19 Shape Presentation and Analysis

19.1 Introduction

All operations discussed in Chapters 11–15 extracted features from images that are represented as images again. Even the morphological operations discussed in Chapter 18 that analyze and modify the shape of segmented objects in binary images work in this way. It is obvious, however, that the shape of objects can be represented in a much more compact form. All information on the shape of an object is, for example, contained in its boundary pixels.

In Section 19.2, we therefore address the question of how to represent a segmented object. We will study the representation of binary objects with the *run-length code* (Section 19.2.1), the *quadtree* (Section 19.2.2), and the *chain code* (Section 19.2.3). Two further object representations, *moments* and *Fourier descriptors*, are of such significance that we devote entire sections to them (Sections 19.3 and 19.4).

A compact representation for the shape of objects is not of much use if it takes a lot of effort to compute it and if it is cumbersome to compute shape parameters directly from it. Therefore we address also the question of shape parameter extraction from the different shape representations in Section 19.5.

Shape parameters are extracted from objects in order to describe their shape, to compare it to the shape of template objects, or to partition objects into classes of different shapes. In this respect the important questions arises how shape parameters can be made invariant on certain transformations. Objects can be viewed from different distances and from different points of view. Thus it is of interest to find shape parameters that are scale and rotation invariant or that are even invariant under affine or perspective projection.

19.2 Representation of Shape

19.2.1 Run-Length Code

A compact, simple, and widely used representation of an image is *run-length code*. The run-length code is produced by the following procedure. An image is scanned line by line. If a line contains a sequence of p equal

B. Jähne, Digital Image Processing
ISBN 3-540-24035-7

a) ***Gray value image***

Original line (hex): 12 12 12 20 20 20 20 25 27 25 20 20 20 20 20 20

Code (hex): 82 **12** 83 **20** 2 **25 27 25** 85 **20**

b) ***Binary image***

Original line (hex): 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 1

Code (hex) 0 6 3 3 2 1 5 8

Figure 19.1: *Demonstration of the run-length code for* ***a*** *gray value image,* ***b*** *binary image.*

pixels, we do not store p times the same value, but store the pixel value once and indicate that it occurs p times (Fig. 19.1). In this way, large uniform line segments can be stored very efficiently.

For binary images, the code can be especially efficient as only the two pixel values zero and one occur. As a sequence of zeroes is always followed by a sequence of ones, there is no need to store the pixel value. We only need to store the number of times a pixel value occurs (Fig. 19.1b). We must be careful at the beginning of a line, however, as it may begin with a one or a zero. This problem can be resolved if we assume a line to begin with zero. If a line should start with a sequence of ones, we start the run-length code with a zero to indicate that the line begins with a sequence of zero zeroes (Fig. 19.1b).

Run-length code is suitable for compact storage of images. It has become an integral part of several standard image formats, for example, the TGA or the *TIFF* file formats. Run-length code is less useful for direct processing of images, however, because it is not object oriented. As a result, run-length encoding is more useful for compact image storage. Not all types of images can be successfully compressed with this scheme. Digitized gray-scale images, for example, always contain some noise so that the probability for sufficiently long sequences of pixels with the same gray value is very low. However, high data reduction factors can be achieved with binary images and many types of computer-generated gray-scale and color images.

19.2.2 Quadtrees

The run-length codes discussed in Section 19.2.1 are a line-oriented representation of binary images. Thus they encode one-dimensional rather than two-dimensional data. The two-dimensional structure is actually not considered at all. In contrast, a *quadtree* is based on the principle of recursive decomposition of space, as illustrated in Fig. 19.2 for a binary image.

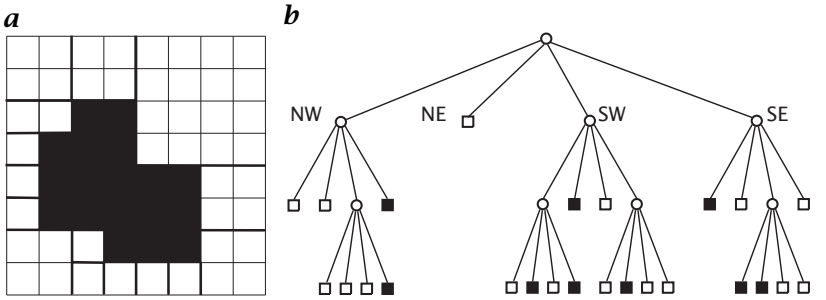


Figure 19.2: *Representation of a binary image by a region quadtree:* ***a*** *successive subdivision of the binary image into quadrants;* ***b*** *the corresponding region quadtree.*

First, the whole image is decomposed into four equal-sized *quadrants*. If one of the quadrants does not contain a uniform region, i. e., the quadrant is not included entirely in the object or background, it is again subdivided into four subquadrants. The decomposition stops if only uniform quadrants are encountered or if the quadrants finally contain only one pixel.

The recursive decomposition can be represented in a data structure known in computer science as *tree* (Fig. 19.2b). At the top level of the tree, the *root*, the decomposition starts. The root corresponds to the entire binary image. It is connected via four edges to four *child nodes* which represent from left to right the NW, NE, SW, and SE quadrants. If a quadrant needs no further subdivision, it is represented by a *terminal node* or a *leaf node* in the tree. It is called black when the quadrant belongs to an object and white otherwise, indicated by a filled and open square, respectively. Nonleaf nodes require further subdivision and are said to be gray and shown as open circles (Fig. 19.2b).

Quadtrees can be encoded, for example, by a *depth-first traversal* of the tree starting at the root. It is only required to store the type of the node with the symbols b (black), w (white), and g (gray). We start the code with the value of the root node. Then we list the values of the child nodes from left to right. Each time we encounter a gray node, we continue encoding at one level lower in the tree. This rule is applied recursively. This means that we return to a higher level in the tree only after the visited branch is completely encoded down to the lowest level. This is why this encoding is named depth-first.

The example quadtree shown in Fig. 19.2b results in the code

$$g\,g\,w\,w\,g\,w\,w\,w\,b\,b\,w\,g\,g\,w\,b\,w\,b\,b\,w\,g\,w\,b\,w\,w\,g\,b\,w\,g\,b\,b\,w\,w\,w\,.$$

The code becomes more readable if we include a left parenthesis each time we descend one level in the tree and a right parenthesis when we

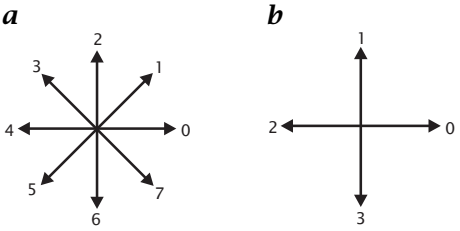


Figure 19.3: *Direction coding in* ***a*** *an 8-neighborhood and* ***b*** *a 4-neighborhood.*

ascend again

$$g(g(wwg(wwwb)b)wg(g(wbwb)bwg(wbww))g(bwg(bbww)w)).$$

However, the code is unique without the parentheses. A quadtree is a compact representation of a binary image if it contains many leaf nodes at high levels. However, in the worst case, for example a regular checkerboard pattern, all leaf nodes are at the lowest level. The quadtree then contains as many leaf nodes as pixels and thus requires many more bytes of storage space than the direct representation of the binary image as a matrix.

The region quadtree discussed here is only one of the many possibilities for recursive spatial decomposition. Three-dimensional binary images can be recursively decomposed in a similar way. The 3-D image is subdivided into eight equally sized octants. The resulting data structure is called a region *octree*. Quadtrees and octrees have gained significant importance in geographic information systems and computer graphics.

Quadtrees are a more suitable encoding technique for images than the line-oriented run-length code. But they are less suitable for image analysis. It is rather difficult to perform shape analysis directly on quadtrees. Without going into further details, this can be seen from the simple fact that an object shifted by one pixel in any direction results in a completely different quadtree. Region quadtrees share their most important disadvantage with run-length code: the technique is a global image decomposition and not one that represents objects extracted from images in a compact way.

19.2.3 Chain Code

In contrast to run-length code and quadtrees, *chain code* is an object-related data structure for representing the boundary of a binary object effectively on a discrete grid. Instead of storing the positions of all the boundary pixels, we select a starting pixel and store only its coordinate. If we use an algorithm that scans the image line by line, this will be the uppermost left pixel of the object. Then we follow the boundary

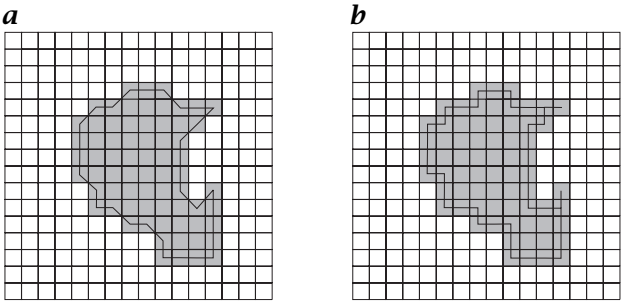


Figure 19.4: *Boundary representation with the chain code:* ***a*** *8-neighborhood;* ***b*** *4-neighborhood.*

in a clockwise direction. In a 4-neighborhood there are 4 and in an 8-neighborhood 8 possible directions to go, which we can encode with a 3-bit or 2-bit code as indicated in Fig. 19.3. Extracted boundaries are shown in Fig. 19.4 for a 4-neighborhood and an 8-neighborhood.

The chain code shows a number of obvious advantages over the matrix representation of a binary object:

First, the chain code is a compact representation of a binary object. Let us assume a disk-like object with a diameter of R pixels. In a direct matrix representation we need to store the *bounding box* of the object (see Section 19.5.4), i. e., about R^2 pixels which are stored in R^2 bits. The bounding rectangle is the smallest rectangle enclosing the object. If we use an 8-connected boundary, the disk has about πR boundary points. The chain code of the πR points can be stored in about $3\pi R$ bits. For objects with a diameter larger than 10, the chain code is a more compact representation.

Second, the chain code is a *translation invariant* representation of a binary object. This property makes it easier to compare objects. However, the chain code is neither rotation nor scale invariant. This is a significant disadvantage for object recognition, although the chain code can still be used to extract rotation invariant parameters, such as the area of the object.

Third, the chain code is a complete representation of an object or curve. Therefore, we can — at least in principle — compute any shape feature from the chain code.

As shown in Section 19.5, we can compute a number of shape parameters — including the perimeter and area — more efficiently using the chain-code representation than in the matrix representation of the binary image. The limitation here is, of course, that the chain code is a digital curve on a discrete grid and as such describes the boundary of the object only within the precision of the discrete grid.

If the object is not connected or if it has holes, we need more than one chain code to represent it. We must also include information on whether the boundary surrounds an object or a hole. Reconstruction of the binary image from a chain code is an easy procedure: we can draw the outline of the object and then use a *fill operation* to paint it.

19.3 Moment-Based Shape Features

19.3.1 Definitions

In this section we present a systematic approach to object shape description. We first define *moments* for gray value and binary images and then show how to extract useful shape parameters from this approach. We will discuss Fourier descriptors in a similar manner in Section 19.4.

We used moments in Section 3.2.2 to describe the probability density function for gray values. Here we extend this description to two dimensions and define the moments of the gray value function $g(\boldsymbol{x})$ of an object as

$$\mu_{p,q} = \int (x_1 - \overline{x_1})^p (x_2 - \overline{x_2})^q \, g(\boldsymbol{x}) \mathrm{d}^2 x, \tag{19.1}$$

where

$$\overline{x_i} = \int x_i g(\boldsymbol{x}) \mathrm{d}^2 x \Big/ \int g(\boldsymbol{x}) \mathrm{d}^2 x. \tag{19.2}$$

The integration includes the area of the object. Instead of the gray value, we may use more generally any pixel-based feature to compute object moments. The vector $\overline{\boldsymbol{x}} = (\overline{x_1}, \overline{x_2})$ is called the *center of mass* of the object by analogy to classical mechanics. Think of $g(\boldsymbol{x})$ as the density $\rho(\boldsymbol{x})$ of the object; then the zero-order moment $\mu_{0,0}$ becomes the total mass of the object.

All the moments defined in Eq. (19.1) are related to the center of mass. Therefore they are often denoted as *central moments*. Central moments are translation invariant and thus are useful features for describing the shape of objects.

For discrete binary images, the moment calculation reduces to

$$\mu_{p,q} = \sum (x_1 - \overline{x_1})^p (x_2 - \overline{x_2})^q. \tag{19.3}$$

The summation includes all pixels belonging to the object. For the description of object shape we may use moments based on either binary, gray scale or feature images. Moments based on gray scale or feature images reflect not only the geometrical shape of an object but also the distribution of features within the object. As such, they are generally different from moments based on binary images.

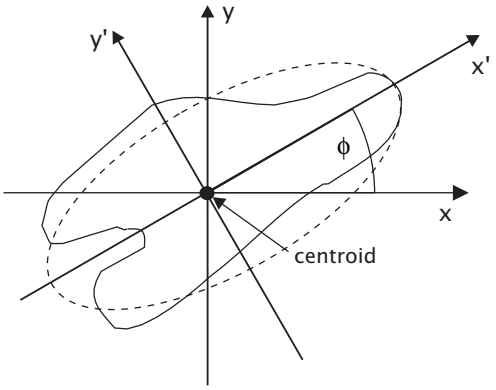


Figure 19.5: *Principal axes of the inertia tensor of an object for rotation around the center of mass.*

19.3.2 Scale-Invariant Moments

Often it is necessary to use shape parameters that do not depend on the size of the object. This is always required if objects observed from different distances must be compared. Moments can be normalized in the following way to obtain *scale-invariant* shape parameters. If we scale an object $g(\boldsymbol{x})$ by a factor of α, $g'(\boldsymbol{x}) = g(\boldsymbol{x}/\alpha)$, its moments are scaled by

$$\mu'_{p,q} = \alpha^{p+q+2}\,\mu_{p,q}.$$

We can then normalize the moments with the zero-order moment, $\mu_{0,0}$, to gain *scale-invariant moments*

$$\bar{\mu} = \frac{\mu_{p,q}}{\mu_{0,0}^{(p+q+2)/2}}.$$

Because the zero-order moment of a binary object gives the area of the object (Eq. (19.3)), the normalized moments are scaled by the area of the object. Second-order moments ($p + q = 2$), for example, are scaled with the square of the area.

19.3.3 Moment Tensor

Shape analysis beyond area measurements starts with the second-order moments. The zero-order moment just gives the area or "total mass" of a binary or gray value object, respectively. The first-order central moments are zero by definition.

The analogy to mechanics is again helpful to understand the meaning of the second-order moments $\mu_{2,0}$, $\mu_{0,2}$, and $\mu_{1,1}$. They contain terms in which the gray value function, i. e., the density of the object, is multiplied

by squared distances from the center of mass. Exactly the same terms are also included in the inertia tensor that was discussed in Section 13.5.1 (see Eqs. (13.62) and (13.63)). The three second-order moments form the components of the *inertia tensor* for rotation of the object around its center of mass:

$$J = \begin{bmatrix} \mu_{2,0} & -\mu_{1,1} \\ -\mu_{1,1} & \mu_{0,2} \end{bmatrix}. \tag{19.4}$$

Because of this analogy, we can transfer all the results from Section 13.3 to shape description with second-order moments. The *orientation* of the object is defined as the angle between the x axis and the axis around which the object can be rotated with minimum inertia. This is the eigenvector of the minimal eigenvalue. The object is most elongated in this direction (Fig. 19.5). According to Eq. (13.12), this angle is given by

$$\theta = \frac{1}{2} \arctan \frac{2\mu_{1,1}}{\mu_{2,0} - \mu_{0,2}}. \tag{19.5}$$

As a measure for the *eccentricity* ε, we can use what we have defined as a coherence measure for local orientation Eq. (13.15):

$$\varepsilon = \frac{(\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2}{(\mu_{2,0} + \mu_{0,2})^2}. \tag{19.6}$$

The eccentricity ranges from 0 to 1. It is zero for a circular object and one for a line-shaped object. Thus, it is a better-defined quantity than circularity with its odd range (Section 19.5.3).

Shape description by second-order moments in the *moment tensor* essentially models the object as an *ellipse*. The combination of the three second-order moments into a tensor nicely results in two rotation-invariant terms, the trace of the tensor, or $\mu_{2,0} + \mu_{0,2}$, which gives the radial distribution of features in the object, and the eccentricity Eq. (19.6), which measures the roundness, and one term which measures the orientation of the object. Moments allow for a complete shape description [163]. The shape description becomes more detailed the more higher-order moments are used.

19.4 Fourier Descriptors

19.4.1 Cartesian Fourier Descriptors

Fourier descriptors, like the chain code, use only the boundary of the object. In contrast to the chain code, Fourier descriptors do not describe curves on a discrete grid. They can be formulated for continuous or sampled curves. Consider the closed boundary curve sketched in Fig. 19.6. We can describe the boundary curve in a parametric description by taking the path length p from a starting point $[x_0, y_0]^T$ as a parameter.

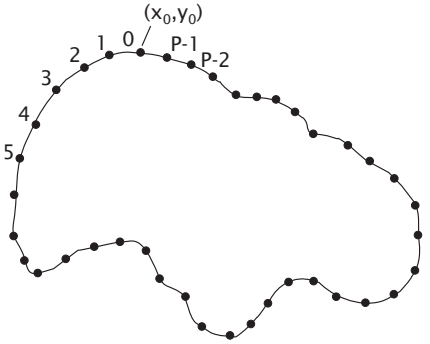


Figure 19.6: *Illustration of a parametric representation of a closed curve. The parameter p is the path length from the starting point* $[x_0, y_0]^T$ *in the counterclockwise direction. An equidistant sampling of the curve with P points is also shown.*

It is not easy to generate a boundary curve with equidistant samples. Discrete boundary curves, like the chain code, have significant disadvantages. In the 8-neighborhood, the samples are not equidistant. In the 4-neighborhood, the samples are equidistant, but the boundary is jagged because the pieces of the boundary curve can only go in horizontal or vertical directions. Therefore, the perimeter tends to be too long. Consequently, it does not seem a good idea to form a continuous boundary curve from points on a regular grid. The only alternative is to extract subpixel-accurate object boundary curves directly from the gray scale images. But this is not an easy task. Thus, the accurate determination of Fourier descriptors from contours in images still remains a challenging research problem.

The continuous boundary curve is of the form $x(p)$ and $y(p)$. We can combine these two curves into one curve with the complex function $z(p) = x(p) + \mathrm{i}y(p)$. This curve is cyclic. If P is the perimeter of the curve, then

$$z(p + nP) = z(p) \quad n \in \mathbb{Z}. \tag{19.7}$$

A cyclic or periodic curve can be expanded in a *Fourier series* (see also Table 2.1). The coefficients of the Fourier series are given by

$$\hat{z}_v = \frac{1}{P} \int_0^P z(p) \exp\left(\frac{-2\pi \mathrm{i} v p}{P}\right) \mathrm{d}p \quad v \in \mathbb{Z}. \tag{19.8}$$

The periodic curve can be reconstructed from the Fourier coefficients by

$$z(p) = \sum_{v=-\infty}^{\infty} \hat{z}_v \exp\left(\frac{2\pi \mathrm{i} v p}{P}\right). \tag{19.9}$$

The coefficients $\hat{z}_v$ are known as the *Cartesian Fourier descriptors* of the boundary curve. Their meaning is straightforward. The first coefficient

$$\hat{z}_0 = \frac{1}{P}\int_0^P z(p)\mathrm{d}p = \frac{1}{P}\int_0^P x(p)\mathrm{d}p + \frac{\mathrm{i}}{P}\int_0^P y(p)\mathrm{d}p \tag{19.10}$$

gives the mean vortex or *centroid* of the boundary. The second coefficient describes a circle

$$z_1(p) = \hat{z}_1 \exp\left(\frac{2\pi\mathrm{i}p}{P}\right) = r_1 \exp\left(\mathrm{i}\varphi_1 + 2\pi\mathrm{i}p/P\right). \tag{19.11}$$

The radius r_1 and the starting point at an angle φ_1 are given by $\hat{z}_1 = r_1 \exp(\mathrm{i}\varphi_1)$. The coefficient $\hat{z}_{-1}$ also results in a circle

$$z_{-1}(p) = r_{-1} \exp\left(\mathrm{i}\varphi_{-1} - 2\pi\mathrm{i}p/P\right), \tag{19.12}$$

but this circle is traced in the opposite direction (clockwise). With both complex coefficients together — in total four parameters — an ellipse can be formed with arbitrary half-axes a and b, orientation ϑ of the main axis a, and starting angle φ_0 on the ellipses. As an example, we take $\varphi_1 = \varphi_{-1} = 0$. Then,

$$z_1 + z_{-1} = (r_1 + r_{-1}) \cdot \cos\left(\frac{2\pi p}{P}\right) + \mathrm{i}(r_1 - r_{-1}) \sin\left(\frac{2\pi p}{P}\right). \tag{19.13}$$

This curve has the parametric form of an ellipse where the axes lie along the coordinate axes and the starting point is on the x axis.

From this discussion it is obvious that Fourier descriptors must always be paired. The pairing of higher-order coefficients also results in ellipses. These ellipses, however, are cycled n times. Added to the basic ellipse of the first pair, this means that the higher-order Fourier descriptors add more and more details to the boundary curve.

For further illustration, the reconstruction of the letters T and L is shown with an increasing number of Fourier descriptors (Fig. 19.7). The example shows that only a few coefficients are required to describe even quite complex shapes.

Fourier descriptors can also be computed easily from sampled boundaries z_n. If the perimeter of the closed curve is P, N samples must be taken at equal distances of P/N (Fig. 19.6). Then,

$$\hat{z}_v = \frac{1}{N}\sum_{n=0}^{N-1} z_n \exp\left(-\frac{2\pi\mathrm{i}nv}{N}\right). \tag{19.14}$$

All other equations are valid also for sampled boundaries. The sampling has just changed the Fourier series to a discrete Fourier transform with only N wave number coefficients that run from 0 to $N-1$ or from $-N/2$ to $N/2-1$ (see also Table 2.1).

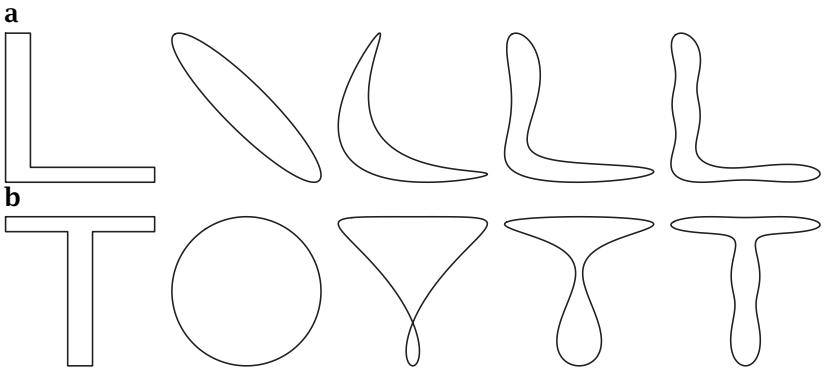


Figure 19.7: *Reconstruction of shape of* ***a*** *the letter "L" and* ***b*** *the letter "T" with 2, 3, 4, and 8 Fourier descriptor pairs.*

19.4.2 Polar Fourier Descriptors

An alternative approach to Fourier descriptors uses another parameterization of the boundary line. Instead of the path length p, the angle θ between the radius drawn from the centroid to a point on the boundary and the x axis is used. Thus, we directly describe the *radius* of the object as a function of the angle. Now we need only a real-valued sequence, $\boldsymbol{r}$, with N equiangular samples to describe the boundary. The coefficients of the discrete Fourier transform of this sequence,

$$\hat{r}_v = \frac{1}{N}\sum_{n=0}^{N-1} r_n \exp\left(-\frac{2\pi i n v}{N}\right), \tag{19.15}$$

are known as the *polar Fourier descriptors* of the boundary. Here, the first coefficient, $\hat{r}_0$, is equal to the mean radius. Polar Fourier descriptors cannot be used for all types of boundaries. The radial boundary parameterization $r(\theta)$ must be single-valued. Because of this significant restriction, we focus the further discussion of Fourier descriptors on Cartesian Fourier descriptors.

19.4.3 Symmetric Objects

Symmetries can easily be detected in Fourier descriptors. If a contour has *m-rotational symmetry*, then only $z_{1\pm vm}$ can be unequal to zero. This is demonstrated in Fig. 19.8 with the Fourier descriptors of a vertical line, a triangle, and a square. If one contour is the mirror contour of another, their Fourier descriptors are conjugate complex to each other.

The Fourier descriptors can also be used for *non-closed boundaries*. To make them closed, we simply trace the curve backward and forward. It is easy to recognize such curves, as their area is zero. From Eq. (19.17),

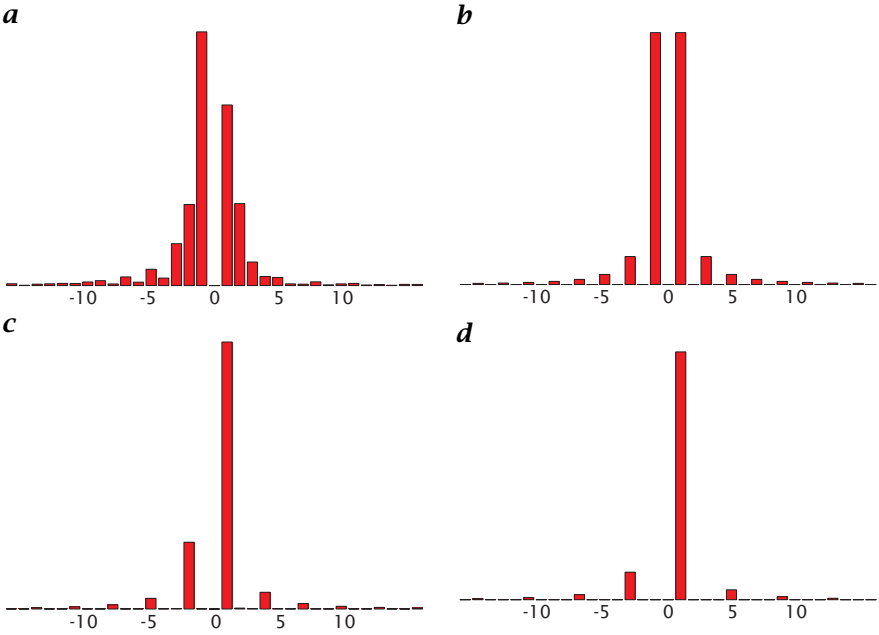


Figure 19.8: *Influence of the symmetry of an object on its Fourier descriptors:* ***a*** *the letter L,* ***b*** *a line,* ***c*** *a triangle, and* ***d*** *a square. Shown are the magnitude of the Fourier descriptors from* $\nu = -16$ *to* $\nu = 16$.

we can then conclude that $|\hat{z}_{-\nu}| = |\hat{z}_\nu|$. If the trace begins at one of the endpoints, even $\hat{z}_{-\nu} = \hat{z}_\nu$.

19.4.4 Invariant Object Description

Translation invariance. The position of the object is confined to a single coefficient $\hat{z}_0$. All other coefficients are translation invariant.

Scale invariance. If the contour is scaled by a coefficient α, all Fourier descriptors are also scaled by α. For an object with non-zero area, and if the contour is traced counterclockwise, the first coefficient is always unequal to zero. Thus, we can simply scale all Fourier descriptors by $|\hat{z}_1|$ to obtain scale invariant shape descriptors. Note that these scaled descriptors are still complete.

Rotation invariance. If a contour is rotated counter clockwise by the angle φ_0, the Fourier descriptor $\hat{z}_\nu$ is multiplied by the phase factor $\exp(\mathrm{i}\nu\varphi_0)$ according to the *shift theorem* for the Fourier transform (Theorem 2.3, p. 54, $\succ$R4). The shift theorem makes the construction of rotation-invariant Fourier descriptors easy. For example, we can relate

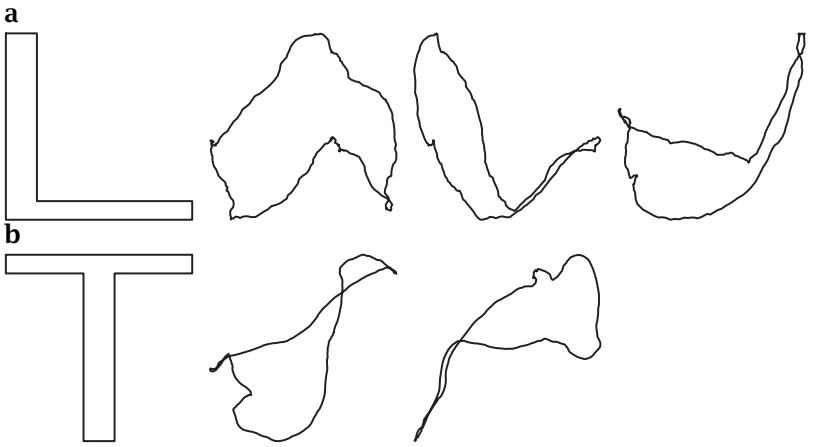


Figure 19.9: *Importance of the phase for the description of shape with Fourier descriptors. Besides the original letters, three random modifications of the phase are shown with unchanged magnitude of the Fourier descriptors.*

the phases of all Fourier descriptors to the phase of $\hat{z}_1$, φ_1, and subtract the phase shift $v\varphi_1$ from all coefficients. Then, all remaining Fourier descriptors are rotation invariant.

Both Fourier descriptors (Section 19.4) and moments (Section 19.3) provide a framework for scale and rotation invariant shape parameters. The Fourier descriptors are the more versatile instrument. However, they restrict the object description to the boundary line while moments of gray scale objects are sensitive to the spatial distribution of the gray values in the object.

Ideally, form parameters describe the form of an object completely and uniquely. This means that different shapes must not be mapped onto the same set of features. A scale and rotation invariant but incomplete shape description is given by the magnitude of the Fourier descriptors. Figure 19.9 shows how different shapes are mapped onto this shape descriptor by taking the Fourier descriptors of the letters "T" and "L" and changing the phase randomly.

Only the complete set of the Fourier descriptors constitutes a unique shape description. Note that for each invariance, one degree of freedom is lost. For translation invariance, we leave out the first Fourier descriptor $\hat{z}_0$ (two degrees of freedom). For scale invariance, we set the magnitude of the second Fourier descriptor, $\hat{z}_1$, to one (one degree of freedom), and for rotation invariance, we relate all phases to the phase of $\hat{z}_1$ (another degree of freedom). With all three invariants, four degrees of freedom are lost.

It is the beauty of the Fourier descriptors that these invariants are simply contained in the first two Fourier descriptors. If we norm all

other Fourier descriptors with the phase and magnitude of the second Fourier descriptor, we have a complete translation, rotation, and scale invariant description of the shape of objects. By leaving out higher-order Fourier descriptors, we can gradually relax fine details from the shape description in a controlled way.

Shape differences can then be measured by using the fact that the Fourier descriptors form a complex-valued vector. A metric for shape differences is then given by the magnitude of the difference vector:

$$d_{zz'} = \sum_{v=-N/2}^{N/2-1} |\hat{z}_v - \hat{z}'_v|^2 . \tag{19.16}$$

Depending on which normalization we apply to the Fourier descriptors, this metric will be scale and/or rotation invariant.

19.5 Shape Parameters

After discussing different ways to represent binary objects extracted from image data, we now turn to the question of how to describe the shape of these objects. This section discusses elementary geometrical parameters such as area and perimeter.

19.5.1 Area

One of the most trivial shape parameters is the *area* A of an object. In a digital binary image the number of pixels belonging to the image gives the area. So in the matrix or pixel list representation of the object, computing its area simply means counting the number of pixels. The area is also given as the zero-order moment of a binary object (Eq. (19.3)).

At first glance, area computation of an object, which is described by its chain-code, seems to be a complex operation. However, the contrary is true. Computation of the area from the chain code is much faster than counting pixels as the boundary of the object contains only a small fraction of the object's pixels and requires only two additions per boundary pixel.

The algorithm works in a similar way as numerical integration. We assume a horizontal base line drawn at an arbitrary vertical position in the image. Then we start the integration of the area at the uppermost pixel of the object. The distance of this point to the base line is B. We follow the boundary of the object and increment the area of the object according to the figures in Table 19.1.

If we, for example, move to the right (8-chain code 0), the area increases by B. If we move upwards to the right (chain code 1), the area also increases by B, but B must also be incremented, because the distance between the boundary pixel and the base line has increased. For

Table 19.1: *Computation of the area of an object from the chain code. Initially, the area is set to 1. With each step, the area and the parameter B are incremented corresponding to the value of the chain code; after Zamperoni [221].*

4-chain code	8-chain code	Flächenzunahme	Zunahme von B
0	0	$+B$	0
	1	$+B$	1
1	2	0	1
	3	$-B$	1
2	4	$-B$	0
	5	$-B$	-1
3	6	0	-1
	7	$+B$	-1

all movements to the left, the area is decreased by B. In this way, we subtract the area between the lower boundary line of the object and the base line, which was included in the area computation when moving to the right. The chain code must though to be located in the middle of the pixel. Therefore it does not give an area that is equal to the number of the pixels in the object. A one-pixel thin line has no area, a 2×2 square an area of one. The area A is initially set to zero.

The area can also be computed from Fourier descriptors. It is given by

$$A = \pi \sum_{v=-N/2}^{N/2-1} v\,|\hat{z}_v|^2. \tag{19.17}$$

This is a fast algorithm, which requires at most as many operations as points on the boundary line of the curve. The Fourier descriptors have the additional advantage that we can compute the area for a certain degree of smoothness by taking only a certain number of Fourier descriptors. The more Fourier descriptors we take, the more detailed is the boundary curve, as demonstrated in Fig. 19.7.

19.5.2 Perimeter

The *perimeter* is another geometrical parameter that can easily be obtained from the chain code of the object boundary. We just need to count the length of the chain code and take into consideration that steps in diagonal directions are longer by a factor of $\sqrt{2}$. The perimeter p is then given by an 8-neighborhood chain code:

$$p = n_e + \sqrt{2} n_o, \tag{19.18}$$

where n_e and n_o are the number of even and odd chain code steps, respectively. The steps with an uneven code are in a diagonal direction.

In contrast to the area, the perimeter is a parameter that is sensitive to the noise level in the image. The noisier the image, the more rugged and thus longer the boundary of an object will become in the segmentation procedure. This means that care must be taken in comparing perimeters that have been extracted from different images. We must be sure that the smoothness of the boundaries in the images is comparable.

Unfortunately, no simple formula exists to compute the perimeter from the Fourier descriptors, because the computation of the perimeter of ellipses involves elliptic integrals. However, the perimeter results directly from the construction of the boundary line with equidistant samples and is well approximated by the number of sampling points times the mean distance between the points.

19.5.3 Circularity

Area and perimeter are two parameters that describe the size of an object in one or the other way. In order to compare objects observed from different distances, it is important to use shape parameters that do not depend on the size of the object on the image plane. The *circularity* c is one of the simplest parameters of this kind. It is defined as

$$c = \frac{p^2}{A}. \tag{19.19}$$

The circularity is a dimensionless number with a minimum value of $4\pi \approx 12.57$ for circles. The circularity is 16 for a square and $12\sqrt{3} \approx 20.8$ for an equilateral triangle. Generally, it tends towards large values for elongated objects.

Area, perimeter, and circularity are shape parameters that do not depend on the orientation of the objects on the image plane. Thus they are useful to distinguish objects independently of their orientation.

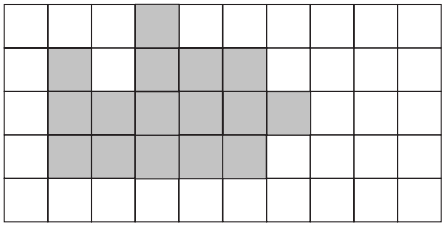
19.5.4 Bounding Box

Another simple and useful parameter for a crude description of the size of an object is the bounding box. It is defined as the rectangle that is just large enough to contain all object pixels. It gives also a rough description of the shape of the object. In contrast to the area (Section 19.5.1), however, it is not rotation invariant. It can be made rotation invariant if the object is first rotated into a standard orientation, for instance using the orientation of the moment tensor (Section 19.3.3). In any case, the bounding box is a useful feature if any further object-oriented pixel processing is required, such as extraction of the object pixels for further reference purposes.

19.6 *Exercises*

19.1: ****Representation of binary objects**

Compute from the binary object on a square grid shown below the run-length code, 4-neighborhood chain code, and 8-neighborhood chain code. Determine how many bytes you need to store it in different codes.



19.2: ****Circumference**

Compute directly from the codes in Exercise 19.1 the circumference of the object. How many computational steps are required?

19.3: ****Area**

Compute directly from the codes in Exercise 19.1 the area of the object. How many computational steps are required?

19.4: Elementary shape parameters

Interactive demonstration of elementary shape parameters, such as area and eccentricity (dip6ex19.01)

19.5: Moment-based shape parameters

Interactive demonstration of moment-based shape analysis (dip6ex19.02)

19.6: Fourier descriptors

Interactive demonstration of the properties of Fourier descriptors (dip6ex19.03)

19.7: ***Cartesian and polar Fourier descriptors**

Two types of Fourier descriptors are available: Cartesian descriptors and polar descriptors.

1. In which respects are these two descriptors different?
2. Are both descriptors suitable for all types of object contours?

19.8: ****Properties of Fourier descriptors**

Cartesian Fourier descriptors are an important tool to describe contours because many geometrical features can easily be extracted from them. We assume an object that is simply connected and thus has a single closed boundary. Answer the follow questions:

1. How can you detect a line-like object? (Hint: a closed curve means that the it runs from the starting point of the line to the end point and back again.)
2. How can you check for a symmetric object and determine its symmetry axis?
3. Can you determine the slope of a contour from the Fourier descriptors?
4. Can you smooth a contour using the Fourier descriptors?

19.9: ** **Detection of equal-sided triangles**

How can you detect equal-sided triangles with Fourier descriptors? Distinguish the following cases:

1. Equal-sized triangle with equal orientation
2. Triangles of different size but equal orientation (scale-invariant detection)
3. Triangles of different size and different orientation (scale-invariant and rotation-invariant detection).

19.10: *** **Moments and Fourier descriptors**

Researchers still argue whether Fourier descriptors or moments are the better method to describe the shape of objects. What is your opinion? Investigate especially the question of various invariant shape descriptors and the question how many parameters you need to describe a complex shape.

19.7 *Further Readings*

Spatial data structures, especially various tree structures, and their applications are detailed in the monographs by Samet [174, 175]. A detailed discussion of moment-based shape analysis with emphasize on invariant shape features can be found in the monography of Reiss [163]. Invariants based on gray values are discussed by Burkhardt and Siggelkow [17].