CHAPTER

Binary shape analysis

8

Although binary images contain much less information than their gray-scale counterparts, they embody shape and size information that is highly relevant for object recognition. However, this information resides in a digital lattice of pixels, and this results in intricacies appearing in the geometry. This chapter resolves these problems and explores a number of important algorithms for processing shapes.

Look out for:

- the connectedness paradox and how it is resolved
- object labeling and how labeling conflicts are resolved
- problems related to measurement in binary images
- size filtering techniques
- the convex hull as a means of characterizing shape, and methods for determining it
- distance functions and how they are obtained using parallel and sequential algorithms
- the skeleton and how it is found by thinning: the crucial role played by the crossing number, both in determining the skeleton and in analyzing it
- simple measures for shape recognition, including circularity and aspect ratio
- more rigorous measures of shape, including moments and boundary descriptors.

In reality, this chapter almost exclusively covers area-based methods of shape analysis, leaving boundary-based procedures to Chapter 9, Boundary Pattern Analysis—though circularity measures and boundary tracking are both covered. However, chapter boundaries cannot be completely exclusive, as any method requires "hooks" that have been laid down in a variety of places, and indeed, it is often valuable to meet a concept before finding out in detail how to put flesh on it.

Returning to the present chapter, it is interesting to note how intricate some of the algorithmic processes are: connectedness, in particular, pervades the whole subject of digital shape analysis and comes with a serious health warning.

8.1 INTRODUCTION

Over the past few decades 2-D shape analysis has provided the main means by which objects are recognized and located in digital images. Fundamentally, 2-D shape has been important because it uniquely characterizes many types of object, from keys to characters, from gaskets to spanners, and from fingerprints to

Computer Vision. DOI: http://dx.doi.org/10.1016/B978-0-12-809284-2.00008-3

chromosomes, whereas in addition it can be represented by patterns in simple binary images. Chapter 1, Vision, The Challenge showed how the template-matching approach leads to a combinatorial explosion even when fairly basic patterns are to be found, so preliminary analysis of images to find features constitutes a crucial stage in the process of efficient recognition and location. Thus, the capability for binary shape analysis is a very basic requirement for practical visual recognition systems.

In fact, 40 years of progress have provided an enormous range of shape analysis techniques and a correspondingly large range of applications. Clearly, it will be impossible to cover the whole field within the confines of a single chapter—so completeness will not even be attempted (the alternative of a catalogue of algorithms and methods, all of which are covered only in brief outline, is eschewed). At one level, the main topics covered are examples with their own intrinsic interest and practical application, and at another level they introduce matters of fundamental principle. Recurring themes are the central importance of connectedness for binary images; the contrasts between local and global operations on images and between different representations of image data; the need to optimize accuracy and computational efficiency; and the compatibility of algorithms and hardware. The chapter starts with a discussion of how connectedness is measured in binary images.

8.2 CONNECTEDNESS IN BINARY IMAGES

This section begins with the assumption that objects have been segmented, by thresholding or other procedures, into sets of 1s in a background of 0s (see Chapters 2–4). At this stage, it is important to realize that a second assumption is already being made implicitly—that it is easy to demarcate the boundaries between objects in binary images. However, in an image that is represented digitally in rectangular tessellation, a problem arises with the definition of connectedness. Consider the following dumbbell-shaped object, which is represented as an array of 1's (all unmarked image points are taken to have the binary value 0):

```
          1 1
      1 1 1 1
          1 1
        1
      1
    1
  1 1
1 1 1 1
  1 1
```

At its center, this object has a segment of the form

$$\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$$

which separates two regions of background. At this point, diagonally adjacent 1s are regarded as being connected, whereas diagonally adjacent 0s are regarded as disconnected—a set of allocations which seems inconsistent. However, we can hardly accept a situation where a connected diagonal pair of 0s crosses a connected diagonal pair of 1s without causing a break in either case. Similarly, we cannot accept a situation in which a disconnected diagonal pair of 0s crosses a disconnected diagonal pair of 1s without there being a join in either case. Hence, a symmetrical definition of connectedness is not possible, and it is conventional to regard diagonal neighbors as connected only if they are foreground, i.e., the foreground is "8-connected," and the background is "4-connected." This convention is followed in the subsequent discussion.

8.3 OBJECT LABELING AND COUNTING

Now, we have a consistent definition of connectedness; we can unambiguously demarcate all objects in binary images and should be able to devise algorithms for labeling them uniquely and counting them. Labeling may be achieved by scanning the image sequentially until a 1 is encountered on the first object; a note is then made of the scanning position, and a "propagation" routine is initiated to label the whole of the object with a 1: as the original image space is already in use, a separate image space has to be allocated for labeling. Next, the scan is resumed, ignoring all points already labeled, until another object is found; this is labeled with a 2 in the separate image space. This procedure is continued until the whole image has been scanned, and all the objects have been labeled (Fig. 8.1). Implicit in this procedure is the possibility of propagating through a

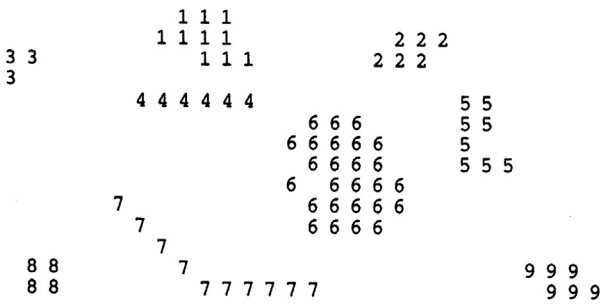

FIGURE 8.1

A process in which all binary objects are labeled.

connected object. Suppose at this stage that no method is available for limiting the field of the propagation routine, so that it has to scan the whole image space. Then, the propagation routine takes the form:

```
do {
    for all points in image
        if point is in an object
          and next to a propagating region labelled N
        assign it the label N
} until no further change;
```
(8.1)

the kernel of the do–until loop being expressed more explicitly as:

```
// original image in A-space; labels to be inserted in P-space
for all pixels in image do {
     if ((A0 == 1)
        && ((P1 == N)||(P2 == N)||(P3 == N)||(P4 == N)
        ||(P5 == N)||(P6 == N)||(P7 == N)||(P8 == N)))
     P0 = N;
}
```
(8.2)

At this stage, a fairly simple type of algorithm for object labeling is obtained, as shown in Table 8.1: the *for forward scan over image do {...}* notation denotes a *sequential* forward raster scan over the image.

Notice that the above object counting and labeling routine requires a *minimum* of $2N + 1$ passes over the image space, and in practice, the number will be closer to $NW/2$ where W is the average width of the objects: hence, the algorithm is inherently rather inefficient. This prompts us to consider how the number of passes over the image could be reduced to save computation. One possibility would be to scan forwards through the image, propagating new labels through objects as they are discovered. Although this would work mostly straightforwardly with convex objects, problems would be encountered with objects possessing concavities—e.g., "U" shapes—as different parts of the same object would end with different labels, and also means would have to be devised for coping with "collisions" of labels (e.g., the largest local label could be propagated through the remainder of the object: see Fig. 8.2). Then, inconsistencies could be resolved by a reverse scan through the image. However, this procedure will not resolve all problems that can arise, as in the case of more complex (e.g., spiral) objects. In such cases, a general parallel propagation, repeatedly applied until no further labeling occurs, might be preferable—though as we have seen, such a process is inherently rather computation intensive. However, it is implemented very conveniently on certain types of parallel processor, such as single instruction stream, multiple data stream (SIMD) machines.

Ultimately, the least computationally intensive procedures for propagation involve a different approach: objects and parts of objects are labeled on a *single* sequential pass through the image, at the same time noting which labels coexist on objects. Then, the labels are sorted separately, in a stage of abstract information processing, to determine how the initially rather ad hoc labels should

Table 8.1 A Simple Algorithm for Object Labeling

```
// start with binary image containing objects in A-space
// clear label space
for all pixels in image do { P0 = 0; }
// start with no objects
N = 0;
/* look for objects using a sequential scan and propagate labels through them*/
do { // search for an unlabelled object
     found = false;
     for forward scan over image do {
        if ( (A0 == 1) && (P0 == 0) && not found) {
              N = N + 1;
              P0 = N;
              found = true;
        }
     }
     if (found) // label the object just found
        do {
              finished = true;
              for all pixels in image do {
                if ( (A0 == 1) && (P0 == 0)
                      && ( (P1 == N) || (P2 == N) || (P3 == N) || (P4 == N)
                      || (P5 == N) || (P6 == N) || (P7 == N) || (P8 == N) ) ) {
                          P0 = N ;
                          finished = false;
                }
              }
        } until finished;
} until not found; // i.e. no (more) objects
// N is the number of objects found and labelled
```

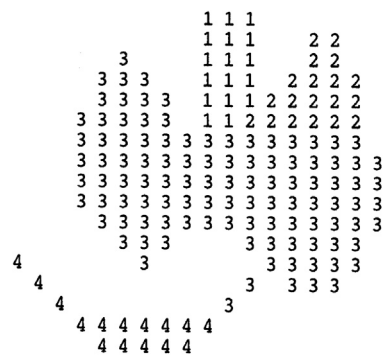

FIGURE 8.2

Labeling U-shaped objects: a problem that arises in labeling certain types of object if a simple propagation algorithm is used. Some provision has to be made to accept "collisions" of labels although the confusion can be removed by a subsequent stage of processing.

Table 8.2 The Improved Algorithm for Object Labeling

```
// clear label space
for all pixels in image do { P0 = 0; }
// start with no objects
N = 0;
// clear the table that is to hold the label coexistence data
for (i = 1; i <= Nmax; i++)
     for (j = 1; j <= Nmax; j++)
          coexist[i][j]= false;
// label objects in a single sequential scan
for forward scan over image do {
    if (A0 == 1) {
    }
        if ( (P2 == 0) && (P3 == 0) && (P4 == 0) && (P5 == 0) ) {
               N = N + 1;
               P0 = N;
         }
         else {
               P0 = max(P2, P3, P4, P5);
               // now note which labels coexist in objects
               coexist[P0][P2] = true;
               coexist[P0][P3] = true;
               coexist[P0][P4] = true;
               coexist[P0][P5] = true;
         }
    }
}
analyse the coexist table and decide ideal labelling scheme;
relabel image if necessary;
```

be interpreted. Finally, the objects are relabeled appropriately in a second pass over the image (in fact, this latter pass is sometimes unnecessary, as the image data are merely labeled in an overcomplex manner and what is needed is simply a key to interpret them). The improved labeling algorithm now takes the form shown in Table 8.2. Clearly, this algorithm with its single sequential scan is intrinsically far more efficient than the previous one, although the presence of particular dedicated hardware or a suitable SIMD processor might alter the situation and justify the use of alternative procedures.

It will be clear that minor amendments to the above algorithms permit the areas and perimeters of objects to be determined: thus, objects may be labeled by their areas or perimeters instead of by numbers representing their order of appearance in the image. More important, the availability of propagation routines means that objects can be considered in turn in their entirety—if necessary by transferring them individually to separate image spaces or storage areas ready for unencumbered independent analysis. Evidently, if objects appear in individual binary spaces, maximum and minimum spatial coordinates are trivially measurable, centroids can readily be found and more detailed calculations of moments (see below) and other parameters can easily be undertaken.

8.3.1 SOLVING THE LABELING PROBLEM IN A MORE COMPLEX CASE

In this section, we add substance to the all too facile statement at the end of Table 8.2—"analyze coexist table and decide ideal labeling scheme." First, we have to make the task nontrivial by providing a suitable example. Fig. 8.3 shows an example image, in which sequential labeling has been carried out in line with the algorithm of Table 8.2. However, one variation has been adopted—of using a minimum rather than a maximum labeling convention, so that the values are in general slightly closer to the eventual ideal labels. (This also serves to demonstrate that there is not just one way of designing a suitable labeling algorithm.) The algorithm itself indicates that the coexist table should now appear as in Table 8.3. However, the whole process of calculating ideal labels can be made more efficient by inserting numbers instead of ticks, and also adding the right numbers along the leading diagonal, as in Table 8.4; for the same reason, the numbers below the leading diagonal, which are technically redundant, are retained here.

The next step is to minimize the entries along the individual rows of the table, as in Table 8.5. Then, we minimize along the individual columns (Table 8.6). Then, we minimize along rows again (Table 8.7). This process is iterated to completion, which has already happened here after three stages of minimization. We can now read off the final result from the leading diagonal. Note that a further stage of computation is needed to make the resulting labels consecutive integers, starting with unity. However, the procedure needed to achieve this is much more basic and does not need to manipulate a 2-D table of data: this will be left as a simple programing task for the reader.

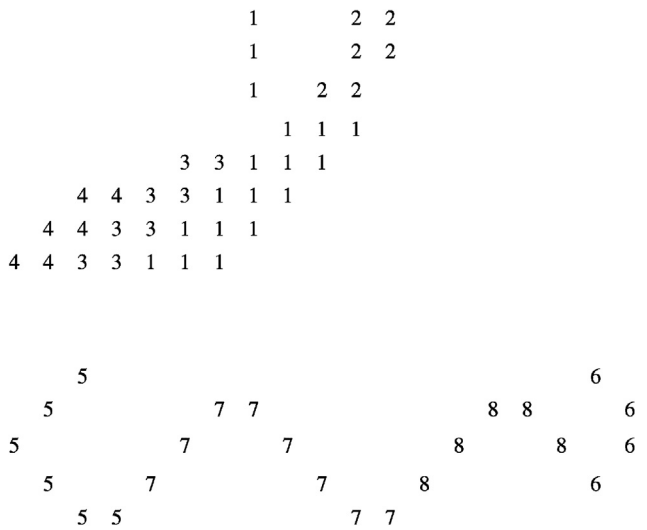

FIGURE 8.3

Solving the labeling problem in a more complex case.

Table 8.3 Coexist Table for the Image of Fig. 8.3

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1		√	√					
2	√							
3	√			√				
4			√					
5							√	
6								√
7					√			√
8						√	√	

The ticks correspond to clashes of labels.

Table 8.4 Coexist Table with Additional Numerical Information

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

This coexist table is an enhanced version of Table 8.3. Technically, the numbers along, and below, the leading diagonal are redundant, but nevertheless they speed up the subsequent computation.

Table 8.5 Coexist Table Redrawn with Minimized Rows

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

At this stage, the table is no longer symmetric.

Table 8.6 Coexist Table Redrawn Again With Minimized Columns

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

Table 8.7 Coexist Table Redrawn Yet Again With Minimized Rows

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

At this stage, the table is in its final form and is once again symmetric.

At this point, some comment on the nature of the process described above will be appropriate. What has happened is that the original image data has effectively been condensed into the minimum space required to express the labels—namely just one entry per original clash. This explains why the table retains the 2-D format of the original image: lower dimensionality would not permit the image topology to be represented properly. It also explains why minimization has to be carried out, to completion, in two orthogonal directions. On the other hand, the particular implementation, including both above- and below-diagonal elements, is able to minimize computational overheads and finalize the operation in remarkably few iterations.

Finally, it might be felt that too much attention has been devoted to finding connected components of binary images. In fact, this is a highly important topic in practical applications such as industrial inspection, where it is crucial to locate all the objects unambiguously before they can individually be identified and scrutinized. In addition, Fig. 8.3 makes clear that it is not only U-shaped objects that give problems but also those that have shape subtleties—as happens at the left of the upper object in this figure.

8.4 SIZE FILTERING

Before proceeding to study size filtering, we draw attention to the fact that the 8-connected and 4-connected definitions of connectedness lead to the following measures of distance (or "metrics") that apply to pairs of pixels, labeled i and j, in a digital lattice:

$$d_8 = \max\left(|x_i - x_j|, |y_i - y_j|\right) \tag{8.3}$$

and

$$d_4 = |x_i - x_j| + |y_i - y_j| \tag{8.4}$$

Although the use of the d_4 and d_8 metrics is bound to lead to certain inaccuracies, it is useful to see what can be achieved with the use of local operations in binary images. This section studies how simple size filtering operations can be carried out, using merely local (3×3) operations. The basic idea is that small objects may be eliminated by applying a series of shrink operations. In fact, N-shrink operations will eliminate an object (or those parts of an object) that are $2N$ or fewer pixels across their narrowest dimension. Of course, this process shrinks *all* the objects in the image, but in principle a subsequent N expand operations will restore the larger objects to their former size.

If complete elimination of small objects is required but perfect retention of larger objects, this will, however, not be achieved by the above procedure, as in many cases the larger objects will be distorted or fragmented by these operations (Fig. 8.4). To recover the larger objects in their *original* form, the proper approach is to use the shrunken versions as "seeds" from which to grow the originals via a propagation process. The algorithm of Table 8.8 is able to achieve this.

Having seen how to remove whole (connected) objects that are everywhere narrower than a specified width, it is possible to devise algorithms for removing any subset of objects that are characterized by a given range of widths: large objects may be filtered out by first removing lesser sized objects and then performing a logical masking operation with the original image, whereas intermediate-sized objects may be filtered out by removing a larger subset and then restoring small objects that have previously been stored in a separate image space. Ultimately, all these schemes depend on the availability of the propagation technique, which in turn depends on the internal connectedness of individual objects.

Finally, note that expand operations followed by shrink operations may be useful for joining nearby objects, filling in holes, and so on. Numerous refinements and additions to these simple techniques are possible. A particularly interesting one is to separate the silhouettes of touching objects such as chocolates by a shrinking operation: this then permits them to be counted reliably (Fig. 8.5).

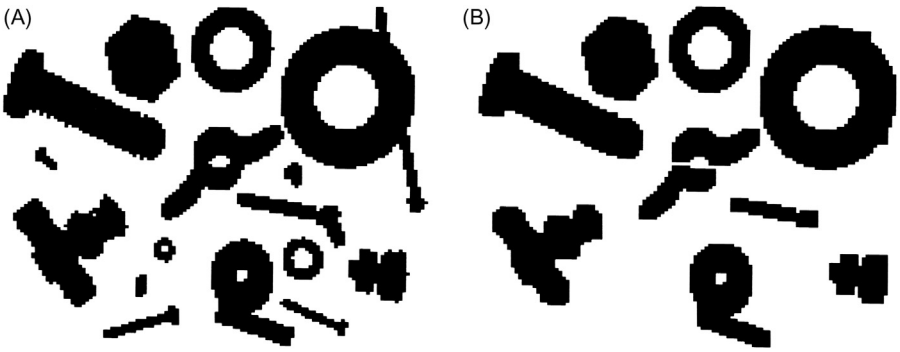

FIGURE 8.4

The effect of a simple size filtering procedure. When size filtering is attempted by a set of *N* shrink and *N* expand operations, larger objects are restored approximately to their original size but their shapes frequently become distorted or even fragmented. In this example, (B) shows the effect of applying two shrink and then two expand operations to the image in (A).

Table 8.8 Algorithm for Recovering Original Forms of Shrunken Objects

```
// save original image
for all pixels in image do { C0 = A0; }
// now shrink the original objects N times
for (i = 1; i <= N; i++) {
      for all pixels in image do {
        sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8;
          if (sigma < 8) B0 = 0; else B0 = A0;
      }
      for all pixels in image do {A0 = B0; }
}
// next propagate the shrunken objects using the original image
do {
      finished = true;
      for all pixels in image do {
        sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8;
          if ( (A0 == 0) && (sigma > 0) && (C0 == 1) ) {
                A0 = 1;
                finished = false;
          }
      }
} until finished;
```

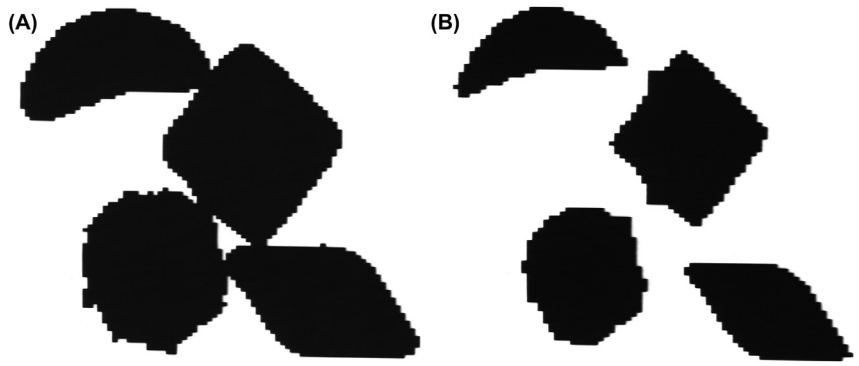

FIGURE 8.5

Separation of touching objects by shrink operations. Here, objects (chocolates) in (A) are shrunk (B) in order to separate them so that they may be counted reliably.

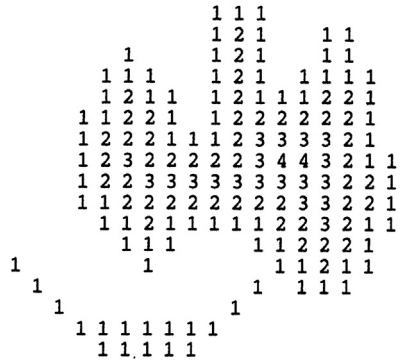

FIGURE 8.6

The distance function of a binary shape: the value at every pixel is the distance (in the d_8 metric) from the background.

8.5 DISTANCE FUNCTIONS AND THEIR USES

The distance function of an object is a very simple and useful concept in shape analysis. Essentially, each pixel in the object is numbered according to its distance from the background. As usual, background pixels are taken as 0s; then edge pixels are counted as 1s; object pixels next to 1s become 2s; next to those are the 3s; and so on throughout all the object pixels (Fig. 8.6).

The parallel algorithm of Table 8.9 finds the distance function of binary objects by propagation. Note that this algorithm performs a final pass in which nothing happens; this is inevitable if we are to be certain that the process will run to completion.

Table 8.9 A Parallel Algorithm for Propagating Distance Functions

```
// Start with binary image containing objects in A-space
for all pixels in image do { Q0 = A0 *255; }
N = 0;
do {
    finished = true;
    for all pixels in image do {
      if ( (Q0 == 255) // in object and no answer yet
           && ( (Q1 == N) || (Q2 == N) || (Q3 == N) || (Q4 == N)
           || (Q5 == N) || (Q6 == N) || (Q7 == N) || (Q8 == N) ) ) {
                // next to an N
                Q0 = N + 1;
                finished = false; // some action has been taken
           }
    }
    N = N + 1;
} until finished;
```

It is possible to perform the propagation of a distance function with far fewer operations if sequential processing is used. In 1-D, the basic idea would be to build up ramps within objects using a routine like the following:

```
for all pixels in a row of pixels do {Q0 = A0*255;}
for forward scan over row of pixels do
  if (Q0 > Q5 + 1) Q0 = Q5 + 1;
```

Next, we need to insist on double-sided ramps within objects, both horizontally and vertically. This is elegantly achieved using two sequential operations, one being a normal forward raster scan and the other being a reverse raster scan:

```
for all pixels in image do {Q0 = A0 *255;}
for forward scan over image do {
    minplusone = min(Q2, Q3, Q4, Q5) + 1;
    if (Q0 > minplusone) Q0 = minplusone;
}                                                          (8.5)
for reverse scan over image do {
    minplusone = min(Q6, Q7, Q8, Q1) + 1;
    if (Q0 > minplusone) Q0 = minplusone;
}
```

Note the compact notation being used to distinguish between forward and reverse raster scans over the image: *for forward scan over image do {...}* denotes a forward raster scan, whereas *for reverse scan over image do {...}* denotes a reverse raster scan. A more succinct version of this algorithm is the following:

```
for all pixels in image do {Q0 = A0*255;}
for forward scan over image do {
    Q0 = min(Q0 − 1, Q2, Q3, Q4, Q5) + 1;
}                                                          (8.6)
for reverse scan over image do {
    Q0 = min(Q0 − 1, Q6, Q7, Q8, Q1) + 1;
}
```

Before moving on, it will be useful to emphasize the value of sequential processing for propagating distance functions. In fact, when this sequential algorithm is run on a serial computer, it will be O(N) times *faster* than the corresponding parallel algorithm running on a serial computer, but O(N) times *slower* than the same parallel algorithm running on a parallel computer, for an $N \times N$ image. Although this statement is specific to propagation of distance functions, similar statements can be made about a good many other operations. (Note that parallel processing is achieved very efficiently using parallel computers known SIMD machines.)

8.5.1 LOCAL MAXIMA AND DATA COMPRESSION

An interesting application of distance functions is that of data compression. To achieve this, operations are carried out to locate those pixels which are local maxima of the distance function (Fig. 8.7), as storing these pixel values and positions permits the original image to be regenerated by a process of downwards propagation (see below). Note that although finding the local maxima of the distance function provides the basic information for data compression, the actual compression occurs only when the data are stored as a list of points rather than in the original picture format. In order to locate the local maxima, the following parallel routine may be employed:

```
for all pixels in image do{
    maximum = max(Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8);
    if ((Q0 > 0) && (Q0 ≥ maximum)) B0 = 1; else B0 = 0;
}
```
(8.7)

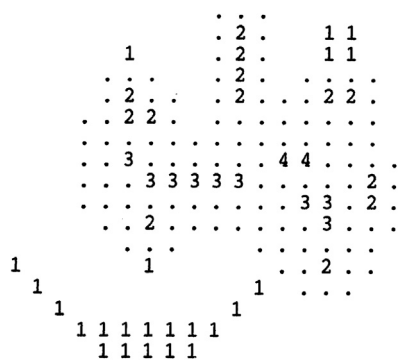

FIGURE 8.7

Local maxima of the distance function of the shape shown in Fig. 8.6, the remainder of the shape being indicated by dots and the background being blank. Notice that the local maxima group themselves into clusters each containing points of equal distance function value, whereas clusters of different values are clearly separated.

Alternatively, the compressed data can be transferred to a single image space:

```
for all pixels in image do {
    maximum = max(Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8);
    if ((Q0 > 0) && (Q0 ≥ maximum)) P0 = Q0; else P0 = 0;
}
```
(8.8)

Note that the local maxima that are retained for the purpose of data compression are not absolute maxima but are maximal in the sense of not being adjacent to larger values. If this were not so, insufficient numbers of points would be retained for completely regenerating the original object. As a result of this, it is found that the absolute maxima group themselves into clusters of connected points, each cluster having a common distance value and being separated from points of different distance values (Fig. 8.7). Thus, the set of local maxima of an object is not a connected subset. This fact has an important bearing on skeleton formation (see below).

Having seen how data compression may be performed by finding local maxima of the distance function, it is relevant to consider a parallel downwards propagation algorithm (Table 8.10) for recovering the shapes of objects from an image into which the values of the local maxima have been inserted. Note again that if it can be assumed that at most N passes are needed to propagate through objects of known maximum width, then the algorithm becomes simply:

```
for (i = 1; i <= N; i++)
    for all pixels in image do {
      Q0 = max(Q0 + 1, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8) − 1;
}
```
(8.9)

Table 8.10 A Parallel Algorithm for Recovering Objects From Local Maxima of the Distance Functions

```
// assume that input image is in Q-space, and that non-maximum values have
value 0
do {
    finished = true;
    for all pixels in image do {
        maxminusone = max(Q0 + 1,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8) − 1;
        if (Q0 < maxminusone){
            Q0 = maxminusone;
            finished = false; // some action has been taken
        }
    }
} until finished;
```

8.6 SKELETONS AND THINNING

The skeleton is a powerful analog concept which may be employed for the analysis and description of shapes in binary images. A skeleton may be defined as a connected set of medial lines along the limbs of a figure: for example, in the case of thick hand-drawn characters, the skeleton may be supposed to be the path actually traveled by the pen. In fact, the basic idea of the skeleton is that of eliminating redundant information while retaining only the topological information concerning the shape and structure of the object that can help with recognition. In the case of hand-drawn characters, the thickness of the limbs is taken to be irrelevant: it may be constant and therefore carry no useful information, or it may vary randomly and again be of no value for recognition (Fig. 1.2).

The definition presented above leads to the idea of finding the loci of the centers of maximal discs inserted within the object boundary. First, suppose the image space to be a continuum. Then, the discs are circles, and their centers form loci which may be modeled very conveniently when object boundaries are approximated by linear segments. In fact, sections of the loci fall into three categories:

1. they may be angle bisectors, i.e., lines which bisect corner angles and reach right up to the apexes of corners;
2. they may be lines that lie half-way between boundary lines;
3. they may be parabolas which are equidistant from lines and from the nearest points of other lines—namely, corners where two lines join.

Clearly, Categories 1 and 2 are special forms of a more general case.

These ideas lead to unique skeletons for objects with linear boundaries, and the concepts are easily generalizable to curved shapes. In fact, this approach tends to give rather more detail than is commonly required, even the most obtuse corner having a skeleton line going into its apex (Fig. 8.8). Hence, a thresholding scheme is often employed such that skeleton lines only reach into corners having a specified minimum degree of sharpness.

We now have to see how the skeleton concept will work in a digital lattice. Here, we are presented with an immediate choice: which metric should we employ? If we select the Euclidean metric (i.e., lattice distance is measured as the Euclidean distance between pairs of pixels), there may be a considerable computational load. If we select the d_8 metric, we will immediately lose accuracy but the computational requirements should be more modest (we do not here consider the d_4 metric, as we are dealing with the shapes of foreground objects). In what follows we concentrate on the d_8 metric.

At this stage, some thought shows that the application of maximal discs in order to locate skeleton lines amounts essentially to finding the positions of local maxima of the distance function. Unfortunately, as seen in the previous section, the set of local maxima does not form a connected graph within a given object, nor is it necessarily composed of thin lines, and indeed it may in places be 2 pixels

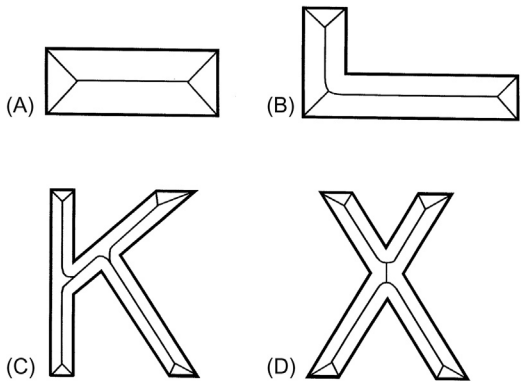

FIGURE 8.8

Four shapes whose boundaries consist entirely of straight line segments. The idealized skeletons go right to the apex of each corner, however obtuse. In certain parts of shapes (B), (C) and (D), the skeleton segments are parts of parabolas rather than straight lines. As a result, the detailed shape of the skeleton (or the approximations produced by most algorithms operating in discrete images) is not exactly what might initially be expected or what would be preferred in certain applications.

wide. Thus, problems arise in trying to use this approach to obtain a connected unit-width skeleton that can conveniently be used to represent the shape of the object. We shall return to this approach again below. Meanwhile, however, we pursue an alternative idea—that of thinning.

Thinning is perhaps the simplest approach to skeletonization. It may be defined as the process of systematically stripping away the outermost layers of a figure until only a connected unit-width skeleton remains (see e.g., Fig. 8.9). A number of algorithms are available to implement this process, with varying degrees of accuracy, and we discuss below how a specified level of precision can be achieved and tested for. First, however, it is necessary to discuss the mechanism by which points on the boundary of a figure may validly be removed in thinning algorithms.

8.6.1 CROSSING NUMBER

The exact mechanism for examining points to determine whether they can be removed in a thinning algorithm must now be considered. This may be decided by reference to the *crossing number* χ (chi) for the 8 pixels around the outside of a particular 3×3 neighborhood. χ is defined as the total number of 0-to-1 and 1-to-0 transitions on going once round the outside of the neighborhood: this number is in fact twice the number of potential connections joining the remainder of the object to the center of the neighborhood (Fig. 8.10). Unfortunately, the formula for χ is made more complex by the 8-connectedness

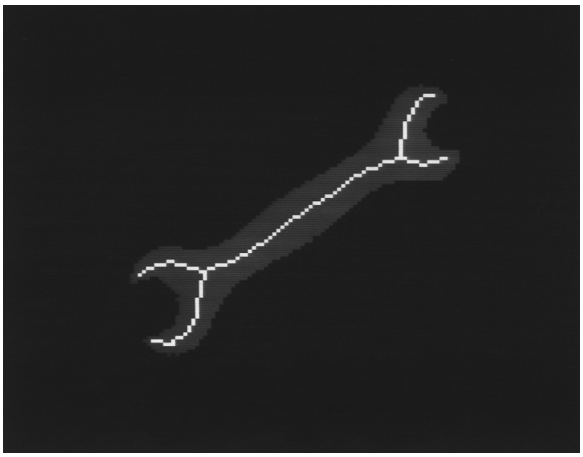

FIGURE 8.9

Typical result of a thinning algorithm operating in a discrete lattice.

0 0 0 0 1 0 0 0 0	0 0 0 0 1 1 1 1 1	1 0 0 0 1 1 1 1 1	1 0 0 0 1 0 0 1 0	1 0 1 0 1 0 1 1 1	1 0 0 0 1 0 1 0 1	1 0 1 0 1 0 1 0 1
0	2	4	4	6	6	8

FIGURE 8.10

Some examples of the crossing number values associated with given pixel neighborhood configurations (0, background; 1, foreground).

criterion. To express this efficiently, we use the C++ "(int)" construct to convert logical outcomes true and false to integer outcomes 1 and 0. Basically, we would expect

$$\begin{aligned}\text{badchi} = {} & (\text{int})(\text{A1} \mathrel{!}= \text{A2}) + (\text{int})(\text{A2} \mathrel{!}= \text{A3}) + (\text{int})(\text{A3} \mathrel{!}= \text{A4}) \\ & + (\text{int})(\text{A4} \mathrel{!}= \text{A5}) + (\text{int})(\text{A5} \mathrel{!}= \text{A6}) + (\text{int})(\text{A6} \mathrel{!}= \text{A7}) \\ & + (\text{int})(\text{A7} \mathrel{!}= \text{A8}) + (\text{int})(\text{A8} \mathrel{!}= \text{A1})\end{aligned} \tag{8.10}$$

However, this is incorrect because of the 8-connectedness criterion. For example, in the case

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

the formula gives the value 4 for χ instead of 2. The reason is that the isolated 0 in the top right-hand corner does not prevent the adjacent 1s from being joined. It is therefore tempting to use the modified formula:

$$\begin{aligned}\text{wrongchi} = {} & (\text{int})(\text{A1} \mathrel{!}= \text{A3}) + (\text{int})(\text{A3} \mathrel{!}= \text{A5}) + (\text{int})(\text{A5} \mathrel{!}= \text{A7}) \\ & + (\text{int})(\text{A7} \mathrel{!}= \text{A1});\end{aligned} \tag{8.11}$$

However, this too is wrong, as in the case

$$\begin{matrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{matrix}$$

it gives the answer 2 instead of 4. It is therefore necessary to add four extra terms to deal with isolated 1s in the corners:

```
chi = (int)(A1 != A3) + (int)(A3 != A5) + (int)(A5 != A7)
    + (int)(A7 != A1)
    + 2*((int)((A2 > A1) && (A2 > A3)) + (int)((A4 > A3) && (A4 > A5))
    + (int)((A6 > A5) && (A6 > A7)) + (int)((A8 > A7) && (A8 > A1)));
```

(8.12)

This (now correct) formula for crossing number gives values 0, 2, 4, 6, or 8 in different cases (Fig. 8.10). The rule for removing points during thinning is that points may only be removed if they are at those positions on the boundary of an object where χ is 2: when χ is greater than 2, the point *must* be retained, as it forms a vital connected point between two parts of the object; in addition, when it is 0, it must be retained as removing it would create a hole.

Finally, there is one more condition that must be fulfilled before a point can be removed during thinning—that the sum σ (sigma) of the eight pixel values around the outside of the 3 $\times$ 3 neighborhood (see Chapter 2: Images and Imaging Operations) must not be equal to 1. The reason for this is to preserve line ends, as in the following cases:

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

Clearly, if line ends are eroded as thinning proceeds, the final skeleton will not represent the shape of an object (including the relative dimensions of its limbs) at all accurately (however, it is possible that we might sometimes wish to shrink an object while preserving connectedness, in which case this extra condition need not be implemented). Having covered these basics, we are now in a position to devise complete thinning algorithms.

8.6.2 PARALLEL AND SEQUENTIAL IMPLEMENTATIONS OF THINNING

Thinning is "essentially sequential" in that it is easiest to ensure that connectedness is maintained by arranging that only one point may be removed at a time. As indicated above, this is achieved by checking before removing a point that it has a crossing number of 2. Now imagine applying the "obvious" sequential algorithm of Table 8.11 to a binary image. Assuming a normal forward raster scan, the result of this process is to produce a highly distorted skeleton, consisting

Table 8.11 An "Obvious" Sequential Thinning Algorithm

```
do {
    finished = true;
    for forward scan over image do {
      sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8;
      chi = (int) (A1 != A3) + (int) (A3 != A5) + (int) (A5 != A7)
            + (int) (A7 != A1)
            + 2*((int) ((A2 > A1) && (A2 > A3))+(int) ((A4 > A3) && (A4 > A5))
            + (int) ((A6 > A5) && (A6 > A7))+(int) ((A8 > A7) && (A8 > A1)));
      if ( (A0 == 1) && (chi == 2) && (sigma != 1) ) {
            A0 = 0;
            finished = false; // some action has been taken
      }
    }
} until finished;
```

of lines along the right-hand and bottom edges of objects. It may now be seen that the $\chi = 2$ condition is necessary but not sufficient, as it says nothing about the order in which points are removed. To produce a skeleton that is unbiased, giving a set of truly medial lines, it is necessary to remove points as evenly as possible around the object boundary. A scheme that helps with this involves a novel processing sequence: mark edge points on the first pass over an image; on the second pass, strip points sequentially as in the above algorithm, *but only where they have already been marked*; then mark a new set of edge points; then perform another stripping pass; then repeat this marking and stripping sequence until no further change occurs. An early algorithm working on this principle is that of Beun (1973).

Although the novel sequential thinning algorithm described above can be used to produce a reasonable skeleton, it would be far better if the stripping action could be performed symmetrically around the object, thereby removing any possible skeletal bias. In this respect, a parallel algorithm should have a distinct advantage. However, parallel algorithms result in several points being removed at once: this means that lines 2 pixels wide will disappear (as masks operating in a 3×3 neighborhood cannot "see" enough of the object to judge whether a point may validly be removed or not), and as a result, shapes can become disconnected. The general principle for avoiding this problem is to strip points lying on different parts of the boundary in different passes, so that there is no risk of causing breaks. In fact, there is a very large number of ways of achieving this, by applying different masks and conditions to characterize different parts of the boundary. If boundaries were always convex, the problem would no doubt be reduced; however, boundaries can be very convoluted and are subject to quantization noise, so the problem is a complex one. With so many potential solutions to the problem, we concentrate here on one that can conveniently be analyzed and which gives acceptable results.

The method discussed is that of removing north, south, east, and west points cyclically until thinning is complete. North points are defined as the following:

$$\begin{array}{ccc} \times & 0 & \times \\ \times & 1 & \times \\ \times & 1 & \times \end{array}$$

where $\times$ means either a 0 or a 1: south, east, and west points are defined similarly. It is easy to show that all north points for which $\chi = 2$ and $\sigma \neq 1$ may be removed in parallel without any risk of causing a break in the skeleton—and similarly for south, east, and west points. Thus, a possible format for a parallel thinning algorithm in rectangular tessellation is the following:

```
do {
   strip appropriate north points;
   strip appropriate south points;
   strip appropriate east points;
   strip appropriate west points;
} until no further change;
```
(8.13)

where the basic parallel routine for stripping "appropriate" north points is:

```
for all pixels in image do {
  sigma = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8;
  chi = (int)(A1 != A3) + (int)(A3 != A5) + (int)(A5 != A7)
        + (int)(A7 != A1)
        + 2* ((int)((A2 > A1) && (A2 > A3)) + (int)((A4 > A3) && (A4 > A5))
        + (int)((A6 > A5) && (A6 > A7)) + (int)((A8 > A7) && (A8 > A1)));
  if ((A3 == 0) && (A0 == 1) && (A7 == 1) //north point
        && (chi == 2) && (sigma != 1))
        B0 = 0;
  else B0 = A0;
}
```
(8.14)

(but extra code needs to be inserted to detect whether any changes have been made in a given pass over the image).

Algorithms of the above type can be highly effective, although their design tends to be rather intuitive and ad hoc. In a survey made by the author in 1981 (Davies and Plummer, 1981), a great many such algorithms exhibited problems. Ignoring cases where the algorithm design was insufficiently rigorous to maintain connectedness, four other problems were evident:

1. the problem of skeletal bias;
2. the problem of eliminating skeletal lines along certain limbs;
3. the problem of introducing "noise spurs";
4. the problem of slow speed of operation.

In fact, problems 2 and 3 are opposites in many ways: if an algorithm is designed to suppress noise spurs, it is liable to eliminate skeletal lines in some circumstances; contrariwise, if an algorithm is designed never to eliminate

skeletal lines, it is unlikely to be able to suppress noise spurs. This situation arises as the masks and conditions for performing thinning are intuitive and ad hoc, and therefore have no basis for discriminating between valid and invalid skeletal lines: ultimately, this is because it is difficult to build overt global models of reality into purely local operators. In a similar way, algorithms that proceed with caution, i.e., which do not remove object points in the fear of making an error or causing bias, tend to be slower in operation than they might otherwise be. Again, it is difficult to design algorithms that can make correct global decisions rapidly via intuitively designed local operators. Hence, a totally different approach is needed if solving one of the above problems is not to cause difficulties with the others. Such an alternative approach is discussed in the next section.

8.6.3 GUIDED THINNING

This section returns to the ideas of Section 8.5.1, where it was found that the local maxima of the distance function do not form an ideal skeleton because they appear in clusters and are not connected. In addition, the clusters are often two pixels wide. On the plus side, the clusters are accurately in the correct positions and should therefore not be subject to skeletal bias. Hence, an ideal skeleton should result if (1) the clusters could be reconnected appropriately and (2) the resulting structure could be reduced to unit width—though, of course, a unit width skeleton can only be perfectly unbiased where the object is an odd number of pixels wide.

A simple means of reconnecting the clusters is to use them to guide a conventional thinning algorithm (see Section 8.6.2). As a first stage, thinning is allowed to proceed normally but with the proviso that no cluster points may be removed. This gives a connected graph which is in certain places 2 pixels wide. Then, a routine is applied to strip the graph down to unit width. At this stage an unbiased skeleton (within 1/2 pixel) should result. The main problem here is the presence of noise spurs. The opportunity now exists to eliminate these systematically by applying suitable global rules. A simple rule is that of eliminating lines on the skeletal graph that terminate in a local maximum of value (say) 1 (or, better, stripping them back to the next local maximum), as such a local maximum corresponds to fairly trivial detail on the boundary of the object. Thus, the level of detail that is ignored can be programed into the system (Davies and Plummer, 1981). The whole guided thinning process is shown in Fig. 8.11.

8.6.4 A COMMENT ON THE NATURE OF THE SKELETON

At the beginning of Section 8.6, the case of character recognition was taken as an example, and it was stated that the skeleton may be supposed to be the path traveled by the pen in drawing out the character. However, in one important respect, this is not valid. The reason is seen both in the analog reasoning and from the results of thinning algorithms. Take the case of a letter K. The vertical limb on

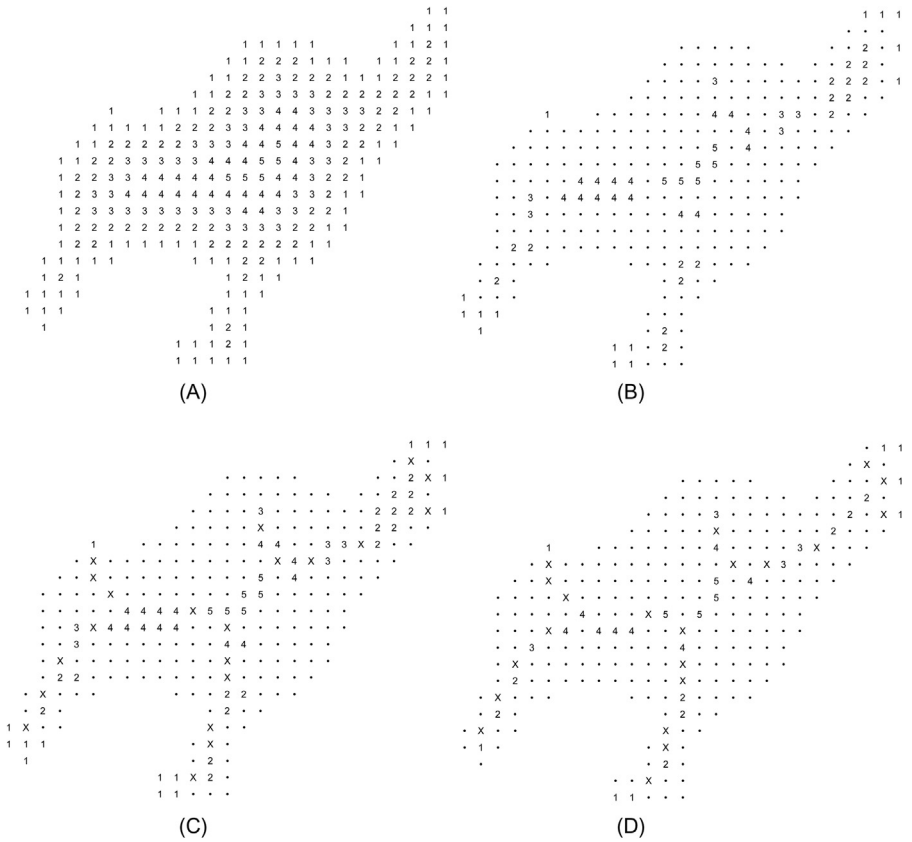

FIGURE 8.11

Results of a guided thinning algorithm: (A) distance function on the original shape; (B) set of local maxima; (C) set of local maxima now connected by a simple thinning algorithm; (D) final thinned skeleton. The effect of removing noise spurs systematically, by cutting limbs terminating in a 1 back to the next local maximum, is easily discernible from the result in (D): the general shape of the object is not perturbed by this process.

the left of the skeleton will theoretically consist of two linear segments joined by two parabolic segments leading into the junction (Fig. 8.8). This limb will only become straight if a higher level model is used to constrain the result.

8.6.5 SKELETON NODE ANALYSIS

Skeleton node analysis may be carried out very simply with the aid of the crossing number concept. Points in the middle of a skeletal line have a crossing number of 4; points at the end of a line have crossing number 2; points at skeletal "T"

junctions have a crossing number of 6; and points at skeletal "X" junctions have a crossing number of 8. However, there is a situation to beware of—places which look like a "+" junction:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{matrix}$$

In such places, the crossing number is actually 0 (see formula), although the pattern is definitely that of a cross. At first, the situation seems to be that there is insufficient resolution in a 3×3 neighborhood to identify a "+" cross, the best option being to look for this particular pattern of 0s and 1s and use a more sophisticated construct than the 3×3 crossing number to check whether or not a cross is present. The problem is that of distinguishing between two situations such as:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \quad \text{and} \quad \begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{matrix}$$

However, further analysis shows that the first of these two cases would be thinned down to a dot (or a short bar), so that if a "+" node appears on the final skeleton (as in the second case), it actually signifies that a cross is present despite the contrary value of χ. Davies and Celano (1993) have shown that the proper measure to use in such cases is the *modified* crossing number $\chi_{\text{skel}} = 2\sigma$, this crossing number being different from χ because it is required not to test whether points can be eliminated from the skeleton, but to ascertain the meaning of points that are at that stage *known* to lie on the final skeleton. Note that χ_{skel} can have values as high as 16—it is not restricted to the range 0 to 8!

Finally, note that sometimes insufficient resolution really is a problem, in that a cross with a shallow crossing angle appears as two "T" junctions:

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

Clearly, resolution makes it impossible to recognize an asterisk or more complex figure from its crossing number, within a 3×3 neighborhood. Probably, the best solution is to label junctions tentatively, then to consider all the junction labels in the image, and to analyze whether a particular local combination of junctions should be reinterpreted—e.g., two "T" junctions may be deduced to form a cross. This is especially important in view of the distortions that appear on a skeleton in the region of a junction (see Section 8.6.4).

8.6.6 APPLICATION OF SKELETONS FOR SHAPE RECOGNITION

Shape analysis may be carried out simply and conveniently by analysis of skeleton shapes and dimensions. Clearly, study of the nodes of a skeleton (points for which there are other than two skeletal neighbors) can lead to the classification of simple shapes but not, for example, discrimination of all block capitals from each other. Many classification schemes exist which can complete the analysis, in terms of limb lengths and positions, and methods for achieving this are touched on in later chapters.

A similar situation exists for analysis of the shapes of chromosomes, which take the form of a cross or a "V." For small industrial components, more detailed shape analysis is called for; this can still be approached with the skeleton technique, by examination of distance function values along the lines of the skeleton. In general, shape analysis using the skeleton proceeds by examination in turn of nodes, limb lengths and orientations, and distance function values, until the required level of characterization is obtained.

The particular importance of the skeleton as an aid in the analysis of connected shapes is not only that it is invariant under translations and rotations but also that it embodies what is for many purposes a highly convenient representation of the figure which (with the distance function values) essentially carries all the original information. If the original shape of an object can be deduced exactly from a representation, this is generally a good sign as it means that it is not merely an ad hoc descriptor of shape but that considerable reliance may be placed on it (compare other methods such as the circularity measure—see Section 8.7).

8.7 OTHER MEASURES FOR SHAPE RECOGNITION

There are many simple tests of shape that can be made to confirm the identity of objects or to check for items such as defects. These include measurements of product area and perimeter, length of maximum dimension, moments relative to the centroid, number and area of holes, area and dimensions of the convex hull (see below) and enclosing rectangle, number of sharp corners, number of intersections with a check circle and angles between intersections (Fig. 8.12), and numbers and types of skeleton nodes.

The list would not be complete without a mention of the widely used shape measure $C = area/perimeter^2$. This quantity is often called "circularity" or "compactness", as it has a maximum value of $1/4\pi$ for a circle, decreases as shapes become more irregular, and approaches zero for long narrow objects: alternatively, its reciprocal is sometimes used, being called "complexity" as it increases in size as shapes become more complex. Note that both measures are dimensionless so that they are independent of size and are therefore sensitive only to the shape of an object. Other dimensionless measures of this type include rectangularity and aspect ratio.

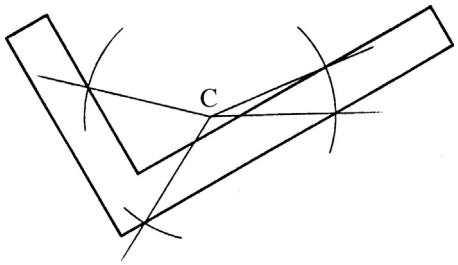

FIGURE 8.12

Rapid product inspection by polar checking.

All these measures have the property of characterizing a shape but not of describing it uniquely. Thus, it is easy to see that there are in general many different shapes having the same values of parameters such as circularity. Hence, these rather ad hoc measures are on the whole less valuable than approaches such as skeletonization (Section 8.6) or moments (see below) that can be used to represent and reproduce a shape to any required degree of precision. Nevertheless, rigorous checking of even one measured number to high precision often permits a machined part to be identified positively.

The use of moments for shape analysis was mentioned above: these are widely used and should be covered in more detail. In fact, moment approximations provide a rigorous means of describing 2-D shapes and take the form of series expansions of the type:

$$M_{pq} = \sum_x \sum_y x^p y^q f(x, y) \tag{8.15}$$

for a picture function $f(x,y)$; such a series may be curtailed when the approximation is sufficiently accurate. By referring axes to the centroid of the shape, moments can be constructed that are position-invariant: they can also be normalized so that they are invariant under rotation and change of scale (Hu, 1962; see also Wong and Hall, 1978). The main value of using moment descriptors is that in certain applications the number of parameters may be made small without significant loss of precision—although the number required may not be clear without tests being made on a range of relevant shapes. Moments can prove particularly valuable in describing shapes such as cams and other fairly round objects, although they have also been used in a variety of other applications including aeroplane silhouette recognition (Dudani et al., 1977).

The convex hull was also mentioned above and has also been used as the basis for sophisticated, complete descriptions of shapes. The *convex hull* is defined as the smallest convex shape that contains the original shape (it may be envisaged as the shape contained by an elastic band placed around the original shape). The *convex deficiency* is defined as the shape that has to be added to a given shape to create the convex hull (Fig. 8.13). The convex hull may be used as

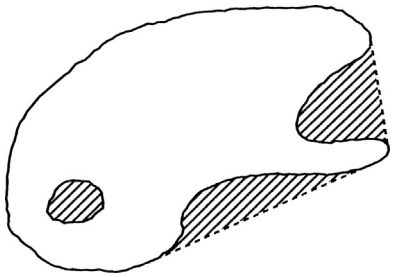

FIGURE 8.13

Convex hull and convex deficiency. The convex hull is the shape enclosed on placing an elastic band around an object. The shaded portion is the convex deficiency that is added to the shape to create the convex hull.

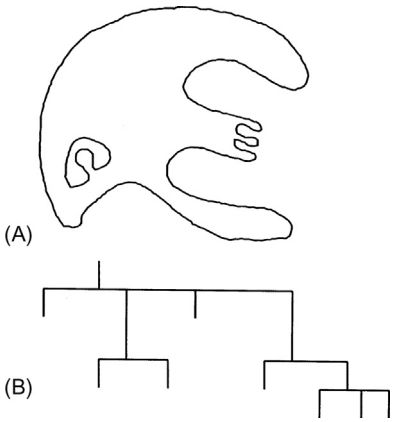

FIGURE 8.14

A simple shape and its concavity tree. The shape in (A) has been analyzed by repeated formation of convex hulls and convex deficiencies until all the constituent regions are convex (see text). The tree representing the entire process as shown in (B): at each node, the branch on the left is the convex hull and the branches on the right are convex deficiencies.

a simple approximation providing a rapid indication of the extent of an object. A fuller description of the shape of an object may be obtained by means of *concavity trees*: here, the convex hull of an object is first obtained with its convex deficiencies, then the convex hulls and deficiencies of the convex deficiencies are found, then the convex hulls and deficiencies of these convex deficiencies—and so on until all the derived shapes are convex, or until an adequate approximation to the original shape is obtained. Thus, a tree is formed which can be used for systematic shape analysis and recognition (Fig. 8.14). We shall not dwell on this

approach beyond noting its inherent utility and that at its core is the need for a reliable means of determining the convex hull of a shape.

A simple strategy for obtaining the convex hull is to repeatedly fill in the center pixel of all neighborhoods that exhibit a concavity, including each of the following:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

until no further change occurs. In fact, the shapes obtained by the above approach are larger than ideal convex hulls and approximate to octagonal (or degenerate octagonal) shapes. Hence, more complex algorithms are required to generate convex hulls, a useful approach involving the use boundary tracking to search for positions on the boundary that have common tangent lines.

8.8 BOUNDARY TRACKING PROCEDURES

The preceding sections have described methods of analyzing shape on the basis of body representations of such as skeletons and moments. However, an important approach has so far been omitted—the use of boundary pattern analysis. This approach has the potential advantage of requiring considerably reduced computation, as the number of pixels to be examined is equal to the number of pixels on the boundary of any object rather than the much larger number of pixels within the boundary. Before proper use can be made of boundary pattern analysis techniques, means must be found for tracking systematically around the boundaries of all the objects in an image: in addition, care must be taken not to ignore any holes that are present or any objects within holes.

In one sense, the problem has been analyzed already, in that the object labeling algorithm of Section 8.3 systematically visits and propagates through all objects in the image. All that is required now is some means of tracking round object boundaries, once they have been encountered. Quite clearly, it will be useful to mark in a separate image space all points that have been tracked: alternatively, an object boundary image may be constructed and the tracking performed in this space, all tracked points being eliminated as they are passed.

In the latter procedure, objects having unit width in certain places may become disconnected. Hence, we ignore this approach and adopt the previous one. There is still a problem when objects have unit-width sections, as these can cause badly designed tracking algorithms to choose a wrong path, going back around the previous section instead of on to the next (Fig. 8.15). To avoid this circumstance, it is best to adopt the following strategy:

1. track round each boundary, keeping to the left path consistently;
2. stop the tracking procedure only when passing through the starting point in the original direction (or passing through the first two points in the same order).

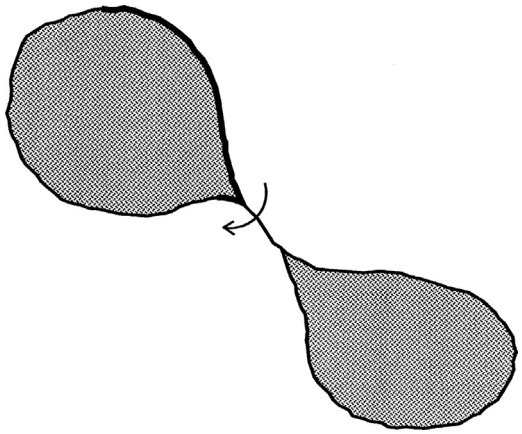

FIGURE 8.15

A problem with an oversimple boundary tracking algorithm: the boundary-tracking procedure takes a short-cut across a unit-width boundary segment instead of continuing and keeping to the left path at all times.

Table 8.12 Basic Procedure for Tracking Around a Single Object

```
do {
    // find direction to move next
    start with current tracking direction;
    reverse it;
    do {
        rotate tracking direction clockwise
    } until the next 1 is met on outer pixels of 3 × 3 neighbourhood;
    record this as new current direction;
    move one pixel along this direction;
    increment boundary index;
    store current position in boundary list;
} until (position == original position) && (direction == original
direction)
```

Apart from necessary initialization at the start, a suitable tracking procedure is given in Table 8.12.

Having seen how to track around the boundaries of objects in binary images, we are now in a position to embark on boundary pattern analysis. This is done in Chapter 9, Boundary Pattern Analysis.

8.9 CONCLUDING REMARKS

This chapter has concentrated on rather traditional methods of performing image analysis—using image processing techniques. This has led naturally to area representations of objects, including for example moment and convex hull-based

schemes, although the skeleton approach appeared as a rather special case in that it converts objects into graphical structures. An alternative schema is to represent shapes by their boundary patterns, after applying suitable tracking algorithms: this latter approach is considered in the following chapter. Meanwhile, connectedness has been an underlying theme in the present chapter, objects being separated from each other by regions of background, thereby permitting objects to be considered and characterized individually. Connectedness has been seen to involve rather more intricate computations than might have been expected, and this necessitates great care in the design of algorithms: this must partly explain why after so many years, new thinning algorithms are still being developed (e.g., Kwok, 1989; Choy et al., 1995) (ultimately, these complexities arise because global properties of images are being computed by purely local means).

Although it will turn out that boundary pattern analysis is in certain ways more attractive than region pattern analysis, this comparison cannot be completely divorced from considerations of the hardware the algorithms have to run on. In this respect, note that many of the algorithms of this chapter can be performed efficiently on SIMD processors, which have one processing element per pixel, whereas boundary pattern analysis will be seen to match better the capabilities of more conventional serial computers.

Shape analysis can be attempted by boundary or region representations. Both are deeply affected by connectedness and related metric issues for a digital lattice of pixels. This chapter has shown that these issues are only solved by carefully incorporating global knowledge alongside local information—e.g., by use of distance transforms.

8.10 BIBLIOGRAPHICAL AND HISTORICAL NOTES

The development of shape analysis techniques has been particularly extensive: hence, only a brief perusal of the history is attempted here. The all-important theory of connectedness and the related concept of adjacency in digital images was developed largely by Rosenfeld (see e.g., Rosenfeld, 1970). The connectedness concept led to the idea of a distance function in a digital picture (Rosenfeld and Pfaltz, 1966, 1968), and the related skeleton concept (Pfaltz and Rosenfeld, 1967). However, the basic idea of a skeleton dates from the classic work by Blum (1967)—see also Blum and Nagel (1978). Important work on thinning has been carried out by Arcelli et al. (1975, 1981; Arcelli and di Baja, 1985) and parallels work by Davies and Plummer (1981). The latter paper demonstrates possibilities for limb pruning, and a rigorous method for testing the results of *any* thinning algorithm, however generated, and in particular for detecting skeletal bias. More recently, Arcelli and Ramella (1995) have reconsidered the problem of skeletons in gray-scale images. There have also been important developments to generalize

the distance function concept and to make distance functions uniform and isotropic: see for example Huttenlocher et al. (1993). The design of a modified crossing number χ_{skel} for the analysis of skeletal shape dates from the same period: as pointed out in Section 8.6.5, χ_{skel} is different from χ as it evaluates the *remaining* (i.e., skeletal) points rather than points that *might* be eliminated from the skeleton (Davies and Celano, 1993).

Sklansky has carried out much work on convexity and convex hull determination (see e.g., Sklansky, 1970; Sklansky et al., 1976), whereas Batchelor (1979) developed concavity trees for shape description. Haralick et al. (1987) have generalized the underlying mathematical (morphological) concepts, including the case of gray-scale analysis. Use of invariant moments for pattern recognition dates from the two seminal papers by Hu (1961, 1962). Pavlidis has drawn attention to the importance of unambiguous ("asymptotic") shape representation schemes (Pavlidis, 1980)—as distinct from ad hoc sets of shape measures.

In the 2000s, skeletons have maintained their interest and utility, becoming if anything more precise by reference to exact analog shapes (Kégl and Krzyżak, 2002), and giving rise to the concept of a shock graph, which characterizes the result of the much earlier grass-fire transformation (Blum, 1967) more rigorously (Giblin and Kimia, 2003). Wavelet transforms have also been used to implement skeletons more accurately in the discrete domain (Tang and You, 2003). In contrast, shape matching has been carried out using self-similarity analysis coupled with tree representation—an approach that has been especially valuable for tracking articulated object shapes, including human profiles and hand motions (Geiger et al., 2003). It is interesting to see graphical analysis of skeletonized hand-written character shapes performed taking account of catastrophe theory (Chakravarty and Kompella, 2003): this is relevant because (1) critical points—where points of inflection exist—can be deformed into pairs of points each corresponding to a curvature maximum plus a minimum; (2) crossing of t's can be actual or noncrossing; and (3) loops can turn into cusps or corners (many other possibilities also exist). The point is that methods are needed for mapping between *variations* of shapes rather than making snap judgments as to classification (this corresponds to the difference between scientific understanding of process and ad hoc engineering).

8.10.1 MORE RECENT DEVELOPMENTS

More recently, increased attention has been devoted to processing skeletons and using them for object matching and classification. Bai and Latecki (2008) discuss how to prune skeletons meaningfully, by ensuring that endpoints of skeleton branches correspond to visual parts of objects (such as all the legs of a horse). Once this has been achieved, it should be possible to match objects (such as horses) in spite of any articulations or contour deformations that may have taken place. The method is found to permit much more efficient matching and to be

more resistant to partial occlusion: this is because meaningfulness is built into the final skeleton, whereas minor intricacies (which may originally have been due to noise forming tiny holes in the object) will have been eliminated. This approach is potentially useful for tracking, stereo matching, and database matching. Ward and Hamarneh (2010) attend to the order in which skeleton branches should be pruned. They report on several pruning algorithms and quantify their performance in terms of denoizing, classification, and within-class skeleton similarity measures. The work is important because of the well-known fact that the medial axis transform is unstable with respect to minor perturbations on the boundary of a shape: this means that before skeletons can be used reliably, noise spurs need to be pruned so that they correspond to the underlying shapes.

8.11 PROBLEMS

1. Write the full C++ routine required to sort the lists of labels, to be inserted at the end of the algorithm of Table 8.2.
2. Show that, as stated in Section 8.6.2 for a parallel thinning algorithm, all north points may be removed in parallel without any risk of causing a break in the skeleton.
3. Describe methods for locating, labeling, and counting objects in binary images. You should consider whether methods based on propagation from a "seed" pixel, or those based on progressively shrinking a skeleton to a point, would provide the more efficient means for achieving the stated aims. Give examples for objects of various shapes.
4. **a.** Give a simple one-pass algorithm for labeling the objects appearing in a binary image, making clear the role played by connectedness. Give examples showing how this basic algorithm goes wrong with real objects: illustrate your answer with clear pixel diagrams, which show the numbers of labels that can appear on objects of different shapes.
   b. Show how a table-orientated approach can be used to eliminate multiple labels in objects. Make clear how the table is set up and what numbers have to be inserted into it. Are the number of iterations needed to analyze the table similar to the number that would be needed in a multipass labeling algorithm taking place entirely within the original image? Consider how the *real* gain in using a table to analyze the labels arises.
5. **a.** Using the following notation for a 3×3 window:

A4	A3	A2
A5	A0	A1
A6	A7	A8

work out the effect of the following algorithm on a binary image containing small foreground objects of various shapes:

```
do {
  for all pixels in image do {
    sum = (int)(A1 + A3 == 2) + (int)(A3 + A5 == 2)
        + (int)(A5 + A7 == 2) + (int)(A7 + A1 == 2);
    if (sum > 0) B0 = 1; else B0 = A0;
  }
  for all pixels in image do {A0 = B0;}
} until no further change;
```

b. Show in detail how to implement the *do . . . until no further change* function in this algorithm.

6. a. Give a simple algorithm operating in a 3 × 3 window for generating a *rectangular* convex hull around each object in a binary image. Include in your algorithm any necessary code for implementing the required *do . . . until no further change* function.

b. A more sophisticated algorithm for finding accurate convex hulls is to be designed. Explain why this would employ a boundary tracking procedure. State the general strategy of an algorithm for tracking around the boundaries of objects in binary images and write a program for implementing it.

c. Suggest a strategy for designing the complete convex hull algorithm and indicate how rapidly you would expect it to operate, in terms of the size of the image.

7. a. Explain the meaning of the term *distance function*. Give examples of the distance functions of simple shapes, including that shown in Fig. 8.P1.

b. Rapid image transmission is to be performed by sending only the coordinates and values of the local maxima of the distance functions. Give a complete algorithm for finding the local maxima of the distance

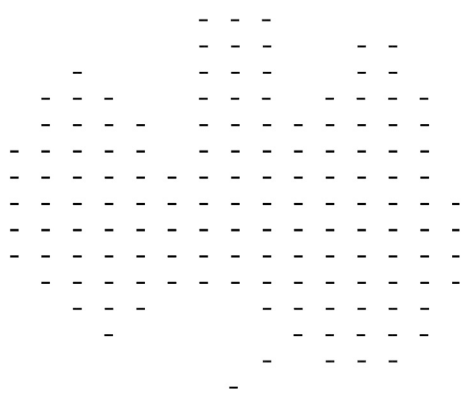

FIGURE 8.P1

Binary picture object for shape analysis tests.

functions in an image and devise a further algorithm for reconstructing the original binary image.

c. Discuss which of the following sets of data would give more compressed versions of the binary picture object shown in Fig. 8.P1:

i. the list of local maxima coordinates and values;

ii. a list of the coordinates of the boundary points of the object;

iii. a list consisting of one point on the boundary and the relative directions (each expressed as a 3-bit code) between each *pair* of boundary points on tracking round the boundary.

8. a. What is the *distance function* of a binary image? Illustrate your answer for the case where a 128 $\times$ 128 image P contains just the object shown in Fig. 8.P2. How many passes of (1) a parallel algorithm and (2) a sequential algorithm would be required to find the distance function of the image?

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
                        1 1 1 1 1 1 1
                        1 1 1 1 1 1 1
                        1 1 1 1 1 1 1
                        1 1 1 1 1 1 1
                        1 1 1 1 1 1 1
                        1 1 1 1 1 1 1 1 1
```

FIGURE 8.P2

Binary picture object for distance function analysis.

b. Give a complete parallel *or* sequential algorithm for finding the distance function and explain how it operates.

c. Image P is to be transmitted rapidly by determining the coordinates of locations where the distance function is locally maximum. Indicate the positions of the local maxima, and explain how the image could be reconstituted at the receiver.

d. Determine the compression factor η if image P is transmitted in this way. Show that η can be increased by eliminating some of the local maxima before transmission and estimate the resulting increase in η.

9. a. The local maxima of the distance function can be defined in the following two ways:

i. those pixels whose values are greater than those of all the neighboring pixels;

ii. those pixels whose values are greater than *or equal to* the values of all the neighboring pixels. Which definition of the local maxima would be more useful for reproducing the original object shapes? Why is this?

b. Give an algorithm that is capable of reproducing the original object shapes from the local maxima of the distance function and explain how it operates.

c. Explain the *run-length-encoding* approach to image compression. Compare the run-length encoding and local maxima methods for compressing binary images. Explain why the one method would be expected to lead to a greater degree of compression with some types of image, whereas the other method would be expected to be better with other types of image.

10. a. Explain how propagation of a distance function may be carried out using a parallel algorithm. Give in full a simpler algorithm that operates using two sequential passes over the image.

b. It has been suggested that a four-pass sequential algorithm will be even faster than the two-pass algorithm, as each pass can use just a 1-D window involving at most three pixels. Write down the code for *one* typical pass of the algorithm.

c. Estimate the approximate speeds of these three algorithms for computing the distance function, in the case of an $N \times N$ pixel image. Assume a conventional serial computer is used to perform the computation.

11. Small dark insects are to be located amongst cereal grains. The insects approximate to rectangular bars of dimensions 20 by 7 pixels, and the cereal grains are approximately elliptical with dimensions 40 by 24 pixels. The proposed algorithm design strategy is: (1) apply an edge detector which will mark all the edge points in the image as s in a 1s background, (2) propagate a distance function in the *background* region, (3) locate the local maxima of the distance function, (4) analyze the values of the local maxima, and (5) carry out necessary further processing to identify the nearly parallel sides of the insects. Explain how to design stages (4) and (5) of the algorithm in order to identify the insects, ignoring the cereal grains. Assume that the image is not so large that the distance function will overflow the byte limit. Determine how robust the method will be if the edge is broken in a few places.

12. Give the general strategy of an algorithm for tracking around the boundaries of objects in binary images. If the tracker has reached a boundary point with crossing number $\chi = 2$ and neighborhood $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, decide in which direction it should now proceed. Hence, give a complete procedure for determining the direction code of the next position on the boundary for cases where $\chi = 2$. Take the direction codes starting *from* the current pixel (*) as being specified by the following diagram:

$$\begin{bmatrix} 4 & 3 & 2 \\ 5 & * & 1 \\ 6 & 7 & 8 \end{bmatrix}$$

How should the procedure be modified for cases where $\chi \neq 2$?

13. a. Explain the principles involved in tracking around the boundaries of objects in binary images to produce reliable outlines. Outline an algorithm which can be used for this purpose, assuming it is to get its information from a normal 3×3 window.

b. A binary image is obtained and the data in it is to be compressed as much as possible. The following range of algorithms is to be tested for this purpose:

i. the boundary image;
ii. the skeleton image;
iii. the image of the local maxima of the distance function;
iv. the image of a suitably chosen subset of the local maxima of the distance function;
v. a set of run-length data, i.e., a series of numbers obtained by counting runs of 0s, then runs of 1s, then runs of 0s, etc., in a continuous line-by-line scan over the image.

c. In (i) and (ii), the lines may be encoded using chain code, i.e., giving the *coordinates* of the first point met, and the *direction* of each subsequent point using the direction codes 1 to 8 defined relative to the current position C by:

$$\begin{array}{ccc} 4 & 3 & 2 \\ 5 & C & 1 \\ 6 & 7 & 8 \end{array}$$

d. With the aid of suitably chosen examples, discuss which of the methods of data compression should be most suitable for different types of data. Wherever possible, give numerical estimates to support your arguments.

e. Indicate what you would expect to happen if noise were added to initially noise-free input images.

14. Test the two-mask strategy outlined in Section 8.7 for obtaining the convex hulls of binary picture objects. Confirm that it operates consistently and give a geometrical construction that predicts the final shapes it produces. What happens if either the first or the second mask is used on its own? Show that the two-mask strategy will operate both as a sequential and as a parallel algorithm. Devise a version of the algorithm that does not permit nearby shapes to be merged.