5.2.4 Contour Representation Based on the Chain Code

A simple and popular alternative to contour representation is based on the chain code proposed in [Freeman, 1961]. In order to understand the chain code scheme, refer to Figure 5.20, which represents a central pixel indicated over an image grid. In an 8-neighborhood, each pixel has 8 neighbors, which can be numbered from 0 to 7, as indicated in that figure. These numbers are the *chain-codes* used by the representation. First, observe that the neighbors have been numbered counterclockwise starting from the right neighbor pixel. Now, refer to Figure 5.21 (the same contour of Figure 5.5), and let p be the starting point of that contour, i.e., p = (4,2). The next neighbor pixel in the counterclockwise direction is the pixel to the right, i.e., (5,2). If the mask of Figure 5.20 is superimposed onto the grid in Figure 5.21 so that the mask central pixel matches the pixel p, then the pixel (5,2) will correspond to the chain-code 0, as indicated in the figure. If the mask is now shifted over the new pixel (5,2), then the next neighbor is (6,2), whose chain-code is also 0. Figure 5.21 illustrates the repetition of this process until the whole contour is traversed. It is easy to see that this contour is represented by the chain-code 00000222224444466666 plus the starting point (4,2).

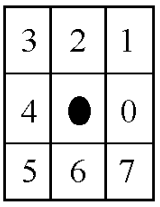


Figure 5.20: *Chain-code.*

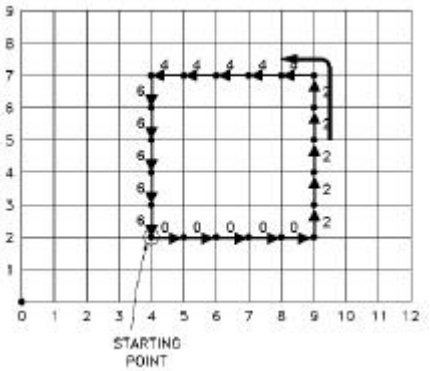
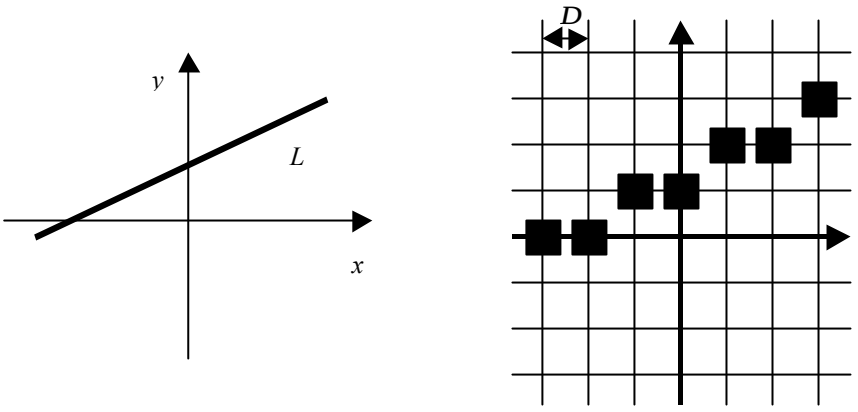


Figure 5.21: *Chain-code representation of a sample contour.*

There are many interesting shape properties (see Box in Section 6.2.16) that can be extracted directly from the chain-code representation [Angulo and Madrigal, 1986; Gonzalez and Woods, 1993]. Two important drawbacks with



(a) (b)

Figure 5.30: *A digital straight generic continuous straight line segment L (a) and the respective line segment (b) obtained by spatially sampling L with a specific spatial sampling D.*

It is therefore clear that, while a DSL is an infinite set of lattice points, a DSLS is a finite set of lattice points, which are henceforth represented in terms of their coordinates (x, y). In case the straight line is defined in terms of one of its possible parametric equations, such as the slope-intercept equation $y = mx + c$, the respective digital straight line can be obtained by varying either x or y while obtaining the other variable. Several are the approaches to spatially sample the straight line given by equation $y = mx + c$, including the following:

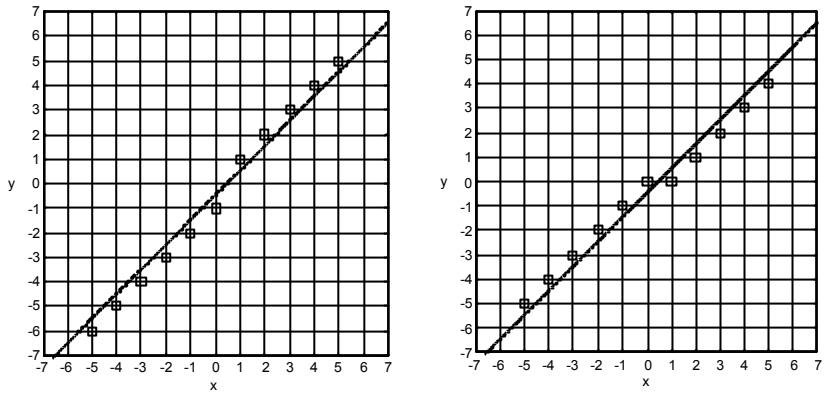
$$y = round(mx + c), \quad x = \cdots, -2, -1, 0, 1, 2, \cdots \tag{5.1}$$

$$y = trunc(mx + c), \quad x = \cdots, -2, -1, 0, 1, 2, \cdots \tag{5.2}$$

$$y = floor(mx + c), \quad x = \cdots, -2, -1, 0, 1, 2, \cdots \tag{5.3}$$

$$y = ceil(mx + c), \quad x = \cdots, -2, -1, 0, 1, 2, \cdots \tag{5.4}$$

Figure 5.31 (a) and (b) present the spatial sampling of the function $y = x - 0.5$ by using Equations (5.1) and (5.2), respectively.



(a) (b)

Figure 5.31: *The spatial sampling of* $y = x - 0.5$ *(represented by the continuous line); for* $y = -5, \cdots -1, 0, 1, 2, \cdots, 5$ *; obtained by using Equation (5.1) in (a) and (5.2) in (b). The sampled values are represented by squares.*

It is clear from this illustration that the *round* and *trunc* functions imply the quantized points behave differently for negative and positive values of y, which is undesirable since it enhances the DSLS discontinuity at $y = 0$. A more consistent approach would be to use the *floor* or *ceil* functions (see Section 2.1.4), but these functions would imply a shift of the quantized straight line towards the left and right, respectively. A more balanced approach consists in shifting the floor function by 0.5 to the left, yielding the following quantization scheme:

$$y_i = floor(mx + c + 0.5), \quad x = \cdots, -2, -1, 0, 1, 2, \cdots \tag{5.5}$$

Figure 5.32 illustrates the quantization of the lines $y = x - 0.5$ by using this scheme. The above characterized additional discontinuity at the transition of y from negative to positive is now clearly avoided.

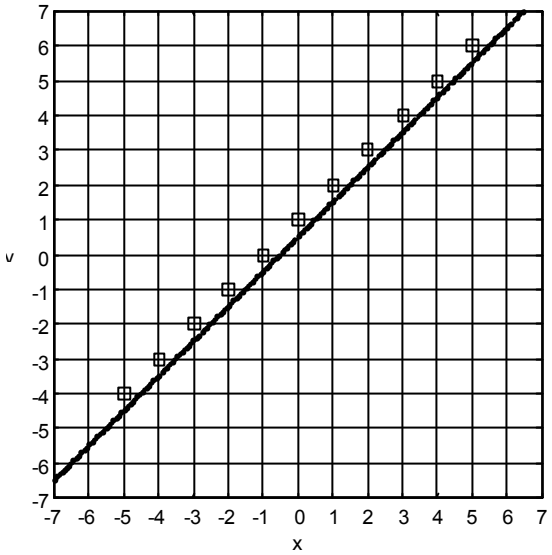


Figure 5.32: *The spatial sampling of the straight line defined by the equation* $y = x - 0.5$ *; for* $x = -5, \cdots -1, 0, 1, 2, \cdots, 5$ *; by using Equation 5.5.*

Therefore, Equation (5.5) is henceforth adopted for obtaining DSLSs from straight line equations such as $y = mx + c$. It is interesting to have a closer look at the sampling implemented by this equation. In order to do so, let us consider that $b = floor(a) \Leftrightarrow b \le a < b + 1$ and b is an integer value, and rewrite Equation (5.5) as below. The reader is referred to the box *Inequalities* for a brief review about inequalities and their algebraic handling.

$$y_i = floor(mx + c + 0.5) \Leftrightarrow y_i \le mx + c + 0.5 < y_i + 1 \Leftrightarrow$$
$$\Leftrightarrow 0 \le mx + c + 0.5 - y_i < 1 \Leftrightarrow -0.5 - mx - c \le -y_i < -mx - c + 0.5 \Leftrightarrow$$
$$\Leftrightarrow mx + c - 0.5 < y_i \le mx + c + 0.5$$

Note: ***Inequalities***

An expression such as $a < x \le b$ is called an *inequality*. As with equalities, inequalities can also be transformed into equivalent forms. For instance, an inequality does not change in case we add (or subtract) the same constant k to all its terms, e.g., $a < x \le b \Leftrightarrow a + k < x + k \le b + k$. A strictly positive constant k (i.e., $k > 0$) can also be multiplied to all terms, i.e., $a < x \le b \Leftrightarrow ak < xk \le bk$. Observe that in case k is strictly negative (i.e., $k < 0$) we have $a < x \le b \Leftrightarrow bk \le xk < ak$. For instance, if $k = -1$ we have $1 < x \le 2 \Leftrightarrow -2 \le -x < -1$.

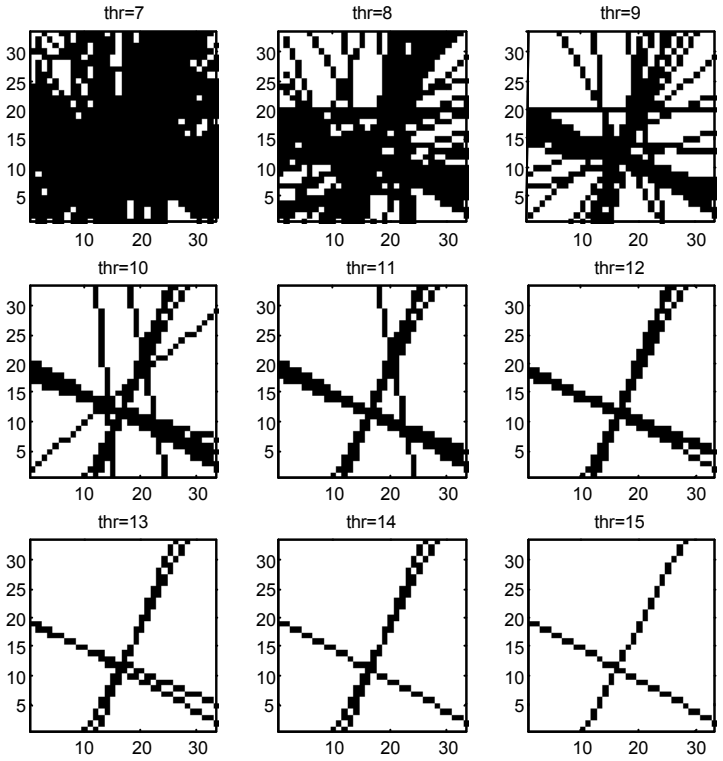


Figure 5.43: *Reconstructions of the image in Figure 5.42 obtained by using successive threshold values (thr). Observe the elimination of the surrounding noise allowed by the HT.*

5.6.4 Backmapping

Introduced in [Gerig and Klein, 1986], the technique known as *backmapping* provides a simple and effective means for reducing the background noise generally obtained in discrete HTs. This technique consists in performing a standard HT, yielding *Acc*, and then repeating the HT calculation over the same input image. However, *Acc* is now searched the peak along each sinusoidal path defined by each foreground pixel in the image, and only the cell in a secondary accumulator array *Acc*2 (initially cleared) having the same coordinates as that peak is incremented. The backmapping technique for the normal HT (the backmapping can easily be extended to other parametrizations) is described by the following pseudo-code:

coordinates of these points, stored sequentially as pairs. This simple implementation of the SEDR as a matrix is illustrated below for $N_M = 4$.

$$SEDR = \begin{bmatrix} 0 & 1 & 0 & 0 & & & & & & & & & & & & & & \\ 1 & 4 & -1 & 0 & 1 & 0 & 0 & -1 & 0 & 1 & & & & & & & & \\ \sqrt{2} & 4 & -1 & -1 & -1 & 1 & 1 & -1 & 1 & 1 & & & & & & & & \\ 2 & 4 & -2 & 0 & 0 & 2 & 2 & 0 & 0 & -2 & & & & & & & & \\ \sqrt{5} & 8 & -2 & 1 & -1 & 2 & 1 & 2 & 2 & 1 & 2 & -1 & 1 & -2 & -1 & -2 & -2 & -1 \end{bmatrix}$$

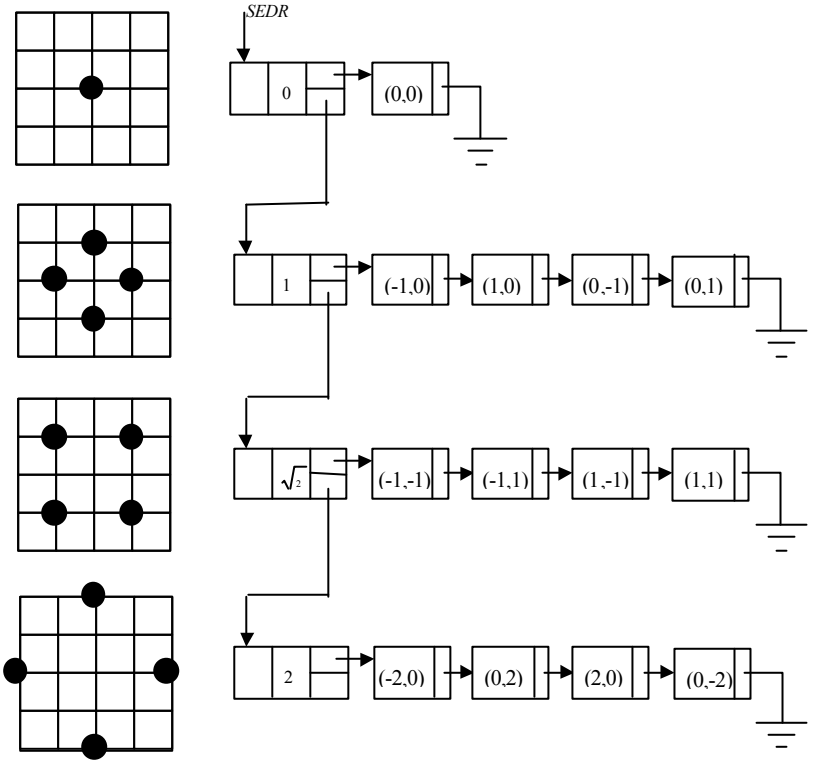


Figure 5.50: *The SEDR for the 4 first exact distances (i.e., $k_M = 3$).*

It is clear that this implementation of the SEDR is not particularly efficient, (especially when compared to the linked list structure in Figure 5.50) as far as storage is concerned, since the number of columns has to be defined in terms of the maximum number of relative positions (in the above example, eight positions for $d = \sqrt{5}$). However, provided that N_M is not too large, the implied additional amount of memory is a small sacrifice to be made in

For instance, a 3×3 image allows only six different distances, as illustrated in Figure 5.55 (i.e., the five distances represented by arrows plus the zero distance, not shown in the figure). Observe that, because of the inherent symmetry of the orthogonal grid, each possible distance in a digital number occurs for a number of points that is always a multiple of 4.

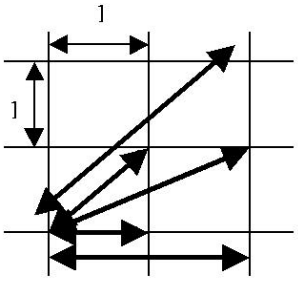


Figure 5.55: *The possible distances in a 3 ´3 orthogonal grid.*

While the number of possible distances, hence N_d, increases substantially for larger image sizes, it will always be finite (as long as the image size is finite). The complexity exhibited by the Euclidean distances in the orthogonal lattice, a consequence of the anisotropy of the latter, has implied a series of practical difficulties that, ultimately, led to many simplified schemes for the calculation of *approximated* Euclidean distance transforms (see, for instance [Borgefors 1984; Hilditch, 1969, Lee and Horng, 1999]). On the other hand, recent advancements have allowed exact Euclidean distance calculation by using vector schemes (e.g. [Vincent, 1991; Cuisenaire, 1999] and Voronoi diagrams (e.g. [Sherbrooke et al., 1996; Ogniewicz, 1992]) which, although highly effective, imply relatively more complex algorithms. Another related reference can be found in I. Ragnemalm, *The Euclidean Distance Transform*, Linköping Studies in Science and Technology 1993. In the present book, we describe an approach to exact Euclidean transform not so simple as the approximated schemes, but also not so complex as the fastest approaches. Therefore, it represents an alternative approach that is easy to implement and whose speed, although not optimal, should be enough to deal with a series of problems in shape analysis, especially when the maximum distance is not too large. This technique, based on the concept of exact dilations described in Section 5.7, is described in the following.

5.9 EXACT DISTANCE TRANSFORM THROUGH EXACT DILATIONS

One of the simplest approaches to obtaining the exact Euclidean distance transform consists in slightly modifying the exact dilation algorithm presented in Section 5.7. This can be done simply by replacing the line $dil\_im(x, y):=1$ by $dst\_tr(x, y):=SEDR(k, 1)$ and including a test in order not to rewrite already

approximated approach to obtaining generalized Voronoi tessellation. This approach is based on the concept of label propagation, also known by the name of SKIZ [Vincent, 1991; Lantuèjoul, 1980], performed by using a simple modification of the above exact dilation algorithm. In this approach, each of the N isolated shapes in the image is labeled with a subsequent integer value (i.e., $L = 1, 2, \ldots, N$) and these labels are propagated around the surrounding space, for instance by using exact dilations. A possible label propagation algorithm is given by the following pseudo-code:

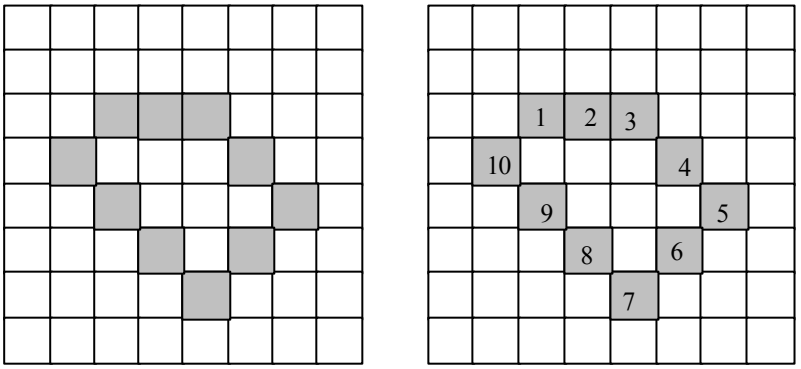
Algorithm: ***Label Propagation through Exact Dilations***

```
Initiate lbl_im with –1;
For k = 1 to N_M  do
  For j = 1 to N  do
    For i = 1 to SEDR(k, 2)  do
      x = L_x(j)+SEDR(k, 2i+1);
      y = L_y(j)+SEDR(k, 2i+2);
      If  lbl_im(x, y) ≠ –1
        lbl_im(x, y)= L_lb(j);
      end
    end
  end
end
```

The labels are propagated through the image *lbl_im*. The list *L_lb* contains the labels of each of the image elements, assigned as above described. Observe that all the pixels in each isolated shape have their coordinates stored in the same lists *L_x* and *L_y*. Although the order in which the isolated shapes are labeled is unimportant, except for a few one-pixel displacements of the separating frontiers, all the pixels in each of the shapes should be stored subsequently into the lists *L_x*, *L_y* and *L_lb*. Figure 5.58 illustrates this fact.

Once the labels have been propagated by using the above-described procedure, a good approximation of the Voronoi tessellation is obtained. This is illustrated in Figure 5.59, with respect to the Voronoi tessellation (b) defined by the isolated shapes in (a).

```
    For y = 2 to Ny-1 do
        max = Max_{i,j=-1,0,1, |i|+|j|=1} {img_lbl(x, y) - img_lbl(x+i, y+j)}        (*)
        If max < N/2
            img_dif(x, y) = max;
        Else
            img_dif(x, y) = N - max;
        end
    end
end
```



(a) (b)

	1	1	2	3	3		
1	1	1	2	3	3	4	
10	1	1	2	3	3	4	4
10	10	1	2	3	4	4	5
10	9	9	8	4	4	5	5
9	9	8	8	6	6	5	5
	8	8	7	7	6	6	5
		7	7	7	7	6	

(c)

Figure 5.61: *Original shape outline (a), contour labeling (b) and propagated labels up to the maximum distance Ö5 (c).*