is to train the system on the fonts being used. A more sophisticated approach, which has been found to work better in practice, is to make the threshold dependent on the symbols perimeter length rather than on the black pixel count.

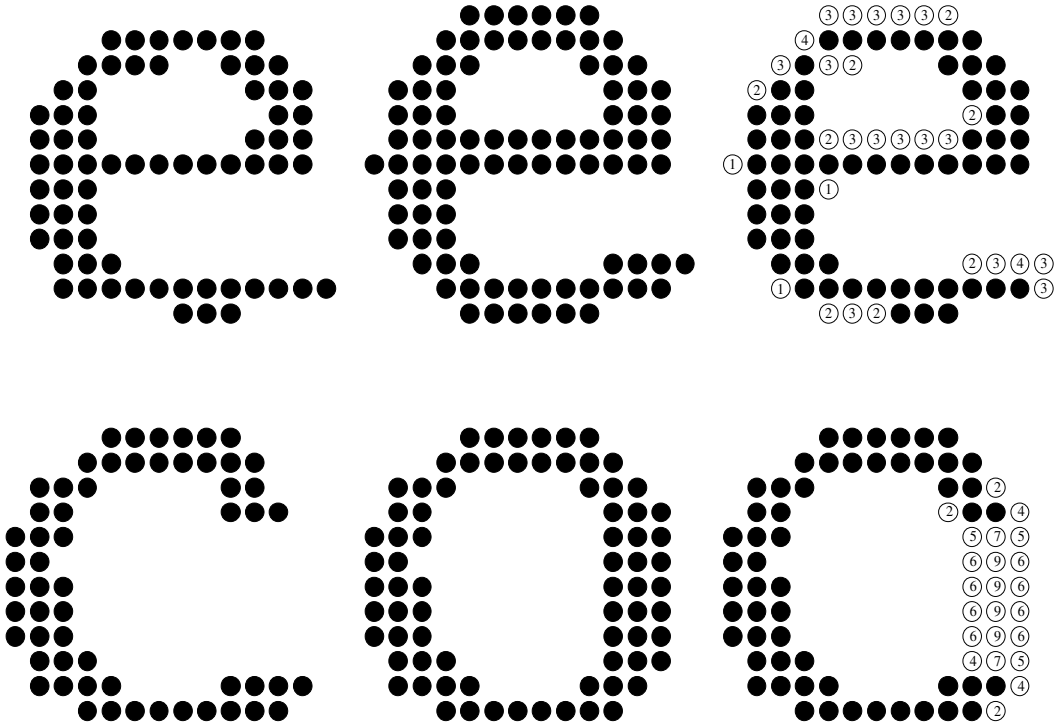


Figure 6.1: Template matching using the error map.

The template matching can be improved by differentiating the two error maps, see Figure 6.2. The first one (*A*-*B*) contains pixels that are set in the first image but not in the second. The second one (*B*-*A*) contains pixels that are set in the second image but not in the first. The weighted errors are then summed over the whole error map. The result is that, for example, a line that is displaced by a single pixel carries a lighter penalty than it would in the original weighting scheme. In Figure 6.2 the sum of the weight is reduced from 131 to 93.

The number of comparisons made against the library symbols can be reduced to gain speed. There is no need to match templates that are obviously dissimilar to the input. For example, templates that differ greatly in width or height are obviously dissimilar. The preselection, or *screening*, of what library symbols will be considered, it can be based on the symbols' perimeter, the number of vertical white runs that are enclosed in the pattern, or the overall spatial distribution of mass.

The perimeter is already correlated with width and height, but it is also sensitive to noise. The other approach is to compare the number of vertical white runs that are enclosed in the pattern. For example, letter 'I' would have none in either direction, but letter 'H' would have none in vertical direction but several in horizontal direction. The overall spatial distribution of mass can be performed by first calculating the centroid of the symbols, and then dividing them into quadrants around the centroid. For each quadrant, a local centroid is calculated, and for each of the four local centroids the distance between its position in the two symbols are determined. Finally, these four distances are averaged.

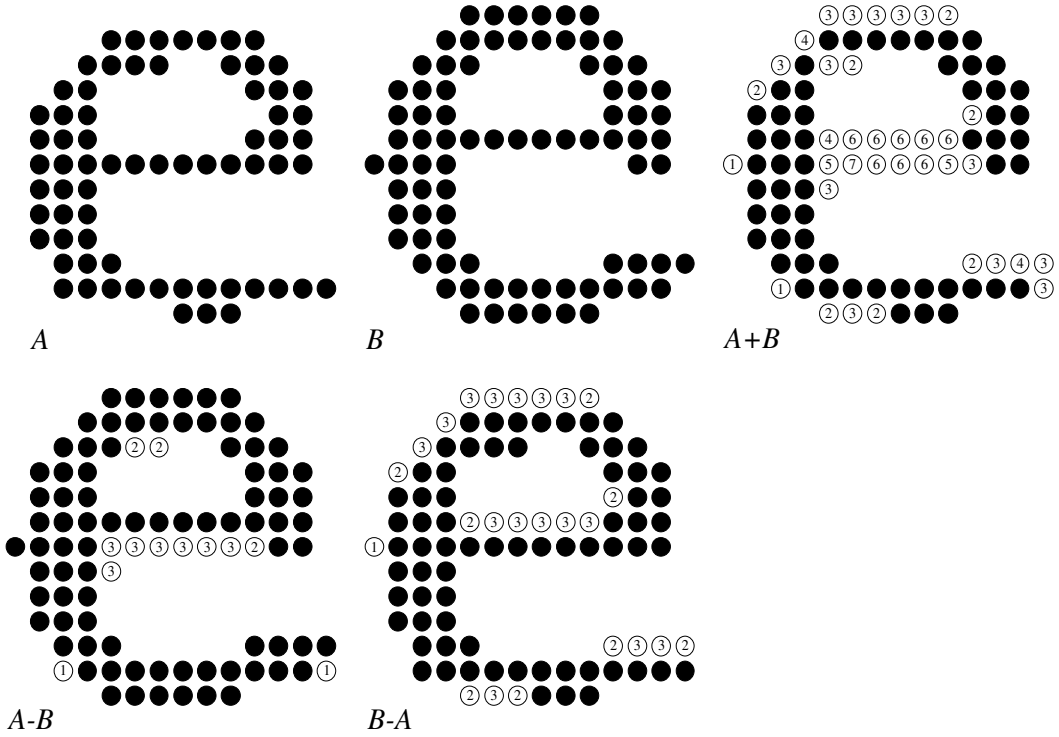


Figure 6.2: Improved template matching by differentiating the errors.

6.3 Cell particle motion analysis

The Computer Science department at the University of Turku is involved in a research project that studies how the *RNA-information* is transferred within cells. A sample cell is given in Figure 6.3. The interesting particles of the cell are the *euchromatin*, *chromatoid*, and *golgi complex*. According to a theory, the euchromatin produces and codes the RNA, the chromatoid transfers it, and the colgi complex is the place where the RNA is stored. One way to verify the theory is to formulate the movement (the speed and places where the particle visits) and to detect the material it is carrying while moving.

The main goal of the research project is to develop a system which is capable of detecting the movement to a certain point, after which, if the detection seems to be impossible, the control is transferred to the user. The system is therefore going to be an user controlled expert system consisting of the following phases:

- Ask the starting point (a sample pixel within the particle).
- Extract the object by region growing technique.
- Parse the object by filling holes inside the object, and by smoothing the boundaries.
- Calculate statistical information of the object for further analysis.
- Analyze the next frame of the image sequence.

The analysis starts with the first image of the digitized image sequence. First the user points at the interesting cell, which is then extracted from the image by region growing technique. The

initial pixel is the one under the mouse pointer. The algorithm considers each of the neighboring pixels and includes them into the object if they meet the uniform criterion, which is standard deviation of the 3×3 neighborhood of the candidate pixel. The region growing continues until no new pixel is found.

In the case of the first image, the threshold value of the growing rule should be manually given so that the extracted object correspond to the real particle as much as possible. If the initial pixel of the algorithm happens to be at a local peak where the standard deviation is high, the growth will stop before it has even been started. Therefore, at the early stage of region growing, the algorithm considers pixels from wider area than just the 8 nearest neighbors. For example, all the pixels within a 5×5 window could be checked.

After the object has been found, it is parsed. All the image pixels are processed by a local 3×3 window. A pixel will remain as an object pixel if the number of object pixels in the 3×3 neighborhood exceeds a predefined threshold value, say 3. Background pixels will also be included to the object if the same condition is met, resulting that possible holes inside the object is filled.

The next frame is then processed on the basis of the information given by the previous frame. The starting pixel is the centroid of the object pixels obtained in the previous frame. The standard deviation threshold of the growing rule is taken as the overall standard deviation of the previous frame. The process will continue until the end of the image sequence, or until the algorithm has lost the object and the user will stop the process.

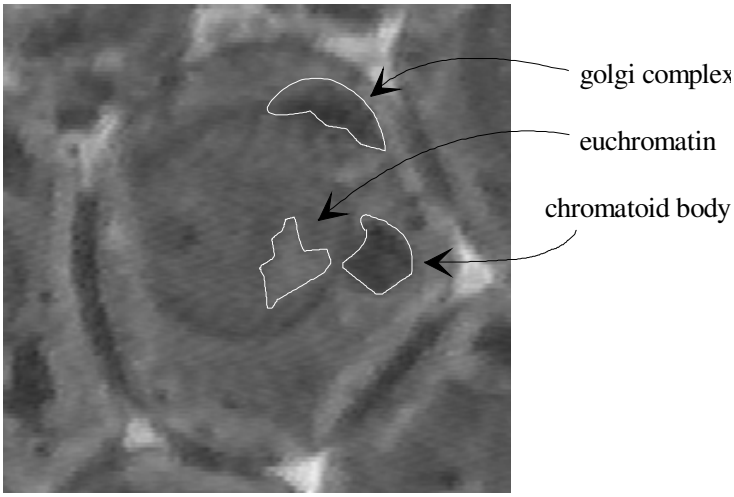


Figure 6.3: Enlargement of the cell where the interesting objects are superimposed.

6.4 Mammography calcification detection

Cancer is a number of diseases caused by abnormal growth of cells. The cells grow out of control, producing masses called *tumors*. There are two main types of tumors. *Benign* tumors do not spread, but may interfere with body functions, and may require removal. On the other hand, cells of *malignant* (*cancerous*) tumors break away from the original tumor. They migrate to other parts of the body, where they may form new malignant tumors (*metastasis*).

X-ray studies of breast are called *mammography*. It is now perhaps the most important tool in aiding the diagnosis of breast diseases. It is particularly efficient in detecting tiny lesions before they enlarge enough to cause problems. *Digital mammography* investigates the use of computer in mammography. Computers could for example detect and prompt signs of abnormality, or distinguish between benign and malignant tumors. It is common that digital mammograms are digitized from conventional film images (*secondary digitization*). Then the quality of the images is limited by the quality of the film. However, acquisition of primary digital images is expected to improve image quality and provide digital mammography with the required high resolution.

Clustered micro calcifications are an early sign of breast cancer. However, it is difficult to decide when the calcifications are of benign or malignant origin. The three major problems are

- Micro calcifications are very small
- Micro calcifications may reside in an inhomogeneous background
- The image contrast is low

The detection process should be independent of the background gray scales. This is done by deleting the low-level frequencies. A Gaussian lowpass filter of width σ (see Figure 6.4) is first applied to the image, then the difference between the original image and the filtered one is obtained. The result of a high pass filtered image is:

1	4	1
4	12	4
1	4	1

Figure 6.4: Gaussian filter with σ=0.391.

$$I_1(x,y) = I(x,y) - G_\sigma \cdot I(x,y) \tag{6.1}$$

Here $I(x,y)$ denotes the original, and $I_1(x,y)$ denotes the image resulted by the filtering. Since the micro calcifications are (generally) brighter than the background, the negative part of the image is rounded to zero:

$$I_2(x,y) = \max\left(0, I_1(x,y)\right) \tag{6.2}$$

The approximate size and distance between the spots (micro calcification) are known. Also, the average gray-value within a spot should be significantly larger than average around the spot. The difference of these averages is computed using the difference of a Gaussian operation with a positive kernel of width σ_+ (reflecting the expected spot size) and a Gaussian operation with a negative kernel of width σ_- (reflecting the expected distance to neighboring spots). In

addition, by using convolution kernel weights, this method is adaptive to the local variations of gray values (to be independent of the local noise level):

$$I_4(x,y) = \left(0.8 \cdot G_{\sigma+} - G_{\sigma-}\right) \cdot I_2(x,y) \tag{6.3}$$

A global thresholding is then applied to the image. A pixel is detected if it exceeds a predefined threshold:

$$I_4(x,y) > T \tag{6.4}$$

To estimate the threshold, the standard deviation of the image is first taken, and T is chosen to be 2.5 times this standard deviation. To make the estimate more robust, the standard deviation is the recalculated by considering only image parts exceeding T. The final threshold is 3 times the recalculated global standard deviation of I_4. It was experimentally determined so that no spot was missed by human judgment.

The segmentation procedure detects sports reliably, but the Gaussian filtering results in smooth boundaries of individual spots. However, the size and shape of the micro calcifications should be preserved, since these features are important in later diagnosis. For example, benign tumors are usually smooth and oval in shape, while malignant tumors have irregular boundaries. The shape of the spots is preserved by using morphological segmentation. They are expanded by *conditional thickening*, which basically consists of a sequence of dilation operations (details are omitted here).

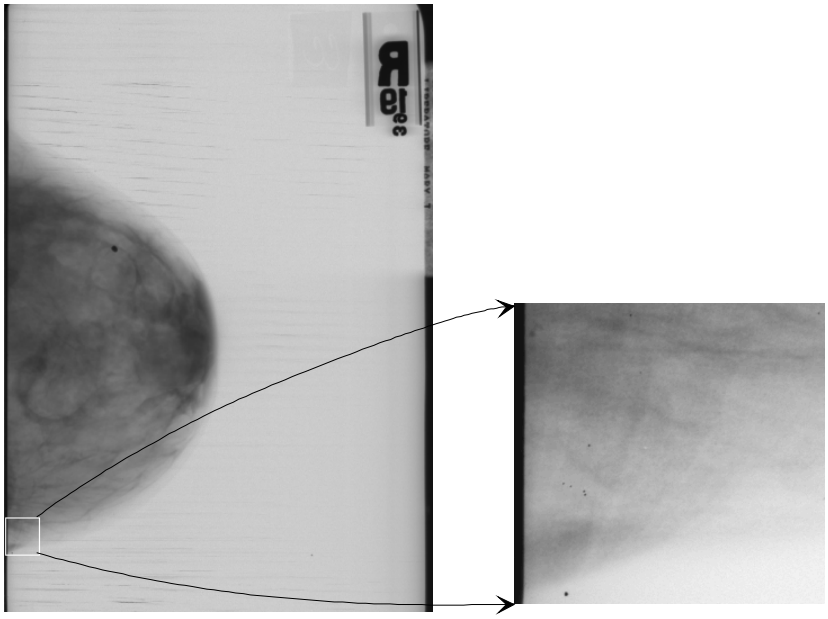


Figure 6.5: Digitized mammogram image (negative).

Literature

Sect. Reference:

* **B. Jähne,** *Digital Image Processing: Concepts, Algorithms, and Scientific Applications*. Springer, 1995.

* **R.E. Gonzalez, R.E. Woods,** *Digital Image Processing*. Addison-Wesley, 1992.

* **A. Low,** *Introductory Computer Vision and Image Processing*. McGraw-Hill, 1991.

* **A.N. Netravali, B.G. Haskell,** *Digital Pictures*. Plenum Press, 1988.

3 **N.R. Pal, S.K. Pal,** A Review on Image Segmentation Techniques. *Pattern Recognition*, Vol. 26 (9), pp.1277-1294, September 1993.

3.1 **H. Radha, R. Leonardi, M. Vetteri, B. Naylor,** Binary Space Partitioning Tree Representation of Images. *Journal of Visual Communication and Image Representation*, Vol. 2 (3), pp. 201-221, September 1991.

3.1 **H. Samet,** *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

3.1 **X. Wu,** Image Coding by Adaptive Tree-Structured Segmentation. *IEEE Transactions on Information Theory*, Vol. 38 (6), pp. 1755-1767, November 1992.

3.3 **Y.-L. Chang, X. Li,** Adaptive Image Region-Growing. *IEEE Transactions on Image Processing*, Vol. 3 (6), pp. 868-872, November 1994.

3.7 **P.C.V. Hough,** Methods and Means for Recognizing Complex Patterns. U.S. Patent 3,069,654.

3.7 **Leavers V.F.,** "Survey: Which Hough Transform", *CVGIP Image Understanding*, Vol. 58 (2), pp. 250-264, September 1993.

3.7 **H. Kälviäinen, P. Hirvonen, L. Xu, and E. Oja,** "Probabilistic and non-probabilistic Hough transforms: overview and comparisons", *Image and Vision Computing*, Vol. 13 (4), pp. 239-251, May 1995

4.3 **C. Arcelli, G.S. di Baja,** A One-Pass Two-Operation Process to Detect the Skeletal Pixels on the 4-Distance Transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11 (4), pp. 411-414, April 1989

4.5 **D.E. Knuth,** Digital Halftones by Dot Diffusion. *ACM Transactions on Graphics*, Vol. 6 (4), pp. 245-273, October 1987.

4.5 **R. Ulichnet,** *Digital Halftoning*. MIT Press, ISBN 0-262-21009-6, 1987.

4.6 **J. Serra,** *Image Analysis and Mathematical morphology*. Academic Press, London, 1982.

5.1 **P. Heckbert,** Color Image Quantization for Frame Buffer Display. *Computer Graphics*, Vol. 16 (3), pp. 297-307, July 1982.

5.1 **Y. Linde, A. Buzo, R.M. Gray,** An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications*, Vol.28 (1), pp.84-95, January 1980.

6.2 **I.H. Witten, A. Moffat, T.C. Bell,** *Managing Gigabytes*, Van Nostrand Reinhold, New York, 1994.

6.3 **H. Reichenberger, M. Pfeifer,** Objectives and Approaches in Biomedical Signal Processing. *Signal Processing II: Theories and Applications*, ELSEVIER 1993.

6.4 **D. Sutton,** *A Textbook of Radiology and Imaging* (third edition), Churchill Livingstone, Great Britain, 1990.