7
Geometry

The need for geometric manipulation of images appears in many image processing applications. The removal of optical distortions introduced by a camera, the geometric 'warping' of an image to conform to some standard reference shape and the accurate registration of two or more images are just a few examples requiring a mathematical treatment involving geometric concepts. Inextricably related to our discussion of geometry in images is the notion of shape. In this chapter we will introduce a preliminary definition of shape which provides the necessary basis for subsequent discussion of the geometric techniques that form the bulk of the chapter.

We note in passing that the definition of useful *shape descriptors* and methods for their automated calculation play an important role in *feature extraction*, a key step in pattern recognition techniques. Typically, shape descriptors use or reduce the coordinate pairs in the shape vector representation to produce some compact or even *single-parameter* measure of the approximate shape. A discussion of these techniques is deferred until Chapter 9.

7.1 The description of shape

It is clear that our simple, everyday concept of shape *implies the existence of some boundary*. If we talk of the shape of objects such as a cigar, a football or an egg, we are implicitly referring to the boundary of the object. These objects are simple and the specification of the coordinates constituting the boundary is considered a *sufficient* mathematical description of the shape. For more complex objects, the boundary coordinates are usually not mathematically sufficient. A good example of the latter is the human face. If we are to describe face-shape accurately enough to discriminate between faces then we need to specify not just the overall outline of the head, but also all the internal features, such as eyes, nose, mouth, etc. Thus, to remain general, our basic description of shape is taken to be some ordered set of coordinate pairs (or tuples for higher dimensional spaces) which we deem sufficient for the particular purpose we have in mind. One of the simplest and most direct ways to describe a shape mathematically is to locate a finite number N of points along the boundary and concatenate them to constitute a *shape vector* which, for a 2-D object, we simply denote as:

$$\mathbf{x} = [x_1 \quad y_1 \quad x_2 \quad \cdots \quad x_N \quad y_N]$$

Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab Chris Solomon and Toby Breckon
 ISBN: 978-0-470-84472-4

In general, the points defining the shape may be selected and annotated manually or they may be the result of an automated procedure.

There are two main aspects to the treatment of boundaries and shape in image processing:

- How do we identify and locate a defining feature boundary in an image?
- How, then, can we label boundaries and features to provide a meaningful and useful definition of the shape?

The first aspect is a very important one and belongs largely within the domain of *image segmentation*. Typically, image segmentation is concerned with *automated* methods for identifying object boundaries and regions of interest in images and is a subject that we will address in Chapter 10. In the remainder of this chapter we will be primarily concerned with the second aspect, namely how we can define meaningful descriptions of feature shapes in images and manipulate them to achieve our goals. Thus, we will implicitly assume either that *manual segmentation* (in which an operator/observer can define the region of interest) is appropriate to the problem in question or that an automated method to delineate the boundary is available.

7.2 Shape-preserving transformations

Translating an object from one place to another, *rotating* it or *scaling* (magnifying) it are all operations which change the object's shape vector coordinates but *do not* change its essential shape (Figure 7.1). In other words, the shape of an object is something which is basically defined by its boundary but which is *invariant to the translation, rotation and scaling of the coordinates* that define that boundary. Accordingly, we will adopt the following definition of shape:

- *Shape* is all the geometrical information that remains after location, scale and rotation effects have been filtered out from an object.

It follows that objects which actually have identical shape may have shape vectors which are quite different and the simple vector description of a shape as given in Section 7.1 is *partially redundant*. If we wish to describe and compare shapes in a compact and meaningful way, therefore, we must seek a *minimalist* representation which removes this redundancy. Such a representation can be achieved by the process known as *Procrustes alignment*. This procedure effectively applies an appropriate sequence of the three shape-preserving transformations of translation, scaling and rotation to two (or more) shape vectors to match them as closely as possible to each other, thereby filtering out all the non-essential differences between their shapes. This important procedure is discussed in Section 7.6. First, however, we consider a simple framework which permits the mathematical treatment of shape within the framework of matrix algebra.

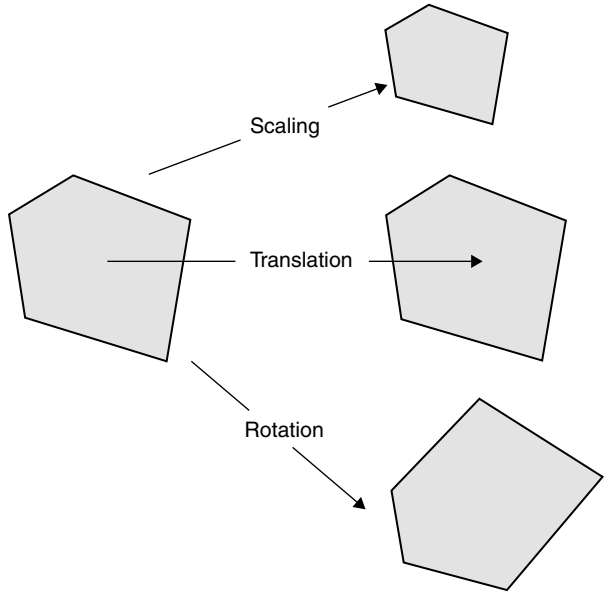

Figure 7.1 Shape is preserved under the linear operations of scaling, translation and rotation

7.3 Shape transformation and homogeneous coordinates

Shape can best be represented as ordered arrays in which the N Euclidean coordinate pairs, $\{(x_1, y_1) \cdots (x_N, y_N)\}$, are written as the columns of a matrix $\mathbf{S}$:

$$\mathbf{S} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_N \\ y_1 & y_2 & y_3 & \cdots & y_N \end{bmatrix} \tag{7.1}$$

The representation $\mathbf{S}$ is often referred to as a *point distribution matrix* (PDM), since each column gives the coordinates of one point in the overall distribution. The advantage of this arrangement is that linear transformations of the shape can be achieved by simple matrix multiplication.

In general, the result of applying a 2×2 transforming matrix $\mathbf{T}$ to the PDM $\mathbf{S}$ is to produce a new set of shape coordinates $\mathbf{S}'$ given by

$$\mathbf{S}' = \mathbf{TS} \tag{7.2}$$

with the individual coordinate pairs transforming as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix} \qquad \text{or} \qquad \mathbf{x}' = \mathbf{Tx} = \begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{7.3}$$

and with the parameters α_{ij} controlling the specific form of the transformation.

For example, operations such as overall scaling in two dimensions, shearing in the x and y directions and rotation about the origin through angle θ can all be expressed in the 2×2 matrix form described by Equation (7.3). Explicitly, these matrices are respectively given by:

$$\mathbf{T}_{sc} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \qquad \mathbf{T}_x = \begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix} \qquad \mathbf{T}_y = \begin{bmatrix} 1 & \alpha \\ 0 & 1 \end{bmatrix} \qquad \mathbf{T}_\theta = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{7.4}$$

where the scalars s and α are scaling and shear factors respectively.

Note, however, that the simple operation of point translation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \alpha_x \\ \alpha_y \end{bmatrix} \qquad \mathbf{x}' = \mathbf{x} + \mathbf{d} \tag{7.5}$$

cannot be accomplished through a single matrix multiplication as per Equation (7.3), but rather only with a further *vector addition*.

The operation of translation is very common. Most often, the need is to refer a set of coordinates to the origin of our chosen coordinate system, apply one or more transformations to the data and then translate it back to the original system. It is highly desirable, for practical reasons, to be able to express all these linear operations (scaling, rotation, shearing *and* translation) through the same mathematical operation of matrix multiplication. This can be achieved through the use of *homogeneous coordinates*.

In homogeneous coordinates, we express 2-D shape vectors

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

in a space of one higher dimension as

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

with w an arbitrary constant. For our purposes, we select $w = 1$ so that 2-D shape vectors in homogeneous coordinates will be given by the general form

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Similarly, the general 2-D transformation matrix $\mathbf{T}$ previously described by Equation (7.3), will be expressed as a 3×3 matrix (one additional dimension $\Rightarrow$ one extra row and one extra column) in the form

$$\mathbf{T} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \tag{7.6}$$

The reader may well ask at this point why it is reasonable to introduce homogeneous coordinates in this seemingly arbitrary way.[1] On one level, we can think of homogeneous coordinates simply as a computational artifice that enables us to include the operation of translation within an overall matrix expressing the linear transformation. For practical purposes, this is adequate and the *utility* of this will become apparent in the following sections. There is, however, a deeper underlying meaning to homogeneous coordinates which relates to the use of projective spaces as a means of removing the special role of the origin in a Cartesian system. Unfortunately, a thorough discussion of this topic lies outside the scope of this text.

7.4 The general 2-D affine transformation

Consider a coordinate pair (x, y) which is subjected to a linear transformation of the form

$$\begin{aligned} x' &= T_x(x, y) = ax + by + c \\ y' &= T_y(x, y) = dx + ey + f \end{aligned} \tag{7.7}$$

where a, b, c, d, e and f are arbitrary coefficients. Such a transformation is the 2-D (linear) affine transformation.

As is evident from its form, the affine transformation in 2-D space has six free parameters. Two of these (c and f) define the vector corresponding to translation of the shape vector, the other four free parameters (a, b, d and e) permit a combination of rotation, scaling and shearing.

From a geometrical perspective, we may summarize the effect of the general affine transformation on a shape as follows:

> *The affine transformation*: translation, rotation, scaling, stretching and shearing may be included. Straight lines remain straight and parallel lines remain parallel but rectangles may become parallelograms.

It is important to be aware that the *general* affine transformation (in which no constraints are placed on the coefficients) permits shearing and, thus, *does not preserve shape*.

The 2-D affine transformation in regular Cartesian coordinates (Equation (7.7)) can easily be expressed in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix} \qquad \mathbf{x}' = \mathbf{T}\mathbf{x} + \mathbf{d} \tag{7.8}$$

Typically, the affine transformation is applied not just on a single pair of coordinates but on the many pairs which constitute the PDM. The matrix-vector form of Equation (7.8)

[1] This addition of one dimension to express coordinate vectors in homogeneous coordinates extends to 3-D (i.e. $[x, y, z] \rightarrow [x, y, z, 1]$) and higher.

is thus inconvenient when we wish to consider a *sequence of transformations*. We would ideally like to find a single operator (i.e. a single matrix) which expresses the overall result of that sequence and simply apply it to the input coordinates. The need to add the translation vector **d** in Equation (7.8) after each transformation prevents us from doing this. We can overcome this problem by expressing the affine transformation in *homogenous coordinates*.

7.5 Affine transformation in homogeneous coordinates

In the homogeneous system, an extra dimension is added so that the PDM **S** is defined by an augmented matrix in which a row of 1s are placed beneath the 2-D coordinates:

$$\mathbf{S} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_N \\ y_1 & y_2 & y_3 & \cdots & y_N \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{7.9}$$

The affine transformation in homogeneous coordinates takes the general form

$$\mathbf{T} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ 0 & 0 & 1 \end{bmatrix} \tag{7.10}$$

where the parameters α_{13} and α_{23} correspond to the translation parameters (c and f) in Equation (7.7) and the other block of four

$$\begin{bmatrix} \alpha_{11} & \alpha_{12} \\ \alpha_{21} & \alpha_{22} \end{bmatrix}$$

correspond directly to a, b, d and e.

Table 7.1 summarizes how the parameters in the affine transform are chosen to effect the operations of translation, rotation, scaling and shearing. Figure 7.2 demonstrates the effect of each transformation on a square whose centre is at the origin and has a side of length equal to 2 units.

Table 7.1 Coefficient values needed to effect the linear transformations of translation, rotation, scaling and shearing in homogeneous coordinates

Transformation	α_{11}	α_{12}	α_{13}	α_{21}	α_{22}	α_{23}
Translation by (x, y)	1	0	x	0	1	y
Rotation by θ	$\cos\theta$	$-\sin\theta$	0	$\sin\theta$	$\cos\theta$	0
Uniform scaling by s	s	0	0	0	s	0
Vertical shear by s	1	s	0	0	1	0
Horizontal shear by s	s	0	0	s	1	0

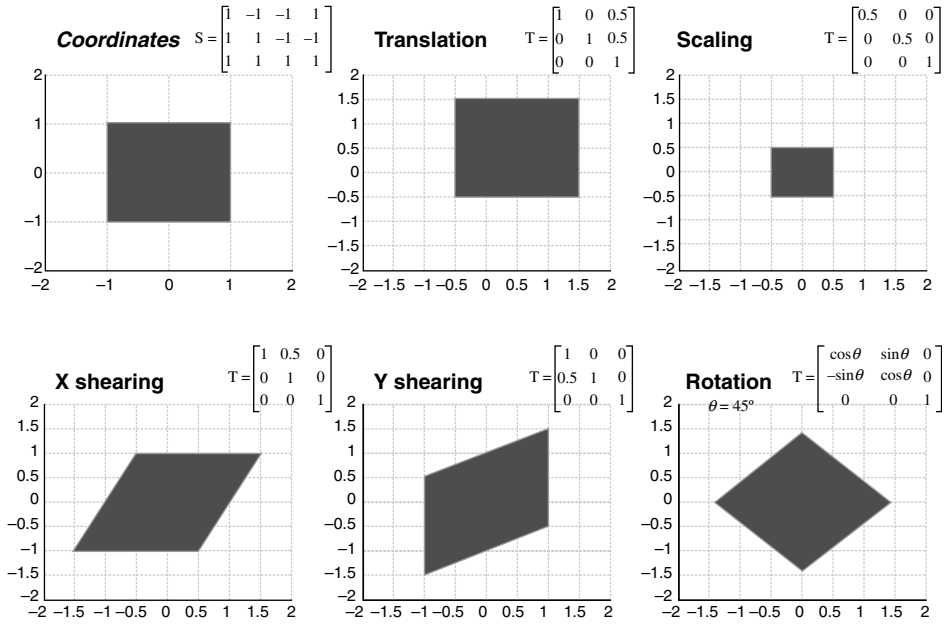

Figure 7.2 Basic linear transformations and their effects. The transformation matrices **T** and the original point distribution matrix **S** are expressed in homogeneous coordinates. (See http://www.fundipbook.com/materials/ for the Matlab code).

Using the homogeneous forms for the transformation matrix and the shape matrix, we can then express the transformed shape coordinates $\widehat{\mathbf{S}}$ as

$$\widehat{\mathbf{S}} = \mathbf{TS} \tag{7.11}$$

Note the following important property of the affine transformation:

- Any sequence of affine transformations reduces to a single affine transformation. A sequence of affine transformations represented by matrices $(\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_N)$ applied to an input point distribution **S** can be represented by a *single* matrix $\mathbf{T} = \mathbf{T}_1\mathbf{T}_2 \ldots \mathbf{T}_N$ which operates on **S**.

Unless explicitly stated otherwise, we will henceforth assume the use of homogeneous coordinates in this chapter.

7.6 The procrustes transformation

The Procrustes transformation matrix is a *special* case of the general affine transformation described by Equation (7.10) and is perhaps the most important form of the affine transform in the field of shape analysis. The key property of the Procrustes transformation

is that multiplication of any point distribution model by the Procrustes matrix will *preserve the shape*. For this reason, it is pivotal in many tasks of shape and image alignment. The general 2-D form in homogeneous coordinates is given by

$$\mathbf{T} = \begin{bmatrix} \alpha & \gamma & \lambda_1 \\ -\gamma & \alpha & \lambda_2 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.12}$$

The Procrustes matrix has only four free parameters (unlike the *general* affine form, which has six). This is because Procrustes transformation is a combination of the three shape-preserving operations of *translation*, *rotation* and *scaling* and we may decompose the Procrustes transformation into three successive, primitive operations of translation ($\mathbf{X}$), rotation ($\mathbf{R}$) and scaling ($\mathbf{S}$). Namely, $\mathbf{T} = \mathbf{SRX}$, where

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & \beta_x \\ 0 & 1 & \beta_y \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} S & 0 & 0 \\ 0 & S & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.13}$$

For this sequence to define the Procrustes transformation, translation must be applied first,[2] but the order of the rotation and scaling is unimportant. Multiplying the matrices in Equation (7.13) together and equating to the general form defined by Equation (7.12), we have

$$\begin{aligned} \alpha &= S\cos\theta & \beta &= S\sin\theta \\ \lambda_1 &= S(\beta_x\cos\theta + \beta_y\sin\theta) & \lambda_2 &= S(-\beta_x\sin\theta + \beta_y\cos\theta) \end{aligned} \tag{7.14}$$

The most common use of the Procrustes transformation is in the procedure known as *Procrustes alignment*. This involves the alignment of one or more shapes to a particular *reference shape*. The criterion for alignment is to find that combination of translation, rotation and scaling (i.e. the four free parameters β_x, β_y, θ and S) which minimize the sum of the mean-square distances between corresponding points on the boundary of the given shape and the reference.

7.7 Procrustes alignment

The Procrustes alignment procedure is conceptually straightforward and summarized diagrammatically in Figure 7.3. The mathematical derivation presented below yields the precise combination of translation, scaling and rotation that is required to achieve the minimization.

Consider two corresponding sets of N coordinates ordered as the columns of the point distribution matrices $\mathbf{X}$ and $\mathbf{Y}$. In a 2-D space, these sets are written as:

[2] This is essential because the subsequent rotation is defined about the origin and rotation on a point model whose centroid is not at the origin will *change the shape* of the point model.

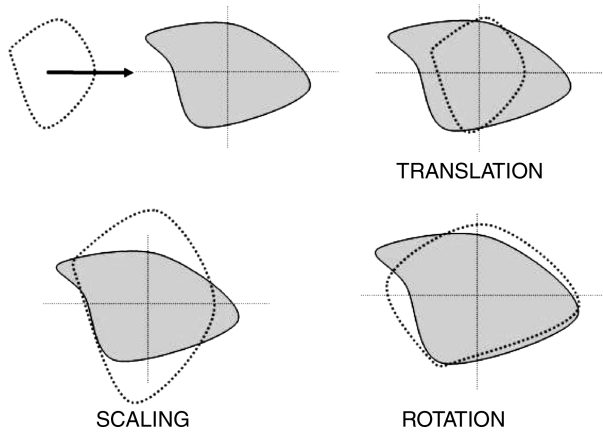

Figure 7.3 Procrustes alignment translates, scales and rotates a shape so as to minimise the sum of the squared distances between the coordinates of the two shapes

$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & \cdots & \cdots & x_N \\ y_1 & y_2 & \cdots & \cdots & y_N \end{bmatrix} \quad \text{and} \quad \mathbf{X}Y = \begin{bmatrix} x'_1 & x'_2 & \cdots & \cdots & x'_N \\ y'_1 & y'_2 & \cdots & \cdots & y'_N \end{bmatrix} \tag{7.15}$$

where $\vec{\mathbf{x}}_i$ and $\vec{\mathbf{x}}'_i$ are the ith points (and thus ith column vectors) of matrices $\mathbf{X}$ and $\mathbf{Y}$ respectively.

The aim is to transform (i.e. align) the *input* coordinates $\mathbf{X}$ to the *reference* coordinates $\mathbf{Y}$ so as to minimize *the total sum of the squared Euclidean distances between the corresponding points*. The three transformations of translation, scaling and rotation are *successively* applied, each independently satisfying this least-squares criterion.

Step 1: translation The aim is to find that global translation $\vec{\mathbf{t}}$ of the coordinates in matrix $\mathbf{X}$ which minimizes the total sum of the squared Euclidean distances between the translated coordinates and their corresponding values in the reference matrix $\mathbf{Y}$. In other words:

$$\vec{\mathbf{x}}_i \rightarrow \vec{\mathbf{x}}_i + \vec{\mathbf{t}}$$

and we seek to minimize the least-squares cost function Q, defined as

$$Q = \sum_{i=1}^{N} \left[\vec{\mathbf{x}}_i + \vec{\mathbf{t}} - \vec{\mathbf{x}}'_i\right]^{\mathbf{T}} \left[\vec{\mathbf{x}}_i + \vec{\mathbf{t}} - \vec{\mathbf{x}}'_i\right] \tag{7.16}$$

by solving for $\vec{\mathbf{t}}$, the translation vector, which achieves this. It is easy to show that

$$\vec{\mathbf{t}} = \langle \vec{\mathbf{y}} \rangle - \langle \vec{\mathbf{x}} \rangle \tag{7.17}$$

where $\langle \vec{\mathbf{x}} \rangle$ and $\langle \vec{\mathbf{y}} \rangle$ are the average (centroid) coordinate vectors of the respective sets of N points.

Note that it is common in the alignment procedure to first refer the reference coordinates (matrix $\mathbf{Y}$) to the origin, in which case $\langle\vec{\mathbf{y}}\rangle = 0$. In this case, the first step in the Procrustes alignment simply subtracts the sample mean value from each of the coordinates and the required translation is described by

$$\text{Step 1:} \quad \vec{\mathbf{x}}_i \rightarrow \vec{\mathbf{x}}_i - \langle\vec{\mathbf{x}}\rangle \tag{7.18}$$

Step 2: scaling A uniform scaling of all the coordinates in $\mathbf{X}$ can be achieved by a *diagonal* matrix $\mathbf{S} = s\mathbf{I}$, where s is the scaling parameter and $\mathbf{I}$ is the identity matrix. Minimization of the least-squares cost function

$$Q = \sum_{i=1}^{N} [\mathbf{S}\vec{\mathbf{x}}_i - \vec{\mathbf{x}}'_i]^{\mathrm{T}} [\mathbf{S}\vec{\mathbf{x}}_i - \vec{\mathbf{x}}'_i] \tag{7.19}$$

with respect to the free parameter s yields the solution

$$s = \frac{\sum_{i=1}^{N} \vec{\mathbf{x}}'^{\mathrm{T}}_i \vec{\mathbf{x}}_i}{\sum_{i=1}^{N} \vec{\mathbf{x}}^{\mathrm{T}}_i \vec{\mathbf{x}}_i} \tag{7.20}$$

$$\text{Step 2:} \quad \mathbf{X} \rightarrow \mathbf{SX} \text{ with } \mathbf{S} = s\mathbf{I} \text{ and } s \text{ given by Equation (7.20)} \tag{7.21}$$

Step 3: rotation The final stage of the alignment procedure is to identify the appropriate orthonormal rotation matrix $\mathbf{R}$. We define an error matrix $\mathbf{E} = \mathbf{Y} - \mathbf{RX}$ and seek that matrix $\mathbf{R}$ which minimizes the least-squares cost Q given by

$$Q = \mathrm{Tr}\{\mathbf{E}^{\mathrm{T}}\mathbf{E}\} = \mathrm{Tr}\{(\mathbf{Y} - \mathbf{RX})^{\mathrm{T}}(\mathbf{Y} - \mathbf{RX})\} \tag{7.22}$$

where Tr denotes the trace operator (the sum of the diagonal elements of a square matrix).

In this final step, the minimization of Equation (7.22) with respect to a variable matrix $\mathbf{R}$ is rather more involved than the translation and scaling stages. It is possible to show that our solution is

$$\mathbf{R} = \mathbf{VU}^{\mathrm{T}} \tag{7.23}$$

where $\mathbf{U}$ and $\mathbf{V}$ are eigenvector matrices obtained from the singular value decomposition (SVD) of the matrix $\mathbf{XY}^{\mathrm{T}}$. (Computationally, the SVD algorithm decomposes the matrix $\mathbf{XY}^{\mathrm{T}}$ into the product of three orthogonal matrices $\mathbf{U}$, $\mathbf{S}$ and $\mathbf{V}$ such that $\mathbf{XY}^{\mathrm{T}} = \mathbf{USV}^{\mathrm{T}}$.)[3]

[3] A detailed and comprehensive derivation of the Procrustes alignment equations for the general N-dimensional case is available at http://www.fundipbook.com/materials/.

$$\text{Step 3:} \quad \mathbf{X} \rightarrow \mathbf{RX} \text{ with } \mathbf{R} \text{ given by Equation(7.23)} \tag{7.24}$$

In summary, the solution steps for Procrustes alignment are as follows.

- *Translation:* place the origin at the centroid of your reference coordinates (that set of points to which you wish to align) and translate the input coordinates (the points you wish to align) to this origin by subtracting the centroids as per Equation (7.18).
- *Scaling* then scale these coordinates as per Equation (7.21).
- *Rotation*:
  - form the product of the coordinate matrices $\mathbf{XY}^{\mathrm{T}}$
  - calculate its SVD as $\mathbf{XY}^{\mathrm{T}} = \mathbf{USV}^{\mathrm{T}}$
  - calculate the matrix $\mathbf{R} = \mathbf{VU}^{\mathrm{T}}$
  - then multiply the translated and scaled coordinates by $\mathbf{R}$ as per Equation (7.24).

The first plot in Figure 7.4 shows two configurations of points, both approximately in the shape of five-pointed stars but at different scale, rotation and location. The shape vector to be aligned is referred to as the *input*, whereas the shape vector to which the inputs are aligned is called the *base*. The second plot shows them after one has been Procrustes aligned to the other. The code that produced Figure 7.4 is given in Example 7.1.

The Procrustes transformation preserves shape and, thus, does not permit stretching or shearing – it is consequently a special case of the *affine* transformation. Stretching and shearing have an important role in image processing too, particularly in the process of *image registration*, but we will address this later. The other major class of linear transformation is the *projective* transform, to which we now turn.

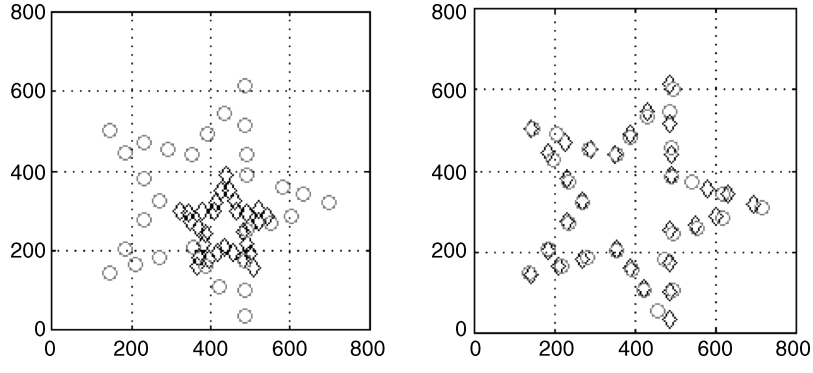

Figure 7.4 Procrustes alignment of two shapes: rotation, scaling and translation are applied to minimize the sum of the squared differences between corresponding pairs of coordinates

Example 7.1

Matlab code	What is happening?
load procrustes_star.mat;	%load coordinates of two shapes
whos	%input_points and base_points
subplot(1,2,1),	
plot(base_points(:,1),base_points(:,2),'kd'); hold on;	%Plot the shape coordinates
plot(input_points(:,1),input_points(:,2),'ro'); axis square; grid on	
[D,Z,transform] = procrustes(input_points, base_points);	%Procrustes align input to base
subplot(1,2,2),	
plot(input_points(:,1),input_points(:,2),'kd'); hold on;	
plot(Z(:,1),Z(:,2),'ro'); axis square; grid on; hold off;	%Plot aligned coordinates

Comments

Matlab functions: ***procrustes***.

The input to the Procrustes function is the two point distributions, i.e. sets of shape coordinates that need to be aligned.

The output variables comprise:

D	the sum of the squared differences between corresponding coordinate pairs. This provides some measure of the similarity between the shapes.
Z	the coordinates of the points after alignment to the reference.
transform	a Matlab structure containing the explicit forms for the rotation, scaling and translation components that effected the alignment.

Type ≫ *doc procrustes* at the Matlab prompt for full details on the procrustes function.

7.8 The projective transform

If a camera captures an image of a 3-D object, then, in general, there will be a *perspective* mapping of points on the object to corresponding points in the image. Pairs of points that are the *same distance* apart on the object will *be nearer or further apart* in the image depending on their distance from the camera and their orientation with respect to the image plane. In a perspective transformation we consider 3-D *world* coordinates (X, Y, Z) which are arbitrarily distributed in 3-D space and mapped (typically) to the image plane of a camera. However, in those situations in which the points of interest in the object can be considered as confined (either actually or approximately) to a plane and we are interested in the mapping of those object points into an arbitrarily oriented image plane, the required transformation is termed *projective*.

The form of the projective transform is completely determined by considering how one arbitrary quadrilateral in the object plane maps into another quadrilateral in the image plane. Thus, the task reduces to the following:

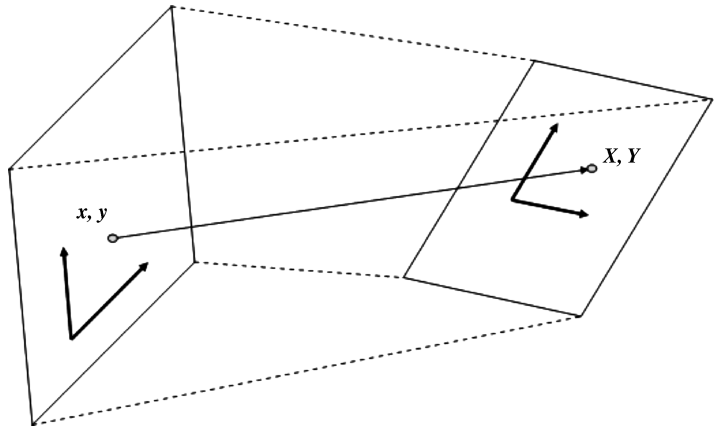

Figure 7.5 The projective transform is defined by the mapping of one arbitrary quadrilateral in the object plane into another in the image plane where the relative orientation of the quadrilaterals is unconstrained. The coordinates (x, y) are the image plane coordinates whereas (X, Y) are referred to as the *world* coordinates of the object point

Given the coordinates of the four corners of both quadrilaterals, compute the projective transform which maps an arbitrary point within one quadrilateral to its corresponding point in the other (see Figure 7.5).

Since this mapping is constrained at four 2-D points, there are eight coordinates and thus eight degrees of freedom in a projective transform. It is possible to show that the general form of the 2-D projective transformation matrix in a homogeneous coordinate system is given by:

$$\mathbf{T} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix}$$

and the transformed (projected) coordinates in the image plane are related to the world points through matrix multiplication as:

$$\widehat{\mathbf{S}} = \mathbf{TS} \rightarrow \begin{bmatrix} x'_1 & \cdots & x'_N \\ y'_1 & \cdots & y'_N \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} x_1 & \cdots & x_N \\ y_1 & \cdots & y_N \\ 1 & 1 & 1 \end{bmatrix} \tag{7.25}$$

Note that there are eight degrees of freedom in the projective matrix (not nine), since the parameters are constrained by the relation

$$\alpha_{31}x + \alpha_{32}y + \alpha_{33} = 1 \tag{7.26}$$

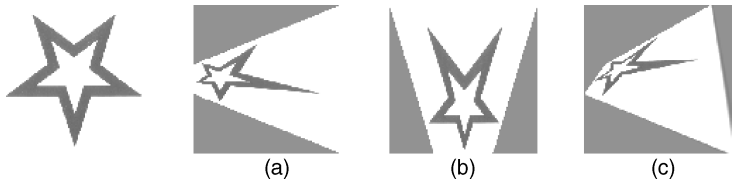

Figure 7.6 Examples of projective transformation. The original image on the left is projectively transformed to examples (a), (b) and (c)

Figure 7.6 shows the visual effect of some projective transforms. In this case, we have a 2-D (and thus planar) object. We define a square quadrilateral in the original object with coordinates $\{0, 0; 1, 0; 1, 1; 0, 1\}$ and demand that these coordinates transform linearly to the corresponding quadrilaterals defined in Table 7.2. The required transform matrix is calculated by inverting Equation (7.25) subject to the constraint expressed by Equation (7.26).

Projective transforms are useful for registering or aligning images or more generally 'scenes' which can be approximated as flat obtained from different viewpoints. Figure 7.6 illustrates the use of projective transformation as a means to register an image with respect to a different viewpoint. The Matlab® code which produced Figure 7.7 is given in Example 7.2.

Example 7.2

Matlab code	**What is happening?**
A = imread('plate_side.jpg');	%Read image to be registered
figure, imshow(A);	%Display
[x,y] = ginput(4); input_points = [x y];	%Interactively define coords of input quadrilateral
figure, imshow('plate_reference.jpg')	%Read base reference)image.
[x,y] = ginput(4); base_points = [x y];	%Interactively define coords of base quadrilateral
t_carplate = cp2tform(input_points, base_points,'projective');	%Create projective transformation structure
registered = imtransform(A,t_carplate);	%Apply projective transform
B = imcrop(registered);	%Interactively crop result
figure, imshow(B)	%Display corrected image

Comments

Matlab functions: ***ginput***, ***cp2tform***, ***imtransform***, ***imcrop***.

Key functions in this example are ***cp2tform*** *and* ***imtransform***. ***cp2tform*** is a general purpose function for geometric transformation which in this case requires specification of the input and base coordinates. ***imtransform*** applies the transform structure to input image and copies the texture to the mapped locations. See Matlab documentation for further details.

Table 7.2 Coordinate mappings for the projective transform in Figure 7.5: a square in the object plane is mapped to the given quadrilateral in the projected plane

Original coordinates	$\{0, 0; 1, 0; 1, 1; 0, 1\}$
Transformed coordinates	
(a)	$\{0, 0.4; 1, 0; 1, 1; 0, 0.6\}$
(b)	$\{0, 1; 1, 0; 0.7, 1; 0.3, 1\}$
(c)	$\{0, 0.4; 1, 0; 1.2, 1; -0.2, 0.6\}$

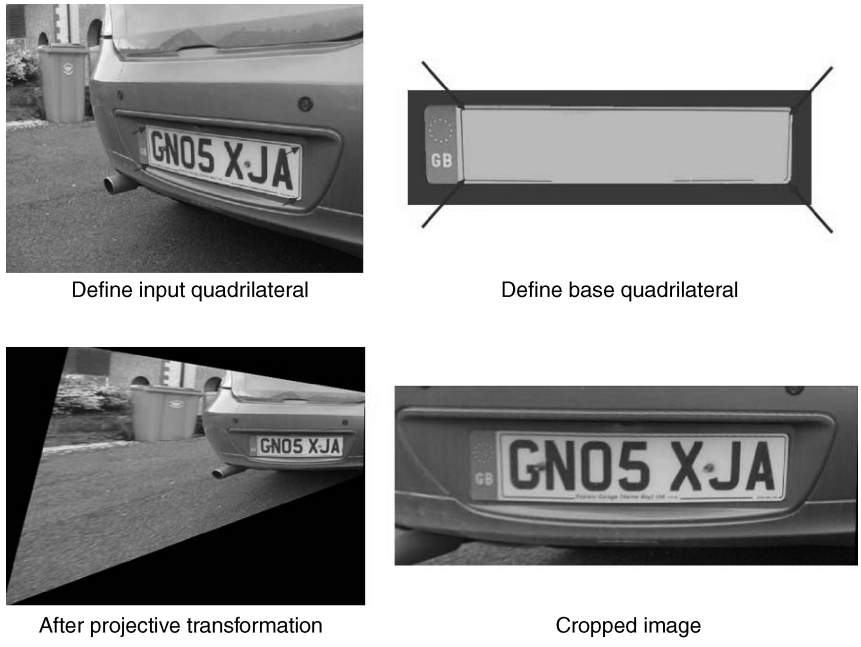

Figure 7.7 Projective transformation. In this example, the coordinates of the corners of the number plate and the reference image form respectively the input and base coordinates required to define the projective transform. Application of the transform results in registration of the plate with respect to the reference (base)

The projective transform is the most general linear transformation and only preserves straight lines in the input. A summary of the different linear transforms in order of increasing generality and the quantities which are preserved under their action is given in Table 7.3.

The geometric transformations described in this chapter so far are all linear transformations, in as much that the coordinates of any point in the transformed image are expressed as some linear combination of the coordinates in the input image. They are particularly appropriate to situations in which we have one or more examples of a shape or image and we wish to align them according to a set rule or method. In the remainder of this chapter, we will take a look at nonlinear transformations and then at piecewise transformation. In this latter technique, an input and a reference image are divided into corresponding triangular regions

Table 7.3 A summary detailing preserved quantities and degrees of freedom of various linear transformations. Each of the transformations is a special case of the successively more general case detailed beneath it

Transformation	Permissible operations	Preserved quantities	Degrees of freedom	
			in 2-D	in 3-D
Euclidean	Rotation and translation	Distances Angles Parallelism	3	6
Similarity	Rotation, scaling and translation	Angles Parallelism	4	7
Affine	Rotation, scaling, shearing and translation	Parallelism	6	12
Projective	Most general linear form	Straight lines	8	15

and this may be used very effectively to achieve image transformations which are locally linear but globally nonlinear.

7.9 Nonlinear transformations

The linear transformations of shape coordinates we have discussed so far are very important. However, many situations occur in which we need to apply a *nonlinear* transformation on the set of input coordinates which constitute the shape vector. In general, we have:

$$x' = T_x(x, y) \qquad y' = T_y(x, y) \tag{7.27}$$

where T_x and T_y are any nonlinear function of the input coordinates (x, y). Whenever T_x and T_y are nonlinear in the input coordinates (x, y), then, in general, straight lines will *not remain straight* after the transformation. In other words, the transformation will introduce a degree of *warping* in the data. One form which can be made to model many nonlinear distortions accurately is to represent T_x and T_y as second-order polynomials:

$$\begin{aligned} x' &= T_x(x, y) = a_0x^2 + a_1xy + a_2y^2 + a_3x + a_4y + a_5 \\ y' &= T_y(x, y) = b_0x^2 + b_1xy + b_2y^2 + b_3x + b_4y + b_5 \end{aligned} \tag{7.28}$$

A common form of nonlinear distortion is the pincushion or barrel effect often produced by lower quality, wide-angle camera lenses. This is a radially symmetric aberration, which is most simply represented in polar coordinates as:

$$\begin{aligned} r' &= T_r(r, \theta) = r + ar^3 \\ \theta' &= T_\theta(r, \theta) = \theta \end{aligned} \tag{7.29}$$

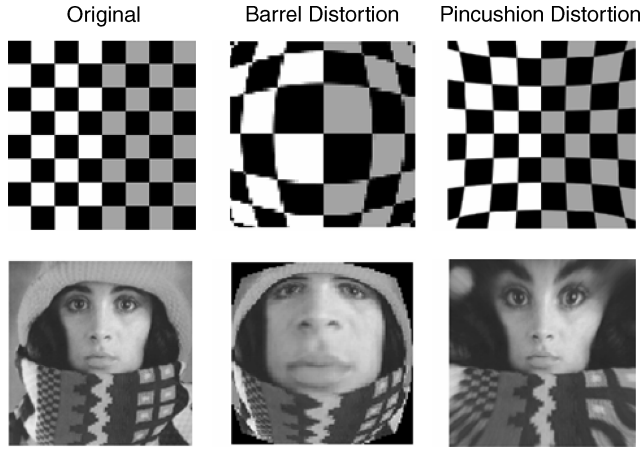

Figure 7.8 Barrel and pincushion distortion: These nonlinear transformations (greatly exaggerated in the examples above) are typical of the distortion that can be introduced by the use of poor-quality wide-angle lenses. The chessboard image of black, white and grey squares in the top row illustrates directly how the reference grid is distorted under the effect of this transformation

where $r = 0$ corresponds to the centre of the image frame. The pincushion ($a < 0$) and barrel ($a > 0$) distortion effects are effectively controlled by the magnitude of the coefficient a in Equation (7.29) and result in the 'hall of mirrors'-type[4] of distortions illustrated in Figure 7.8 (the Matlab code for which is given in Example 7.3).

Example 7.3

Matlab code	**What is happening?**
I = checkerboard(20,4);	%Read in image
%I = imread('trui.png');	%Read in image
[nrows,ncols] = size(I);	%Extract no. of cols and rows
[xi,yi] = meshgrid(1:ncols,1:nrows);	%Define grid
imid = round(size(I,2)/2);	%Find index of middle element
xt = xi(:) - imid;	%Subtract off and thus
yt = yi(:) - imid;	%shift origin to centre
[theta,r] = cart2pol(xt,yt);	%Convert from cartesian to polar
a = .0005;	%Set the amplitude of the cubic term
s = r + a.*r.^3;	%Calculate BARREL distortion
[ut,vt] = pol2cart(theta,s);	%Return the (distorted) Cartesian coordinates
u = reshape(ut,size(xi)) + imid;	%Reshape the coordinates to original 2-D grid
v = reshape(vt,size(yi)) + imid;	%Reshape the coordinates into original 2-D grid
tmap_B = cat(3,u,v);	%Assign *u* and *v* grids as the 2 planes of a 3-D array
resamp = makeresampler('linear', 'fill');	

[4] Some readers may recall visiting a room at a funfair or seaside pier in which curved mirrors are used to produce these effects. In my boyhood times on Brighton pier, this was called 'The hall of mirrors'.

```
I_barrel = tformarray(I,[],resamp,
    [2 1],[1 2],[],tmap_B,.3);
[theta,r] = cart2pol(xt,yt);          %Convert from cartesian to polar
a = −0.000015;                        %Set amplitude of cubic term
s = r + a.*r.^3;                      %Calculate PINCUSHION distortion
[ut,vt] = pol2cart(theta,s);          %Return the (distorted) Cartesian coordinates
u = reshape(ut,size(xi)) + imid;      %Reshape the coordinates into original 2-D grid
v = reshape(vt,size(yi)) + imid;      %Reshape the coordinates into original 2-D grid
tmap_B = cat(3,u,v);                  %Assign u and v grids as the 2 planes of a 3-D array
resamp = makeresampler('linear',      %Define resampling structure for use with
    'fill');                            tformarray
I_pin = tformarray(I,[],resamp,       %Transform image to conform to grid in tmap_B
    [2 1],[1 2],[],tmap_B,.3);
subplot(131); imshow(I);
subplot(1,3,2); imshow(I_barrel);
subplot(1,3,3), imshow(I_pin);
```

Comments

Matlab functions: ***meshgrid***, ***reshape***, ***cart2pol*** and ***pol2cart***.

Key functions in this example are ***makeresampler*** and ***tformarray***. tformarray is a general purpose function for geometric transformation of a multidimensional array and **makeresampler** produces a structure that specifies how tformarray will interpolate the data onto the transformed coordinate array. See Matlab documentation for further details.

7.10 Warping: the spatial transformation of an image

There are certain special applications where we are concerned only with the object shape and, therefore, it is satisfactory to treat the object simply as a set of geometric landmark points. However, the majority of real images also have intensity values which are associated with the spatial coordinates and it is the spatial transformation of these intensity values over the entire image that is the real goal in most practical applications. This mapping of intensity values from one spatial distribution into another spatial distribution is commonly known as *image warping*. Thus, spatially transforming or warping an image consists of both a geometrical/spatial transformation of landmark coordinates *and* the subsequent transfer of pixel values from locations in one image to corresponding locations in the warped image.

Typical warping applications require us to transform an image so that its shape (as defined by the landmarks) conforms to some corrected, reference form. The image which is to be transformed is termed the *input*. The normal situation is that we wish to transform the input image so that it conforms in some well-defined sense to a reference geometry. This geometry is defined by identifying a set of landmarks on a reference image which clearly correspond to those in the input. The reference image is termed the *base* image and the base landmarks effectively define the *target* spatial coordinates to which the input landmarks must be transformed. This is depicted in Figure 7.9. Note that a pair of corresponding landmarks in the input and base images are referred to as *tie-points* (as they are in a certain sense tied to each other).

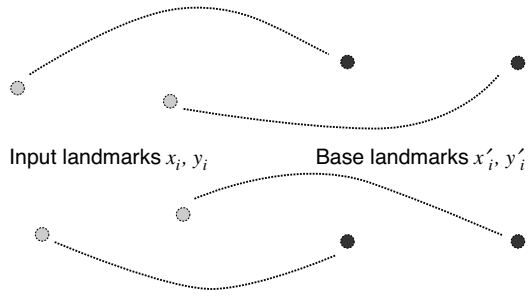

Figure 7.9 The central concept in geometric transformation or warping is to define a mapping between input and base landmarks

There are theoretically an infinite number of spatial transformations which we might apply to effect a warp, although, in practice, a relatively limited number are considered. Whatever the specific transformation we choose, the basic steps in the overall procedure are the same:

(1) Locate an equal number of matching landmarks on the image pair (x'_i, y'_i for the base image and x_i, y_i for the input). Note that matching landmarks are often called *tie-points* and we will use these terms interchangeably.

(2) Define a chosen functional form to map coordinates from the input to the base. Thus, $x' = f(x, y)$, $y' = g(x, y)$.

(3) Write a system of equations each of which constrains the required mapping from a tie-point in the input to its corresponding tie-point in the base. Thus, for N tie-points, we have

$$\begin{array}{ll} x'_1 = f(x_1, y_1) & y'_1 = g(x_1, y_1) \\ x'_2 = f(x_2, y_2) & y'_2 = g(x_2, y_2) \\ \vdots & \vdots \\ x'_N = f(x_N, y_N) & y'_N = g(x_N, y_N) \end{array}$$

(4) The chosen functional forms $f(x, y)$ and $g(x, y)$ will be characterized by one or more free (fit) parameters. We solve the system of equations for the free parameters.

(5) For every valid point in the base image, calculate the corresponding location in the input using the transforms f and g and copy the intensity value at this point to the base location.

This basic process is illustrated schematically in Figure 7.10.

Consider the simple example depicted in Figure 7.11 (produced using the Matlab code in Example 7.4). The aim is to transform the *input* image containing 10 landmarks shown top

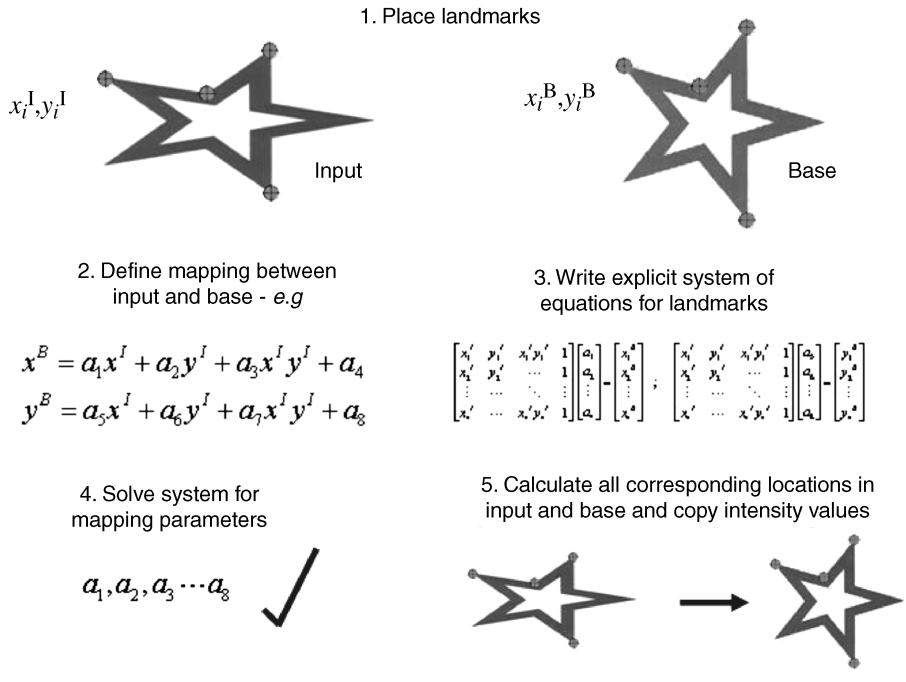

Figure 7.10 The basic steps in the warping transformation

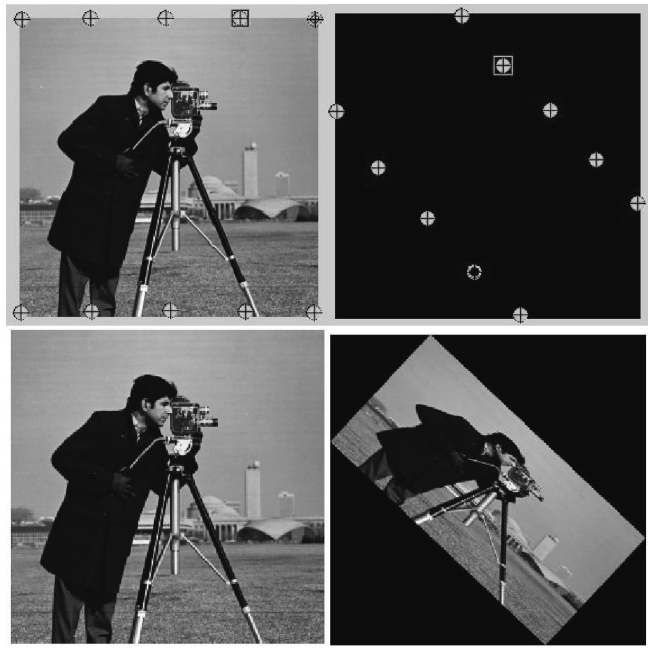

Figure 7.11 Top: the 10 input landmarks on the image on the left need to be transformed to conform to the 10 base landmarks on the right. An affine transformation (as defined by Equation (7.8) or equivalently by Equation (7.10) in homogeneous coordinates) can approximate this quite well. Bottom: the original and transformed images are displayed

Example 7.4

Matlab code	What is happening?
I = imread('cameraman.tif');	%Read in image
cpstruct=cpselect(I,I);	%Mark the tie points and save within gui %Gives input_points and output_points
tform = cp2tform(input_points, base_points,'affine');	%Infer affine transformation from tie points
B = imtransform(I,tform);	%Transform input
subplot(1,2,1), imshow(I),subplot(1,2,2), imshow(B)	%Display

Comments

Matlab functions: ***cpselect***, ***cp2tform***, ***imtransform***.

cpselect is a graphical tool which forms part of the image processing toolbox and which is designed to make the identification, placement and storage of landmarks on image pairs easier.

cp2tform is a function that calculates the transform parameters which best achieves the warp from the input to the base coordinates in a least-squares sense. The user must supply the input and base coordinates and specify the type of transform from a number of permissible options. The output is a Matlab structure which contains the parameters of the transform and associated quantities of use.

imtransform is a function that carries out the warping transformation. The user must supply the input image and the transform structure (typically provided by use of cp2tform).

See the Matlab documentation for more details of these functions and their usage.

left to conform to the *base* shape depicted by the corresponding landmarks on the top right. The input shape is characterized by two approximately horizontal lines and the base by two rotated and scaled but also approximately parallel lines. We might expect that a general affine transformation (which permits a combination of translation, shearing, scaling and rotation) will be effective in ensuring a smooth warp. The image shown bottom right, in which the input image has been mapped to its new location, shows this to be the case.

7.11 Overdetermined spatial transformations

The first step in image warping is to transform the input landmarks such that their new coordinate values will match the corresponding base landmarks. In general, it is rarely possible to achieve *exact* matching using a global spatial transform. The fundamental reason for this is that, whatever transformation function we choose to act on the input coordinates, it must always be characterized by *a fixed number of free parameters*. For example, in a bilinear spatial transform, the coordinates of the input are transformed to the base according to

$$\begin{aligned} x' &= f(x,y) = a_1x + a_2y + a_3xy + a_4 \\ y' &= g(x,y) = a_5x + a_6y + a_7xy + a_8 \end{aligned} \tag{7.30}$$

The bilinear transform is thus characterized by eight free parameters a_1–a_8. It follows that any attempt to warp the input coordinates (x, y) to precisely conform to the base (x', y') at more than four landmark pairs using Equation (7.30) cannot generally be achieved, since the resulting system has more equations than unknown parameters, i.e. it is *overdetermined*. In general, whenever the number of equations exceeds the number of unknowns, we cannot expect an exact match and must settle for a least-squares solution. A least-squares solution will select those values for the transform parameters which minimize the sum of the squared distances between the transformed input landmarks and the base landmarks. This effectively means that the input shape is made to conform *as closely as possible* to that of the base but cannot match it exactly.

Consider Figure 7.12, in which two different faces are displayed. Each face image has been marked with a total of 77 corresponding landmarks. The aim is to spatially transform image A on the left such that it assumes the shape of the base image on the right (B). Image C shows the original face and image D the result of a global spatial transformation using a second-order, 2-D polynomial form. Although the result is a smooth warp, careful comparison with image E (the base) indicates that the warp has been far from successful in transforming the shape of the features to match the base.

This is the direct result of our highly overdetermined system (77 landmark pairs, which is far greater than the number of free parameters in the global transformation function). The Matlab code which produces Figure 7.12 is given in Example 7.5. Speaking generally, global transformation functions (of which Equations (7.29) and (7.30) are just two possible examples), are a suitable choice when the distortion introduced into the imaging process has a relatively simple form and can be well approximated by a specific function. However, when the requirement is to warp an image to conform exactly to some new shape at a number of points (particularly if this is a large number of points), they are not suitable. The solution to this particular problem is through the *piecewise* method described in the next section.

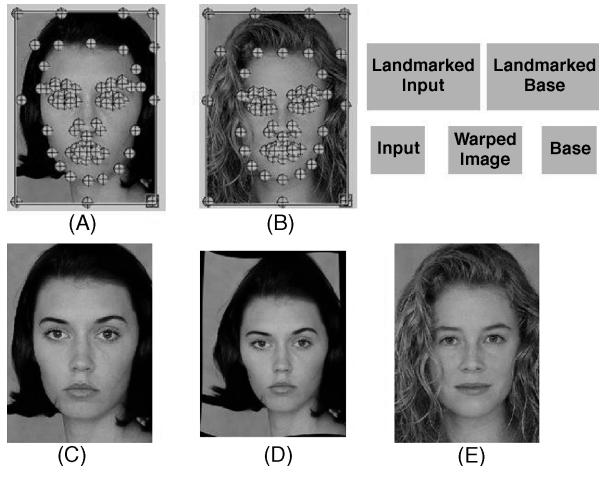

Figure 7.12 Warping using a global polynomial transformation. In this case, the warped image only approximates the tie-points in the base image

Example 7.5

Matlab code	**What is happening?**
load ('ripollfaces.mat')	%Load images and control point %structure
cpselect(rgb2gray(A),rgb2gray(B),cpstruct);	%Load up the tie points
tform = cp2tform(input_points,base_points, 'polynomial');	%Infer affine transformation from %tie-points
C = imtransform(A,tform);	%Transform input
subplot(1,2,1), imshow(I),subplot(1,2,2), imshow(C)	%Display

Comments
Matlab functions: ***cpselect***, ***cp2tform***, ***imtransform***.

7.12 The piecewise warp

In general, only the polynomial class of functions can be easily extended to higher order to allow exact matching at a larger number of tie-points (an nth-order polynomial has $n + 1$ free parameters). However, the fitting of a limited set of data points by a high-order polynomial generally results in increasingly wild and unpredictable behaviour in the regions between the tie-points. The inability to define reasonably behaved functions which can *exactly* transform the entire input image to the base at more than a modest number of landmarks but still maintain reasonable behaviour in between the landmark points is the major weakness of the global spatial transformations outlined in the previous section. A means of successfully overcoming this limitation of global, spatial transforms is the *piecewise* warp.

In the *piecewise* approach, the input and base coordinates are divided into a similar number of piecewise, contiguous regions in *one-to-one* correspondence. Typically, each such region is defined by a small number of landmarks; the major advantage is that it allows us to define a simple transformation function whose domain of validity is restricted to each small segment of the image. In this way, we can enforce an exact transformation of the input landmarks to the base values over the entire image plane.

7.13 The piecewise affine warp

The simplest and most popular piecewise warp implicitly assumes that the mapping is locally linear. Such an assumption becomes increasingly reasonable the larger the density of reliable landmark points which are available. In many cases, a linear, piecewise affine transformation produces excellent results, and its computational and conceptual simplicity are further advantages. The key to the piecewise approach is that the landmarks are used to divide the input and base image into an equal number of contiguous regions which

Figure 7.13 Delaunay triangulation divides the image piecewise into contiguous triangular regions. This is a common first stage in piecewise warping

correspond on a one-to-one basis. The simplest and most popular approach to this is to use Delaunay triangulation (see Figure 7.13), which connects an irregular point set by a mesh of triangles each satisfying the Delaunay property.[5] A division of the image into nonoverlapping quadrilaterals is also possible, though not as popular.

Once the triangulation has been carried out, each and every pixel lying within a specified triangle of the base image (i.e. every valid pair of base coordinates) is mapped to its corresponding coordinates in the input image triangle and the input intensities (or colour values as appropriate) are copied to the base coordinates.

Let us now consider the mapping procedure explicitly. Figure 7.14 depicts two corresponding triangles in the input and base images. Beginning (arbitrarily) at the vertex with position vector $\mathbf{x}_1$, we can identify any point within the triangle by adding suitable multiples of the shift vectors $(\mathbf{x}_2-\mathbf{x}_1)$ and $(\mathbf{x}_3-\mathbf{x}_1)$. In other words, any point $\mathbf{x}$ within the input region t can be expressed as a linear combination of the vertices of the triangle:

$$\begin{aligned}\mathbf{x} &= \mathbf{x}_1+\beta(\mathbf{x}_2-\mathbf{x}_1)+\gamma(\mathbf{x}_3-\mathbf{x}_1)\\ &= \alpha\mathbf{x}_1+\beta\mathbf{x}_2+\gamma\mathbf{x}_3\end{aligned} \tag{7.31}$$

where we define

$$\alpha+\beta+\gamma=1 \tag{7.32}$$

[5] This property means that no triangle has any points inside its circumcircle (the unique circle that contains all three points (vertices) of the triangle).

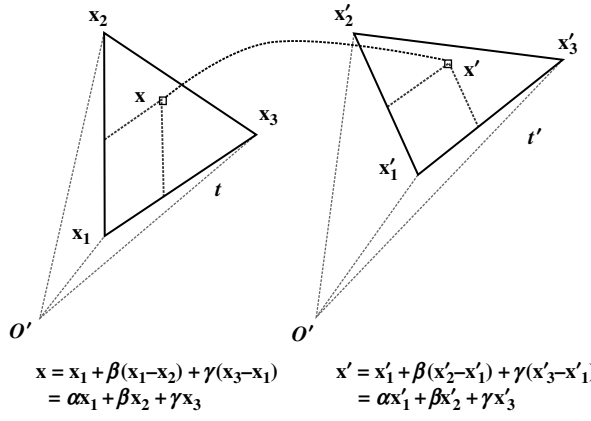

Figure 7.14 The linear, piecewise affine warp. Each point in the input image (vector coordinate **x**) is mapped to a corresponding position in the base (warped) image by assuming the texture variation from corresponding triangular regions is locally linear.

The required values of the parameters α, β and γ to reach an arbitrary point $\mathbf{x} = [x \quad y]$ within the input triangle can be obtained by writing Equations (7.31) and (7.32) in matrix form:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \mathbf{T}\underline{\alpha} = \mathbf{x} \tag{7.33}$$

and solving the resulting system of linear equations as

$$\underline{\alpha} = \mathbf{T}^{-1}\mathbf{x} \tag{7.34}$$

The corresponding point in the base (warped) image is now given by *the same linear combination* of the shift vectors in the partner triangle t'. Thus, we have

$$\begin{bmatrix} x'_1 & x'_2 & x'_3 \\ y'_1 & y'_2 & y'_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \qquad \mathbf{S}\underline{\alpha} = \mathbf{x}' \tag{7.35}$$

But in this case the coordinate matrix is **S** containing the vertices of the triangle t'.

Combining Equations (7.34) and (7.35), it thus follows that coordinates map from the input to the base as

$$\mathbf{x}' = \mathbf{S}\underline{\alpha} = \mathbf{S}\mathbf{T}^{-1}\mathbf{x} = \mathbf{M}\mathbf{x} \tag{7.36}$$

Figure 7.15 A linear piecewise affine warp (right) of the input (centre) to the base (left) using a total of 89 tie-points

Piecewise affine warp

(1) Divide the input and base images into piecewise, corresponding triangular regions using Delaunay triangulation.

(2) Calculate the mapping matrix $\mathbf{M}$ as defined by Equations (7.34)–(7.36) for each pair of corresponding triangles in the input and base images.

(3) Find all pixels lying within the base image triangle (all valid coordinates $\mathbf{x}_i$).

(4) Apply Equation (6.7) to all valid $\mathbf{x}_i$ in the base image to get the corresponding locations in the input image triangle ($\mathbf{x}'_i$).

(5) Set the pixel values in the base at coordinates $\mathbf{x}_i$ to the values in the input at coordinates $\mathbf{x}'_i$.

Figure 7.15 shows the result (right) of applying a linear piecewise affine warp of the input image (centre) to the base (left). The tie-points used for the warp are similar to those displayed in Figure 7.12. The vastly improved result is evident.

7.14 Warping: forward and reverse mapping

As we have seen, image warping is a geometric operation that defines a coordinate mapping between an input image and a base image and assigns the intensity values from corresponding locations accordingly. There are, however, *two distinct ways* of carrying out this process. If we take each coordinate location in the input and calculate its corresponding location in the base image, we are implicitly considering a forward mapping (i.e. from 'source' to 'destination'). Conversely, we may consider a reverse mapping ('destination' to 'source') in which we successively consider each spatial location in the base image and calculate its corresponding location in the source. Initially, forward mapping might appear the most logical approach. However, the piecewise affine procedure described in the previous section

employs *reverse* mapping. Digital images are discrete by nature and, thus, forward mapping procedures can give rise to problems:

(1) Depending on the specific transformation defined, pixels in the source may be mapped to positions beyond the boundary of the destination image.

(2) Some pixels in the destination may be assigned more than one value, whereas others may not have any value assigned to them at all.

The pixels with no values assigned to them are particularly problematic because they appear as 'holes' in the destination image that are aesthetically unpleasing (and must be filled through interpolation). In general, the greater the difference between the shape of the input and base triangles, the more pronounced this problem becomes. Reverse mapping, however, guarantees a single value for each and every pixel in the destination – there are no holes in the output image and no mapping out-of-bounds. This is the main advantage of reverse mapping.

For further examples and exercises see http://www.fundipbook.com