characteristics measured in this co-ordinate system yield an invariant description. However, the pose must be hypothesized for each object and each image, which makes this approach difficult and unreliable.

Application of **invariant theory**, where invariant descriptors can be computed directly from image data without the need for a particular co-ordinate system, represents another approach. In addition, invariant theory can determine the total number of functionally independent invariants for a given situation, therefore showing completeness of the description invariant set. Invariant theory is based on a collection of transforms that can be composed and inverted. In vision, the **plane-projective group** of transforms is considered which contains all the perspectives as a subset. The **group approach** provides a mathematical tool for generating invariants; if the transform does not satisfy the properties of a group, this machinery is not available [Mundy and Zisserman, 1992]. Therefore, the change of co-ordinates due to the plane-projective transform is generalized as a **group action**. **Lie group** theory is especially useful in designing new invariants.

Let corresponding entities in two different co-ordinate systems be distinguished by capital and lowercase letters. An invariant of a linear transformation is defined as follows:

> An invariant, $I(\mathbf{P})$, of a geometric structure described by a parameter vector $\mathbf{P}$, subject to a linear transformation $\mathbf{T}$ of the co-ordinates $\mathrm{x} = \mathbf{TX}$, is transformed according to $I(\mathbf{p}) = I(\mathbf{P})\,|\mathbf{T}|^w$. Here $(\mathbf{p})$ is the function of the parameters after the linear transformation, and $|\mathbf{T}|$ is the determinant of the matrix $\mathbf{T}$.

In this definition, w is referred to as the weight of the invariant. If $w = 0$, the invariants are called **scalar invariants**, which are considered below. Invariant descriptors are unaffected by object pose, by perspective projection, and by the intrinsic parameters of the camera.

Several examples of invariants are now given:

1. **Cross ratio**: The cross ratio represents a classic invariant of a projective line. As mentioned earlier, a straight line is always projected as a straight line. Any four collinear points A, B, C, D may be described by the cross-ratio invariant

$$I = \frac{(A-C)(B-D)}{(A-D)(B-C)}, \tag{8.24}$$

where $(A - C)$ represents the distance between points A and C (see Figure 8.19). Note that the cross ratio depends on the order in which the four collinear points are labeled.

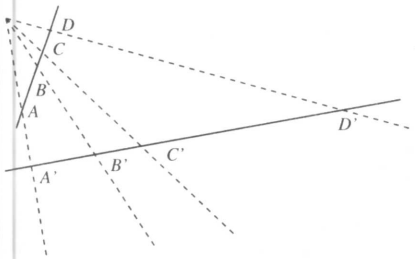


Figure 8.19: Cross ratio; four collinear points form a projective invariant.

2. **Systems of lines or points**: A system of four co-planar concurrent lines (meeting at the same point) is dual to a system of four collinear points and the cross ratio is its invariant; see Figure 8.19.

   A system of five general co-planar lines forms two invariants

$$I_1 = \frac{|\mathbf{M}_{431}|\,|\mathbf{M}_{521}|}{|\mathbf{M}_{421}|\,|\mathbf{M}_{531}|}, \qquad I_2 = \frac{|\mathbf{M}_{421}|\,|\mathbf{M}_{532}|}{|\mathbf{M}_{432}|\,|\mathbf{M}_{521}|}, \tag{8.25}$$

where $\mathbf{M}_{ijk} = (\mathbf{l}_i, \mathbf{l}_j, \mathbf{l}_k)$. $\mathbf{l}_i = (l_i^1, l_i^2, l_i^3)^T$ is a representation of a line $l_i^1 x + l_i^2 y + l_i^3 = 0$, where $i \in [1, 5]$, and $|\mathbf{M}|$ is the determinant of $\mathbf{M}$.

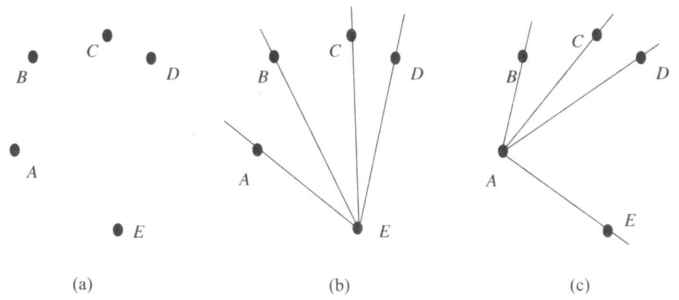


Figure 8.20: Five co-planar points form two cross-ratio invariants. (a) Co-planar points. (b) Five points form a system of four concurrent lines. (c) The same five points form another system of four co-planar lines.

If the three lines forming the matrix $\mathbf{M}_{ijk}$ are concurrent, the matrix becomes singular and the invariant is undefined.

A system of five co-planar points is dual to a system of five lines and the same two invariants are formed. These two functional invariants can also be formed as two cross ratios of two co-planar concurrent line quadruples; see Figure 8.20. Note that even though combinations other than those given in Figure 8.20 may be formed, only the two presented functionally independent invariants exist.

3. **Plane conics**: A plane conic may be represented by an equation

$$a x^2 + b x y + c y^2 + d x + e y + f = 0 \tag{8.26}$$

for $\mathbf{x} = (x, y, 1)^T$. Then the conic may also be defined by a matrix $\mathbf{C}$

$$\mathbf{C} = \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix}$$

and

$$\mathbf{x}^T \mathbf{C}\, \mathbf{x} = 0\,. \tag{8.27}$$

For any conic represented by a matrix $\mathbf{C}$, and any two co-planar lines not tangent to the conic, one invariant may be defined

$$I_1 = \frac{\left(\mathbf{l}_1^T \mathbf{C}^{-1} \mathbf{l}_2\right)^2}{\left(\mathbf{l}_1^T \mathbf{C}^{-1} \mathbf{l}_1\right)\left(\mathbf{l}_2^T \mathbf{C}^{-1} \mathbf{l}_2\right)}.$$

The same invariant can be formed for a conic and two co-planar points.

Two invariants can be determined for a pair of conics represented by their respective matrices $\mathbf{C}_1$, $\mathbf{C}_2$ normalized so that $|\mathbf{C}_i| = 1$

$$I_1 = \text{trace}\left[\mathbf{C}_1^{-1}\, \mathbf{C}_2\right], \quad I_2 = \text{trace}\left[\mathbf{C}_2^{-1}\, \mathbf{C}_1\right]. \tag{8.29}$$

(The trace of a matrix is calculated as the sum of elements on the main diagonal.) For non-normalized conics, the invariants of associated quadratic forms are

$$I_1 = \text{trace}\left[\mathbf{C}_1^{-1}\, \mathbf{C}_2\right]\left(\frac{|\mathbf{C}_1|}{|\mathbf{C}_2|}\right)^{\frac{1}{3}}, \quad I_2 = \text{trace}\left[\mathbf{C}_2^{-1}\, \mathbf{C}_1\right]\left(\frac{|\mathbf{C}_2|}{|\mathbf{C}_1|}\right)^{\frac{1}{3}} \tag{8.30}$$

and two true invariants of the conics are [Quan et al., 1992]

$$I_1 = \frac{\text{trace}\left[\mathbf{C}_1^{-1}\,\mathbf{C}_2\right]}{\text{trace}^2\left[\mathbf{C}_2^{-1}\,\mathbf{C}_1\right]}\,\frac{|\mathbf{C}_2|}{|\mathbf{C}_1|}, \quad I_2 = \frac{\text{trace}\left[\mathbf{C}_2^{-1}\,\mathbf{C}_1\right]}{\text{trace}^2\left[\mathbf{C}_1^{-1}\,\mathbf{C}_2\right]}\,\frac{|\mathbf{C}_2|}{|\mathbf{C}_1|}. \tag{8.31}$$

An interpretation of these invariants is given in [Maybank, 1992]. Two plane conics uniquely determine four points of intersection, and any point that is not an intersection point may be chosen to form a five-point system together with the four intersection points. Therefore, two invariants exist for the pair of conics, as for the five-point system.

Many man-made objects consist of a combination of straight lines and conics, and these invariants may be used for their description. However, if the object has a contour which cannot be represented by an algebraic curve, the situation is much more difficult. **Differential invariants** can be formed (e.g., curvature, torsion, Gaussian curvature) which are not affected by projective transforms. These invariants are local—that is, the invariants are found for each point on the curve, which may be quite general. Unfortunately, these invariants are extremely large and complex polynomials, requiring up to seventh derivatives of the curve, which makes them practically unusable due to image noise and acquisition errors. However, if additional information is available, higher derivatives may be avoided. In [Mundy and Zisserman, 1992], higher derivatives are traded for extra reference points which can be detected on curves in different projections, although the necessity of matching reference points in different projections brings other difficulties.

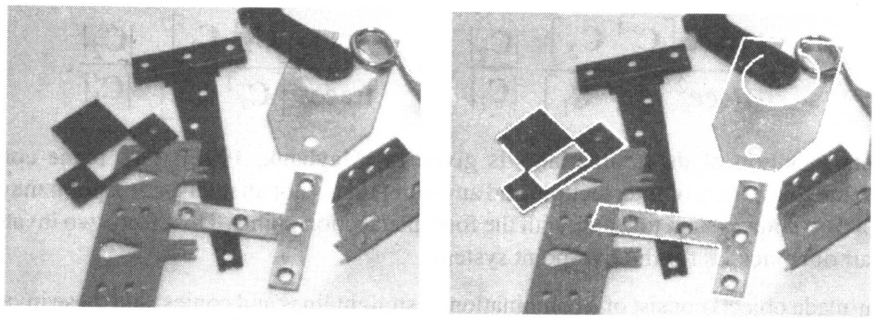
Designing new invariants is an important part of invariant theory in its application to machine vision. The easiest way is to combine primitive invariants, forming new ones from these combinations. Nevertheless, no new information is obtained from these combinations. Further, complete tables of invariants for systems of vectors under the action of the rotation group, the affine transform group, and the general linear transform group may be found in [Weyl, 1946]. To obtain new sets of functional invariants, several methods (eliminating transform parameters, the infinitesimal method, the symbolic method) can be found in [Forsyth et al., 1991; Mundy and Zisserman, 1992].

Stability of invariants is another crucial property which affects their applicability. The robustness of invariants to image noise and errors introduced by image sensors is of prime importance, although not much is known about this. Results of plane-projective invariant stability testing (cross ratio, five co-planar points, two co-planar conics) can be found in [Forsyth et al., 1991]. Further, different invariants have different stabilities and distinguishing powers. It was found, for example [Rothwell et al., 1992], that measuring a single conic and two lines in a scene is too computationally expensive to be worthwhile. It is recommended to combine different invariants to enable fast object recognition.

An example of recognition of man-made objects using invariant description of four co-planar lines, a conic and two lines, and a pair of co-planar conics is given in [Rothwell et al., 1992]. The recognition system is based on a model library containing over 30 object models—significantly more than are reported for other recognition systems. Moreover, the construction of the model library is extremely easy; no special measurements are needed, the object is digitized in a standard way, and the projectively invariant description is stored as a model. Further, there is no need for camera calibration. The recognition accuracy is 100% for occluded objects viewed from different viewpoints if the objects are not severely disrupted by shadows and specularities. An example of such object recognition is given in Figure 8.21.

8.3 REGION-BASED SHAPE REPRESENTATION AND DESCRIPTION

We can use boundary information to describe a region, and shape can be described from the region itself. A large group of shape description techniques is represented by heuristic approaches which yield acceptable results in description of simple shapes. Region area, rectangularity, elongatedness, direction, compactness, etc., are examples of these methods. Unfortunately, they cannot be used for region reconstruction and do not work for more complex shapes. Other procedures based on region decomposition into smaller and simpler



(a) (b)

Figure 8.21: Object recognition based on shape invariants. (a) Original image of overlapping objects taken from an arbitrary viewpoint. (b) Object recognition based on line and conic invariants. *Courtesy of D. Forsyth, The University of Iowa; C. Rothwell, A. Zisserman, University of Oxford; J. Mundy, General Electric Corporate Research and Development, Schenectady, NY.*

sub-regions must be applied to describe more complicated regions, then sub-regions can be described separately using heuristic approaches. Objects are represented by a planar graph with nodes representing sub-regions resulting from region decomposition, and region shape is then described by the graph properties [Rosenfeld, 1979; Bhanu and Faugeras, 1984; Turney et al., 1985]. There are two general approaches to acquiring a graph of sub-regions: The first one is region thinning leading to the **region skeleton**, which can be described by a graph. The second option starts with the **region decomposition** into sub-regions, which are then represented by nodes, while arcs represent neighborhood relations of sub-regions. It is common to stipulate that sub-regions be convex.

Graphical representation of regions has many advantages; the resulting graphs:

- Are translation and rotation invariant; position and rotation can be included in the graph definition.
- Are insensitive to small changes in shape.
- Are highly invariant with respect to region magnitude.
- Generate a representation which is understandable.
- Can easily be used to obtain the information-bearing features of the graph.
- Are suitable for syntactic recognition.

On the other hand, the shape representation can be difficult to obtain and the classifier-learning stage is not easy either (see Chapter 9). Nevertheless, if we are to get closer to the reality of computer vision, and to understand complex images, there is no alternative.

8.3.1 Simple scalar region descriptors

A number of simple heuristic shape descriptors exist which relate to statistical feature description. These methods are basic and are used for description of sub-regions in complex regions, and may then be used to define graph node classification [Bribiesca and Guzman, 1980].

Area

The simplest and most natural property of a region is its area, given by the number of pixels of which the region consists. The *real* area of each pixel may be taken into consideration to get the *real size* of a region, noting that in many cases, especially in satellite imagery, pixels in different positions correspond to different areas in the real world. If an image is represented as a rectangular raster, simple counting of region pixels will provide its area. If the image is represented by a quadtree, however, it may be more difficult to find the region area. Assuming that regions have been identified by labeling, the following algorithm may be used.

Algorithm 8.4: Calculating area in quadtrees

1. Set all region area variables to zero, and determine the global quadtree depth H; for example, the global quadtree depth is $H = 8$ for a 256×256 image.
2. Search the tree in a systematic way. If a leaf node at a depth h has a non-zero label, proceed to step 3.
3. Compute:
$$area[region\_label] = area[region\_label] + 4^{(H-h)}.$$
4. The region areas are stored in variables $area[region\_label]$.

The region can be represented by n polygon vertices (i_k, j_k), and $(i_0, j_0) = (i_n, j_n)$. The area is given by

$$area = \frac{1}{2}\left|\sum_{k=0}^{n-1}\left(i_k j_{k+1} - i_{k+1} j_k\right)\right| \tag{8.32}$$

—the sign of the sum represents the polygon orientation. If a smoothed boundary is used to overcome noise sensitivity problems, the region area value resulting from equation (8.32) is usually somewhat reduced. Various smoothing methods and accurate area-recovering techniques are given in [Koenderink and v Doorn, 1986].

If the region is represented by the (anti-clockwise) Freeman chain code, the following algorithm provides the area.

Algorithm 8.5: Region area calculation from Freeman 4-connectivity chain code representation

1. Set the region *area* to zero. Assign the value of the starting point i co-ordinate to the variable *vertical_position*.
2. For each element of the chain code (values 0, 1, 2, 3) do

```
switch(code) {
    case 0:
        area := area - vertical_position;
        break;
    case 1:
        vertical_position := vertical_position + 1;
        break;
    case 2:
        area := area + vertical_position;
        break;
    case 3:
        vertical_position := vertical_position - 1;
        break;
}
```

3. If all boundary chain elements have been processed, the region area is stored in the variable *area*.

Euler's number

Euler's number ϑ (sometimes called **genus** or the **Euler-Poincaré characteristic**) describes a simple, topologically invariant property of the object. It is based on S, the number of contiguous parts of an object, and N, the number of holes in the object (an object can consist of more than one region, otherwise the number of contiguous parts is equal to one; see Section 2.3.1). Then

$$\vartheta = S - N. \tag{8.33}$$

Special procedures to compute Euler's number can be found in [Dyer, 1980; Rosenfeld and Kak, 1982; Pratt, 1991], and in Chapter 13.

Projections

Horizontal and vertical region projections $p_h(i)$ and $p_v(j)$ are defined as

$$p_h(i) = \sum_j f(i,j), \qquad p_v(j) = \sum_i f(i,j). \tag{8.34}$$

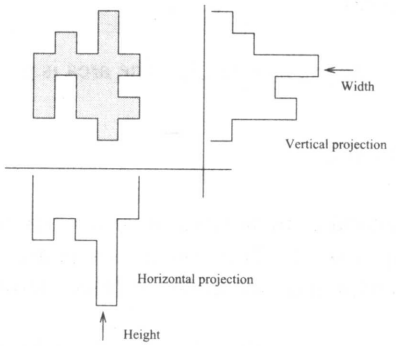


Figure 8.22: Projections.

Region description by projections is usually connected to binary image processing. Projections can serve as a basis for definition of related region descriptors; for example, the width (height) of a region with no holes is defined as the maximum value of the horizontal (vertical) projection of a binary image of the region. These definitions are illustrated in Figure 8.22. Note that projections can be defined in any direction.

Eccentricity

The simplest eccentricity characteristic is the ratio of the length of the maximum chord A to the maximum chord B which is perpendicular to A (the ratio of major and minor axes of an object)—see Section 8.2.2, Figure 8.23. Another approximate eccentricity measure is based on a ratio of main region axes of inertia [Ballard and Brown, 1982; Jain, 1989].

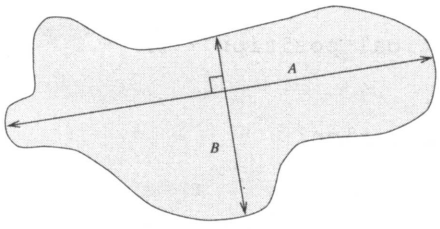
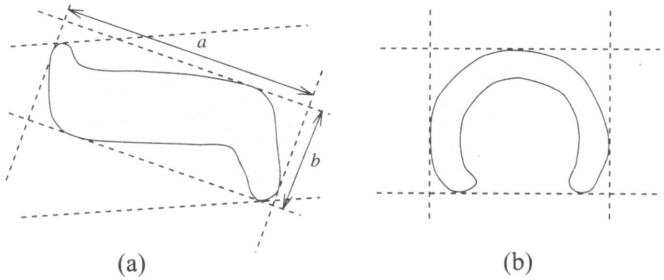


Figure 8.23: Eccentricity.

Elongatedness

Elongatedness is a ratio between the length and width of the region bounding rectangle. This is the rectangle of minimum area that bounds the shape, which is located by turning in discrete steps until a minimum is located (see Figure 8.24a). This criterion cannot succeed in curved regions (see Figure 8.24b), for which the evaluation of elongatedness must be based on maximum region thickness. Elongatedness can be evaluated as a ratio of the region area and the square of its thickness. The maximum region thickness (holes must be filled if present) can be determined as the number of erosion steps (see Chapter 13) that may be applied before the region totally disappears. If the number of erosion steps is d, elongatedness is then

$$elongatedness = \frac{area}{(2d)^2}. \tag{8.35}$$



(a) (b)

Figure 8.24: Elongatedness: (a) bounding rectangle gives acceptable results; (b) bounding rectangle cannot represent elongatedness.

Another method based on longest central line detection is described in [Nagao and Matsuyama, 1980]; representation and recognition of elongated regions is also discussed in [Lipari and Harlow, 1988].

Note that the bounding rectangle can be computed efficiently from boundary points, if its direction $\grave{e}$ is known. Defining

$$\alpha(x, y) = x \cos\theta + y \sin\theta, \quad \alpha(x, y) = -x \sin\theta + y \cos\theta, \tag{8.36}$$

search for the minimum and maximum of α and β over all boundary points (x, y). The values of α_{min}, α_{max}, β_{min}, β_{max} then define the bounding rectangle, and $l_1 = (\alpha_{max} - \alpha_{min})$ and $l_2 = (\alpha_{max} - \alpha_{min})$ are its length and width.

Rectangularity

Let F_k be the ratio of region area and the area of a bounding rectangle, the rectangle having the direction k. The rectangle direction is turned in discrete steps as before, and **rectangularity** measured as a maximum of this ratio F_k:

$$rectangularity = \max_k F_k. \tag{8.37}$$

The direction need only be turned through one quadrant. Rectangularity assumes values from the interval (0, 1], with 1 representing a perfectly rectangular region. Sometimes, it may be more natural to draw a bounding triangle; a method for similarity evaluation between two triangles called **sphericity** is presented in [Ansari and Delp, 1990].

Direction

Direction is a property which makes sense in elongated regions only. If the region is elongated, **direction** is the direction of the longer side of a minimum bounding rectangle. If the shape moments are known (Section 8.3.2), the direction θ can be computed as

$$\theta = \frac{1}{2}\arctan\left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}}\right). \tag{8.38}$$

It should be noted that elongatedness and rectangularity are independent of linear transformations—translation, rotation, and scaling. Direction is independent on all linear transformations which do not include rotation. Mutual direction of two rotating objects is rotation invariant.

Compactness

Compactness is a popular shape description characteristic independent of linear transformations given by

$$compactness = \frac{(region\_border\_length)^2}{area}. \tag{8.39}$$

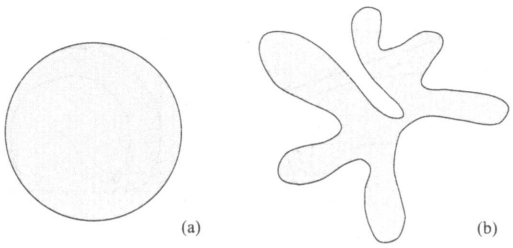


Figure 8.25: Compactness: (a) compact; (b) non-compact.

The most compact region in a Euclidean space is a circle. Compactness assumes values in the interval $[1, \infty)$ in digital images if the boundary is defined as an inner boundary (see Section 6.2.3); using the outer boundary, compactness assumes values in the interval [16, 8). Independence from linear transformations is gained only if an outer boundary representation is used. Examples of a compact and a non-compact region are shown in Figure 8.25.

8.3.2 Moments

Region moment representations interpret a normalized gray-level image function as a probability density of a 2D random variable. Properties of this random variable can be described using statistical characteristics—**moments** [Papoulis, 1991]. Assuming that non-zero pixel values represent regions, moments can be used for binary or gray-level region description. A moment of order $(p + q)$ is dependent on scaling, translation, rotation, and even on gray-level transformations and is given by

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) \mathrm{d}x \mathrm{d}y. \tag{8.40}$$

In digitized images we evaluate sums

$$m_{pq} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} i^p j^q f(i, j), \tag{8.41}$$

where x, y, i, j are the region point co-ordinates (pixel co-ordinates in digitized images). Translation invariance can be achieved if we use the central moments

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - x_c)^p (y - y_c)^q f(x, y) \mathrm{d}x \mathrm{d}y, \tag{8.42}$$

or in digitized images

$$\mu_{pq} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} (i - x_c)^p (j - y_c)^q f(i, j), \tag{8.43}$$

where x_c, y_c are the co-ordinates of the region's center of gravity (centroid), which can be obtained using the following relationships:

$$x_c = \frac{m_{10}}{m_{00}}, \qquad y_c = \frac{m_{01}}{m_{00}}. \tag{8.44}$$

In the binary case, m_{00} represents the region area (see equations (8.40) and (8.41)). Scale-invariant features can also be found in scaled central moments η_{pq} (scale change $x' = \alpha x$, $y' = \alpha y$)

$$\eta_{pq} = \frac{\mu'_{pq}}{\left(\mu'_{00}\right)^{\gamma}}, \qquad \gamma = \frac{p+q}{2} + 1, \qquad \mu'_{pq} = \frac{\mu_{pq}}{\alpha^{p+q+2}}, \tag{8.45}$$

and normalized un-scaled central moments ϑ_{pq}

$$\vartheta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}}. \tag{8.46}$$

Rotation invariance can be achieved if the co-ordinate system is chosen such that $\mu_{11} = 0$ [Cash and Hatamian, 1987]. Many aspects of moment properties, normalization, descriptive power, sensitivity to noise, and computational cost are discussed in [Savini, 1988]. A less general form of invariance was given in [Hu, 1962] and is discussed in [Maitra, 1979; Jain, 1989; Pratt, 1991], in which seven rotation-, translation-, and scale-invariant moment characteristics are used.

$$\varphi_1 = \vartheta_{20} + \vartheta_{02}, \tag{8.47}$$

$$\varphi_2 = (\vartheta_{20} - \vartheta_{02})^2 + 4\vartheta_{11}^2, \tag{8.48}$$

$$\varphi_3 = (\vartheta_{30} - 3\vartheta_{12})^2 + (3\vartheta_{21} - \vartheta_{03})^2, \tag{8.49}$$

$$\varphi_4 = (\vartheta_{30} + \vartheta_{12})^2 + (\vartheta_{21} + \vartheta_{03})^2, \tag{8.50}$$

$$\begin{aligned}\varphi_5 = {} & (\vartheta_{30} - 3\vartheta_{12})(\vartheta_{30} + \vartheta_{12})((\vartheta_{30} + \vartheta_{12})^2 - 3(\vartheta_{21} + \vartheta_{03})^2) \\ & + (3\vartheta_{21} - \vartheta_{03})(\vartheta_{21} + \vartheta_{03})(3(\vartheta_{30} + \vartheta_{12})^2 - (\vartheta_{21} + \vartheta_{03})^2),\end{aligned} \tag{8.51}$$

$$\varphi_6 = (\vartheta_{20} - \vartheta_{02})((\vartheta_{30} + \vartheta_{12})^2 - (\vartheta_{21} + \vartheta_{03})^2) + 4\vartheta_{11}(\vartheta_{30} + \vartheta_{12})(\vartheta_{21} + \vartheta_{03}), \tag{8.52}$$

$$\begin{aligned}\varphi_7 = {} & (3\vartheta_{21} - \vartheta_{03})(\vartheta_{30} + \vartheta_{12})((\vartheta_{30} + \vartheta_{12})^2 - 3(\vartheta_{21} + \vartheta_{03})^2) \\ & - (\vartheta_{30} - 3\vartheta_{12})(\vartheta_{21} + \vartheta_{03})(3(\vartheta_{30} + \vartheta_{12})^2 - (\vartheta_{21} + \vartheta_{03})^2),\end{aligned} \tag{8.53}$$

where the ϑ_{pq} values can be computed from equation (8.46).

While the seven moment characteristics presented above were shown to be useful, they are invariant only to translation, rotation, and scaling. Recent algorithms for fast computation of translation-, rotation-, and scale-invariant moments were given in [Li and Shen, 1991; Jiang and Bunke, 1991]. However, these approaches do not yield descriptors that are invariant under general affine transforms. A complete set of four affine moment invariants derived from second- and third-order moments is presented in [Flusser and Suk, 1993]

$$I_1 = \frac{\mu_{20}\mu_{02} - \mu_{11}^2}{\mu_{00}^4}, \tag{8.54}$$

$$I_2 = \frac{\mu_{30}^2\mu_{03}^2 - 6\mu_{30}\mu_{21}\mu_{12}\mu_{03} + 4\mu_{30}\mu_{12}^3 + 4\mu_{21}^3\mu_{03} - 3\mu_{21}^2\mu_{12}^2}{\mu_{00}^{10}}, \tag{8.55}$$

$$I_3 = \frac{\mu_{20}(\mu_{21}\mu_{03} - \mu_{12}^2) - \mu_{11}(\mu_{30}\mu_{03} - \mu_{21}\mu_{12}) + \mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2)}{\mu_{00}^7}, \tag{8.56}$$

$$\begin{aligned}I_4 = {} & (\mu_{20}^3\mu_{03}^2 - 6\mu_{20}^2\mu_{11}\mu_{12}\mu_{03} - 6\mu_{20}^2\mu_{02}\mu_{21}\mu_{03} + 9\mu_{20}^2\mu_{02}\mu_{12}^2 \\ & + 12\mu_{20}\mu_{11}^2\mu_{21}\mu_{03} + 6\mu_{20}\mu_{11}\mu_{02}\mu_{30}\mu_{03} - 18\mu_{20}\mu_{11}\mu_{02}\mu_{21}\mu_{12} \\ & - 18\mu_{11}^3\mu_{30}\mu_{03} - 6\mu_{20}\mu_{02}^2\mu_{30}\mu_{12} + 9\mu_{20}\mu_{02}^2\mu_{21}^2 \\ & + 12\mu_{11}^2\mu_{02}\mu_{30}\mu_{12} - 6\mu_{11}\mu_{02}^2\mu_{30}\mu_{21} + \mu_{02}^3\mu_{30}^2)/\mu_{00}^{11}.\end{aligned} \tag{8.57}$$

Details of the process for the derivation of invariants and examples of invariant moment object descriptions can be found in [Flusser and Suk, 1993].

All moment characteristics are dependent on the linear gray-level transformations of regions; to describe region shape properties, we work with binary image data ($f(i,j) = 1$ in region pixels) and dependence on the linear gray-level transform disappears.

Moment characteristics can be used in shape description even if the region is represented by its boundary. A closed boundary is characterized by an ordered sequence $z(i)$ that represents the Euclidean distance between the centroid and all N boundary pixels of the digitized shape. No extra processing is required for shapes having spiral or concave contours. Translation-, rotation-, and scale-invariant one-dimensional normalized contour sequence moments $\bar{m}_r$, $\bar{\mu}_r$ are defined in [Gupta and Srinath, 1987]. The r^{th} contour sequence moment m_r and the r^{th} central moment μ_r can be estimated as

$$m_r = \frac{1}{N}\sum_{i=1}^{N}\left(z(i)\right)^r, \tag{8.58}$$

$$\mu_r = \frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^r. \tag{8.59}$$

The r^{th} normalized contour sequence moment $\bar{m}_r$ and normalized central contour sequence moment $\bar{\mu}_r$ are defined as

$$\bar{m}_r = \frac{m_r}{\mu_2^{r/2}} = \frac{\frac{1}{N}\sum_{i=1}^{N}\left(z(i)\right)^r}{\left(\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^2\right)^{r/2}}, \tag{8.60}$$

$$\bar{\mu}_r = \frac{\mu_r}{(\mu_2)^{r/2}} = \frac{\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^r}{\left(\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^2\right)^{r/2}}. \tag{8.61}$$

While the set of invariant moments $\bar{m}_r$, $\bar{\mu}_r$ can be used directly for shape representation, less noise-sensitive results can be obtained from the following shape descriptors [Gupta and Srinath, 1987]

$$F_1 = \frac{(\mu_2)^{1/2}}{m_1} = \frac{\left(\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^2\right)^{1/2}}{\frac{1}{N}\sum_{i=1}^{N}z(i)}, \tag{8.62}$$

$$F_2 = \frac{\mu_3}{(\mu_2)^{3/2}} = \frac{\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^3}{\left(\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^2\right)^{3/2}}, \tag{8.63}$$

$$F_3 = \frac{\mu_4}{(\mu_2)^2} = \frac{\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^4}{\left(\frac{1}{N}\sum_{i=1}^{N}\left(z(i)-m_1\right)^2\right)^2}, \tag{8.64}$$

$$F_4 = \bar{\mu}_5. \tag{8.65}$$

Lower probabilities of error classification were obtained using contour sequence moments than area-based moments (8.47)-(8.53) in a shape recognition test; also, contour sequence moments are less computationally demanding.

8.3.3 Convex hull

A region R is convex if and only if for any two points $\mathbf{x}_1, \mathbf{x}_2 \in R$, the whole line segment $\mathbf{x}_1\mathbf{x}_2$ defined by its end points $\mathbf{x}_1, \mathbf{x}_2$ is inside the region R. The convex hull of a region is the smallest convex region H which

satisfies the condition $R \subseteq H$—see Figure 8.26. The convex hull has some special properties in digital data which do not exist in the continuous case. For instance, concave parts can appear and disappear in digital data due to rotation, and therefore the convex hull is not rotation invariant in digital space [Gross and Latecki, 1995]. The convex hull can be used to describe region shape properties and can be used to build a tree structure of region concavity.

A discrete convex hull can be defined by the following algorithm which may also be used for convex hull construction. This algorithm has complexity $\mathcal{O}(n^2)$ and is presented here as an intuitive way of detecting the convex hull. Algorithm 8.7 describes a more efficient approach.

Algorithm 8.6: Region convex hull construction

1. Find all pixels of a region R with the minimum row co-ordinate; among them, find the pixel P_1 with the minimum column co-ordinate. Assign $\mathbf{P}_k = \mathbf{P}_1$, $\mathbf{v} = (0, -1)$; the vector v represents the direction of the previous line segment of the convex hull.
2. Search the region boundary in an anti-clockwise direction (Algorithm 6.7) and compute the angle orientation φ_n for every boundary point $\mathbf{P}_n$ which lies after the point $\mathbf{P}_1$ (in the direction of boundary search—see Figure 8.26). The angle orientation φ_n is the angle of vector $\mathbf{P}_k\mathbf{P}_n$. The point $\mathbf{P}_q$ satisfying the condition $\varphi_q = \min_n \varphi_n$ is an element (vertex) of the region convex hull.
3. Assign $\mathbf{v} = \mathbf{P}_k\mathbf{P}_q$, $\mathbf{P}_k = \mathbf{P}_q$.
4. Repeat steps 2 and 3 until $\mathbf{P}_k = \mathbf{P}_1$.

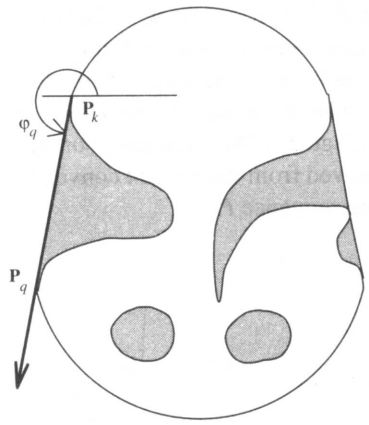


Figure 8.26: Convex hull.

The first point $\mathbf{P}_1$ need not be chosen as described in the given algorithm, but it must be an element of a convex segment of the inner region boundary.

As has been mentioned, more efficient algorithms exist, especially if the object is defined by an ordered sequence $P = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ of n vertices, $\mathbf{v}_i$ representing a polygonal boundary of the object. Many algorithms [Toussaint, 1985] exist for detection of the convex hull with computational complexity $\mathcal{O}(n \log n)$ in the worst case; these algorithms and their implementations vary in speed and memory requirements. As discussed in [Toussaint, 1991], the code of [Bhattacharya and Toussaint, 1983] (in which a Fortran listing appears) seems to be the fastest to date, using only $5n$ storage space.

If the polygon P is a *simple* polygon (self-non-intersecting polygon) which is always the case in a polygonal representation of object borders, the convex hull may be found in linear time $\mathcal{O}(n)$. In the past two decades, many linear-time convex hull detection algorithms have been published; however more than half of them were later discovered to be incorrect [Toussaint, 1985, 1991], with counter-examples published. The algorithm of [McCallum and Avis, 1979] was the first correct linear-time one. The simplest correct convex hull algorithm was given in [Melkman, 1987] and was based on previous work [Lee, 1983; Bhattacharya and Gindy, 1984; Graham and Yao, 1984]. Melkman's convex hull detection algorithm is now discussed further.

Let the polygon for which the convex hull is to be determined be a simple polygon $P = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ and let the vertices be processed in this order. For any three vertices **x**, **y**, **z** in an ordered sequence, a directional function δ may be evaluated (Figure 8.27)

$$\begin{aligned}\delta(\mathbf{x}, \mathbf{y}, \mathbf{z}) &= 1 \quad \text{if } \mathbf{z} \text{ is to the right of the directed line } \mathbf{xy},\\ &= 0 \quad \text{if } \mathbf{z} \text{ is collinear with the directed line } \mathbf{xy},\\ &= -1 \quad \text{if } \mathbf{z} \text{ is to the left of the directed line } \mathbf{xy}.\end{aligned}$$

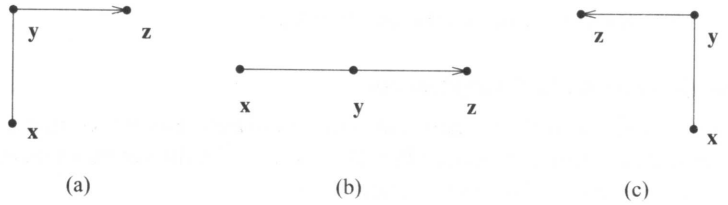


Figure 8.27: Directional function δ. (a) $\delta(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 1$. (b) $\delta(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0$. (c) $\delta(\mathbf{x}, \mathbf{y}, \mathbf{z}) = -1$.

The main data structure H is a list of vertices (deque) of polygonal vertices already processed. The current contents of H represents the convex hull of the currently processed part of the polygon, and after the detection is completed, the convex hull is stored in this data structure. Therefore, H always represents a closed polygonal curve, $H = \{d_b, ..., d_t\}$ where d_b points to the bottom of the list and d_t points to its top. Note that d_b and d_t always refer to the same vertex simultaneously representing the first and the last vertex of the closed polygon.

Here are the main ideas of the algorithm. The first three vertices A, B, C from the sequence P form a triangle (if not collinear) and this triangle represents a convex hull of the first three vertices—Figure 8.28a. The next vertex D in the sequence is then tested for being located inside or outside the current convex hull. If D is located inside, the current convex hull does not change—Figure 8.28b. If D is outside of the current convex hull, it must become a new convex hull vertex (Figure 8.28c) and, based on the current convex hull shape, either none, one, or several vertices must be removed from the current convex hull—Figure 8.28c,d. This process is repeated for all remaining vertices in the sequence P.

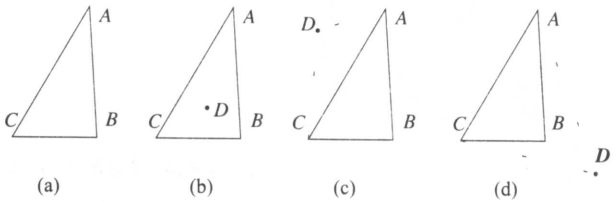


Figure 8.28: Convex hull detection. (a) First three vertices A, B, C form a triangle. (b) If the next vertex D is positioned inside the current convex hull ABC, current convex hull does not change. (c) If the next vertex D is outside of the current convex hull, it becomes a new vertex of the new current convex hull $ABCDA$. (d) In this case, vertex B must be removed from the current convex hull and the new current convex hull is $ADCA$.

Following the terminology used in [Melkman, 1987], the variable v refers to the input vertex under consideration, and the following operations are defined:

```
push v:      t := t + 1,     d_t → v,
pop d_t:     t := t − 1,
insert v:    b := b − 1,     d_b → v,
remove d_b:  b := b + 1,
input v:     next vertex is entered from sequence P, if P is empty, stop,
```

where → means 'points to'. The algorithm is then as follows.

Algorithm 8.7: Simple polygon convex hull detection

1. Initialize.
   - $t := -1$;
   - $b := 0$;
   - input $\mathbf{v}_1$; input $\mathbf{v}_2$; input $\mathbf{v}_3$;
   - if ($\delta(\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) > 0$)
   - {push $\mathbf{v}_1$;
   - push $\mathbf{v}_2$; }
   - else
   - {push $\mathbf{v}_2$;
   - push $\mathbf{v}_1$; }
   - push $\mathbf{v}_3$;
   - insert $\mathbf{v}_3$;
2. If the next vertex $\mathbf{v}$ is inside the current convex hull H, enter and check a new vertex; otherwise process steps 3 and 4;
   - input $\mathbf{v}$;
   - while ($\delta(\mathbf{v}, d_b, d_{b+1}) \geq 0$ AND $\delta(d_{t-1}, d_t, \mathbf{v}) \geq 0$)
   - input $\mathbf{v}$;
3. Rearrange vertices in H, top of the list.
   - while ($\delta(d_{t-1}, d_t, \mathbf{v}) \leq 0$)
   - pop d_t;
   - push $\mathbf{v}$;
4. Rearrange vertices in H, bottom of the list.
   - while ($\delta(\mathbf{v}, d_b, d_{b+1}) = 0$)
   - remove d_b;
   - insert $\mathbf{v}$;
   - go to step 2;

The algorithm as presented may be difficult to follow, but a less formal version would be impossible to implement; a formal proof is given in [Melkman, 1987]. The following example makes the algorithm more understandable.

Let $P = \{A, B, C, D, E\}$ as shown in Figure 8.29a. The data structure H is created in the first step:

$$\begin{array}{lcccccc} t,b\ldots & & -1 & 0 & 1 & 2 \\ H & = & C & A & B & C \\ & & d_b & & & d_t \end{array}$$

In the second step, vertex D is entered (Figure 8.29b):

$$\delta(D, d_b, d_{b+1}) = \delta(D, C, A) = 1 > 0\,,$$

$$\delta(d_{t-1}, d_t, D) = \delta(B, C, D) = -1 < 0\,.$$

Based on the values of the directional function δ, in this case, no other vertex is entered during this step. Step 3 results in the following current convex hull H

$$\delta(B,C,D) = -1 \rightarrow \text{pop } d_t \rightarrow \begin{array}{lcccccc} t,b\ldots & & -1 & 0 & 1 & 2 \\ H & = & C & A & B & C\,, \\ & & d_b & & d_t & \end{array}$$

$$\delta(A,B,D) = -1 \rightarrow \text{pop } d_t \rightarrow \begin{array}{lcccccc} & t,b\ldots & & -1 & 0 & 1 & 2 \\ H & = & & C & A & B & C \\ & & d_b & & d_t & & \end{array},$$

$$\delta(C,A,D) = 1 \rightarrow \text{push } D \rightarrow \begin{array}{lcccccc} & t,b\ldots & & -1 & 0 & 1 & 2 \\ H & = & & C & A & D & C \\ & & d_b & & & d_t & \end{array}.$$

In step 4—Figure 8.29c

$$\delta(D,C,A) = 1 \rightarrow \text{insert } D \rightarrow \begin{array}{lcccccc} & t,b\ldots & & -2 & -1 & 0 & 1 & 2 \\ H & = & & D & C & A & D & C \\ & & d_b & & & & d_t & \end{array}.$$

Go to step 2; vertex E is entered—Figure 8.29d

$$\delta(E, D, C) = 1 > 0,$$

$$\delta(A, D, E) = 1 > 0.$$

A new vertex should be entered from P, but there is no unprocessed vertex in the sequence P and the convex hull generating process stops. The resulting convex hull is defined by the sequence $H = \{d_b,\ldots,d_t\} = \{D, C, A, D\}$, which represents a polygon $DCAD$, always in the clockwise direction—Figure 8.29e.

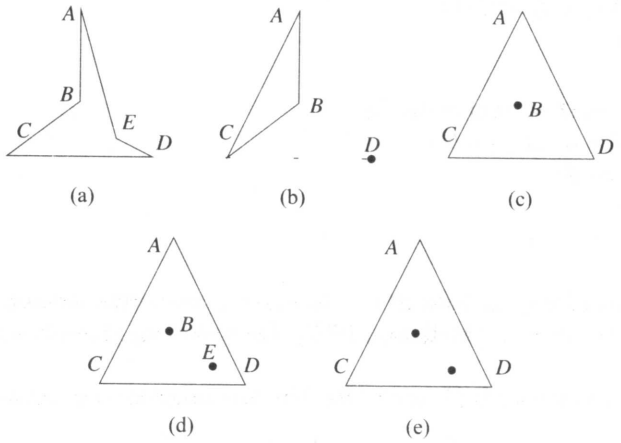


Figure 8.29: Example of convex hull detection. (a) The processed region—polygon *ABCDEA*. (b) Vertex D is entered and processed. (c) Vertex D becomes a new vertex of the current convex hull *ADC*. (d) Vertex E is entered and processed, E does not become a new vertex of the current convex hull. (e) The resulting convex hull *DCAD*.

A **region concavity tree** is another shape representation option [Sklansky, 1972]. A tree is generated recursively during the construction of a convex hull. A convex hull of the whole region is constructed first, and convex hulls of concave residua are found next. The resulting convex hulls of concave residua of the regions from previous steps are searched until no concave residuum exists. The resulting tree is a shape representation of the region. Concavity tree construction can be seen in Figure 8.30.

8.3.4 Graph representation based on region skeleton

This method corresponds significantly curving points of a region boundary to graph nodes. The main disadvantage of boundary-based description methods is that geometrically close points can be far away from