Part I

Geometrical Fundamentals

Chapter 3

Spline curves

Throughout this book, visual curves are represented in terms of parametric spline curves, as is common in computer graphics. These are curves $(x(s), y(s))$ in which s is a parameter that increases as the curve is traversed, and x and y are particular functions of s, known as splines. A spline of order d is a piecewise polynomial function, consisting of concatenated polynomial segments or *spans*, each of some polynomial order d, joined together at *breakpoints*. Parametric spline curves are attractive because they are capable of representing efficiently sets of boundary curves in an image (figure 3.1). Simple shapes can be represented by a curve with just a few spans. More complex shapes could be accommodated by raising the polynomial order d but it is preferable to increase the number of spans used. Usually the polynomial order is fixed at quadratic ($d = 3$) or cubic ($d = 4$)[1]. Maintaining a fixed, low polynomial degree, even in the face of geometric complexity, makes for computational stability and simplicity.

The chapter begins by explaining spline functions and their construction. Later sections explain how parametric curves are constructed from spline functions and introduce methods for matching one curve to another. This forms the basis for the algorithms developed in subsequent chapters for active contour matching.

[1]The *order* of a polynomial is the number of its coefficients. Hence a quadratic function $a+bx+cx^2$ has order $d = 3$. Its *degree* — the highest power of x — is 2.

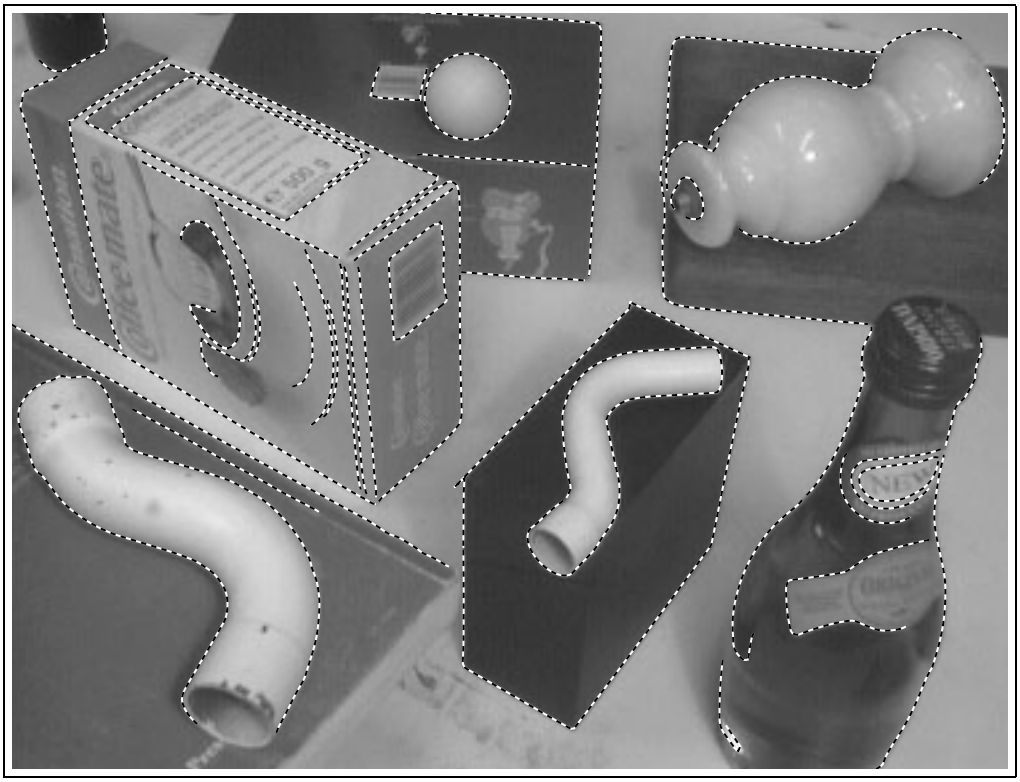


Figure 3.1: Image edges represented as parametric spline curves.

3.1 B-spline functions

B-splines are a particular, computationally convenient representation for spline functions. In the B-spline form, a spline function $x(s)$ is constructed as a weighted sum of N_B *basis functions* (hence 'B'-splines) $B_n(s),\ n = 0, \ldots, N_B - 1$. In the simplest ("regular") case, each basis function consists of d polynomials each defined over a *span* of the s-axis. We take each span to have unit length. The spans are joined at *knots* as in figure 3.2. It shows the simplest case in which the knots are evenly spaced and the joins between polynomials are *regular* — that is, as smooth as possible, having $d - 2$ continuous derivatives. The quadratic spline, for instance, has continuous gradient in the regular case. The constructed spline function is

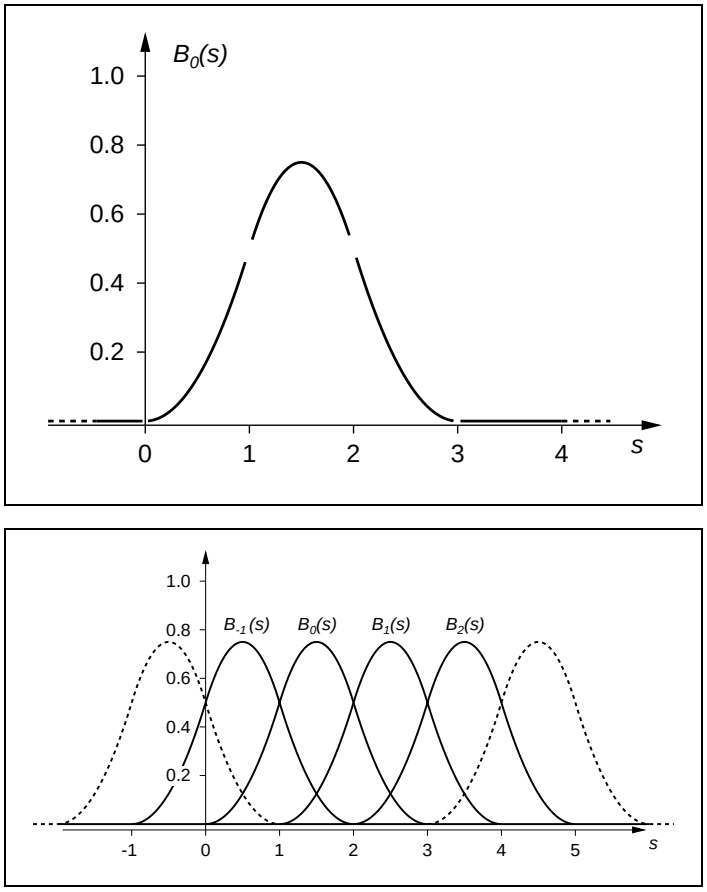


Figure 3.2: *(top) A single quadratic B-spline basis function $B_0(s)$. "Knots" at $s =, 0, 1, 2, 3, 4$ mark transitions between polynomial segments of the function. (bottom) In the regular case which has evenly spaced knots (at integral values of s), each B-spline basis function is a translated copy of the previous one.*

$$x(s) = \sum_{n=0}^{N_B-1} x_n B_n(s) \tag{3.1}$$

where x_n are the weights applied to the respective basis functions $B_n(s)$, as in figure 3.3. This can be expressed compactly in matrix notation as

$$x(s) = \mathbf{B}(s)^T \mathbf{Q}^x, \tag{3.2}$$

a matrix product between a vector of B-spline functions

$$\mathbf{B}(s) = (B_0(s), B_1(s), \ldots, B_{N_B-1}(s))^T \tag{3.3}$$

and a vector of weights

$$\mathbf{Q}^x = \begin{pmatrix} x_0 \\ \vdots \\ x_{N_B-1} \end{pmatrix}. \tag{3.4}$$

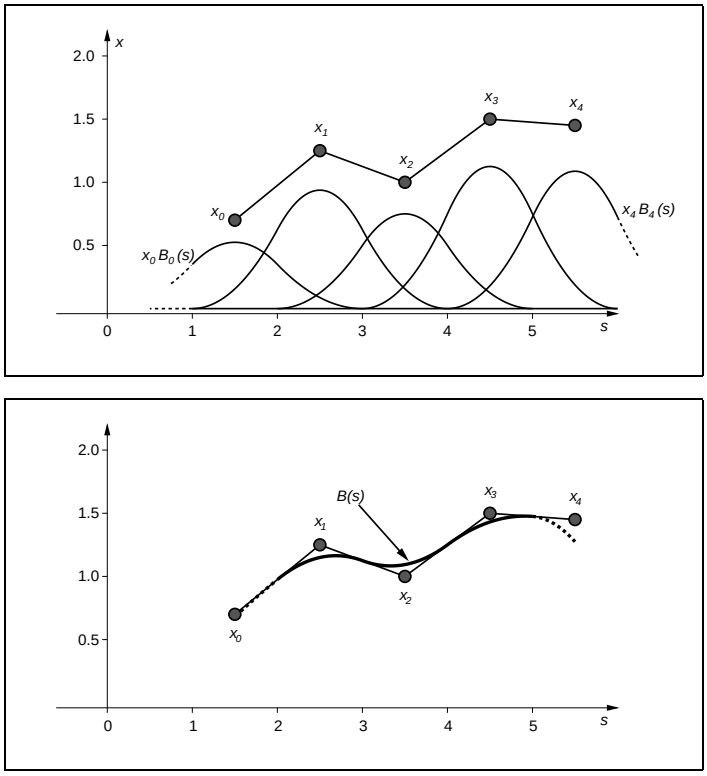


Figure 3.3: *B-spline basis functions $B_n(s)$ are weighted by coefficients x_n (top) and combined linearly to form a spline function $x(s)$ (bottom). Note that the spline function follows the "control polygon" closely.*

By convention, B-spline basis functions are constructed in such a way that they

sum to 1 at all points:

$$\sum_{n=0}^{N_B-1} B_n(s) = 1 \quad \text{for all } s. \tag{3.5}$$

This summation or "convex hull" property is the underlying reason that the B-spline function in figure 3.3 follows the "control polygon," made up of the points $(\frac{3}{2}, x_0), (\frac{5}{2}, x_1), \ldots$ quite closely.

In the simple case of a quadratic B-spline with knots spaced regularly at unit intervals, the first B-spline basis function has the form

$$B_0(s) = \begin{cases} s^2/2 & \text{if } 0 \le s < 1 \\ \frac{3}{4} - (s - \frac{3}{2})^2 & \text{if } 1 \le s < 2 \\ (s-3)^2/2 & \text{if } 2 \le s < 3 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

and the others are simply translated copies:

$$B_n(s) = B_0(s - n).$$

However, this basis is bi-infinite: there are infinitely many B_n and the functions $x(s)$ they are used to construct are bi-infinite, extending to $s \to \pm\infty$. For practical applications finite bases are needed.

3.2 Finite bases

A finite spline basis can be either periodic (figure 3.4) or aperiodic (figures 3.5 and 3.6) over a closed interval $0 \le s \le L$. The periodic basis is simply the bi-infinite basis suitably wrapped around. For example, the basis functions for regular, quadratic splines are $B_0, \ldots, B_{L-1}$ ($N_B = L$), defined as above, but treated as periodic over the interval $0 \le s \le L$. The four periodic basis functions for the case that $L = 4$ are illustrated in figure 3.4. A non-periodic basis on a finite interval is more complex to construct, requiring so-called "multiple knots" (see later) at its endpoints. This allows full control over boundary conditions — the value of the function $x(s)$ and its derivatives at the ends $s = 0, L$ of the interval. Details of the construction of the B_n for this case can be found in appendix A. It is no longer the case that the number of

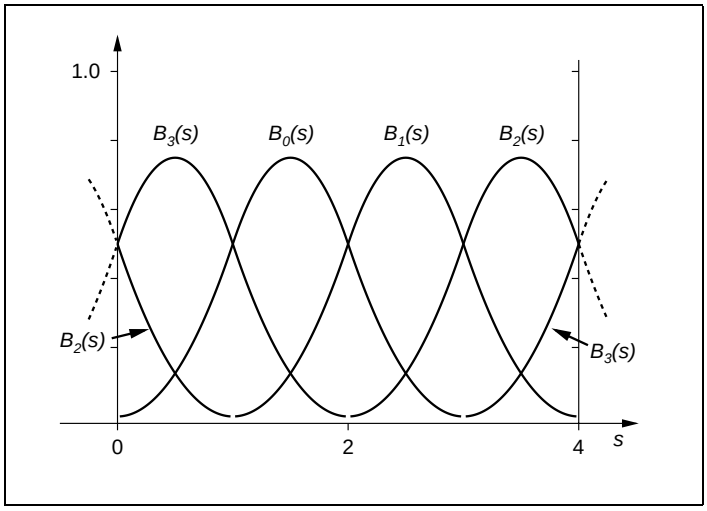


Figure 3.4: Periodic B-spline basis, *as figure 3.2 but for construction of functions that are periodic over the range* $0 \leq s \leq 4$. *Again each B-spline basis function is a translated copy of the previous one, but also wrapped around where the periodicity demands it.*

basis functions N_B is equal to the interval length L. Additional basis functions are needed to control boundary conditions (values of the spline function and its derivatives at $s = 0, L$). In the regular case, $d - 1$ extra functions are needed (d is the order of the polynomial) so that $N_B = L + d - 1$. In figure 3.5, for example, $L = 5$ and $d = 3$ (quadratic) so there must be $N_B = 7$ basis functions.

There is an efficient algorithm for generating spline functions from the weights x_n in which the basis functions are represented in terms of a matrix of polynomial coefficients. The standard method is given in appendix A.

3.3 Multiple knots

Sometimes it is desirable to allow a reduced degree of continuity at some point within the domain of a function $x(s)$. This can be achieved by forming a multiple knot, in which two knots in the B-spline basis approach one another and coincide. The spline then consists of a sequence of polynomial spans joined at *breakpoints*, some of which are single knots while others are multiple knots. At a regular breakpoint (single knot), the degree of smoothness is at its maximal value, that is C^{d-2} — continuity of

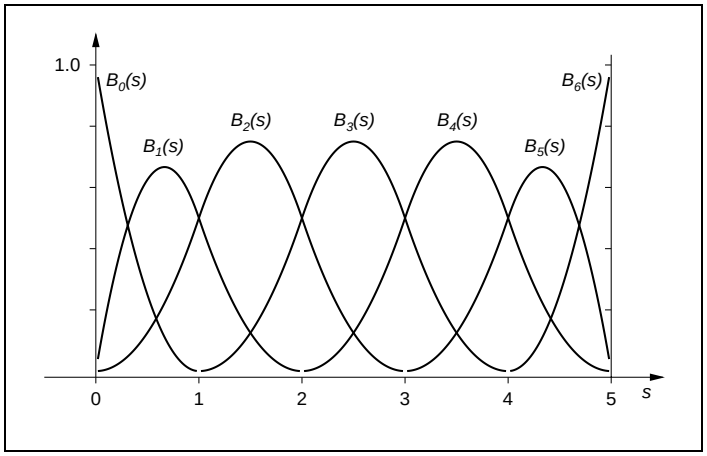


Figure 3.5: Spline basis over an interval. *Basis functions are not entirely composed of translated copies of one another, as they were in the bi-infinite case, but include special functions at the extremes of the interval for control of boundary conditions.*

all derivatives up to the $(d-2)$th. At a double knot, however, continuity is reduced to C^{d-3} and generally, continuity at a knot of multiplicity m is C^{d-m-1}. Forming a multiple knot is a limiting process in which m consecutive regular knots approach one another, as illustrated in figure 3.7 for the quadratic case. Once a double knot has been introduced into the basis, any constructed spline function generally loses one order of continuity at that breakpoint. In the quadratic case for instance, a spline function is C^0 at the knot: it remains continuous but its gradient becomes discontinuous — see figure 3.8 for an illustration. If a triple knot is introduced, the function becomes discontinuous, broken into two continuous pieces, one on each side of the knot. Hence triple knots are used to terminate a quadratic B-spline basis over a finite interval as in figure 3.5.

3.4 Norm and inner product for spline functions

It is very useful to be able to calculate the so-called "L_2-norm" $\|x\|$ of a function $x(s)$:

$$\|x\|^2 = \frac{1}{L}\int_0^L x(s)^2\,ds. \tag{3.7}$$

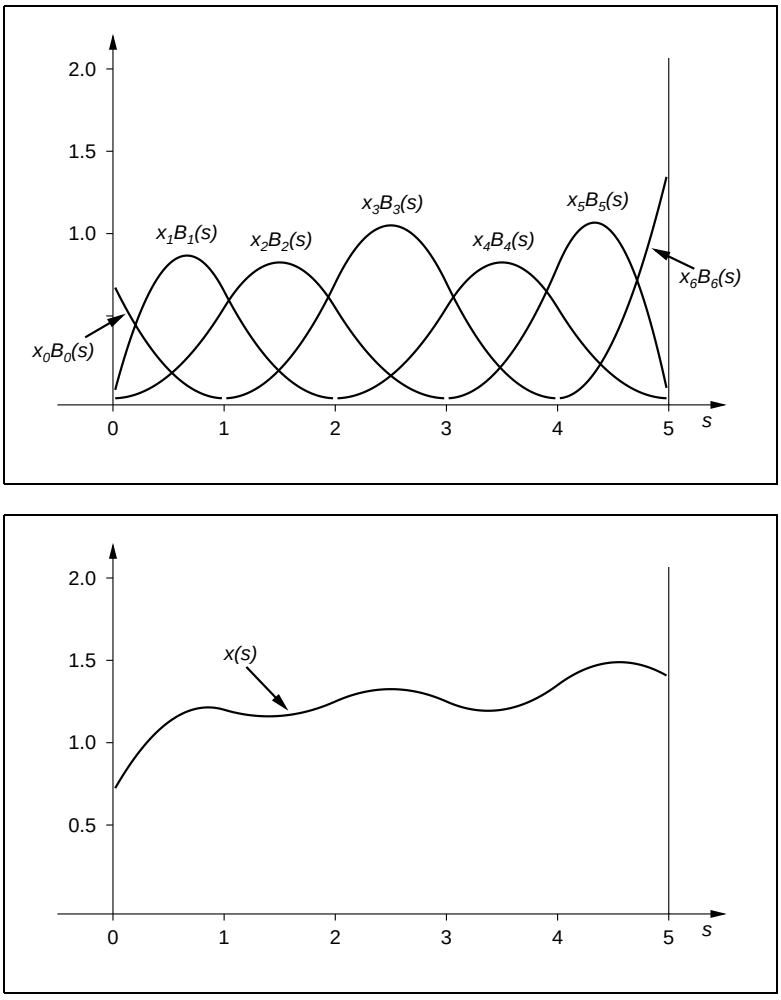


Figure 3.6: Splines over an interval. *Basis functions in figure 3.5 are blended to form a spline function $x(s)$. The choice of basis functions allows the boundary values $x(0), x(5)$ to be controlled directly by the weights x_0, x_6. Then weights x_1, x_5 control the values of derivatives at the extremes of the interval.*

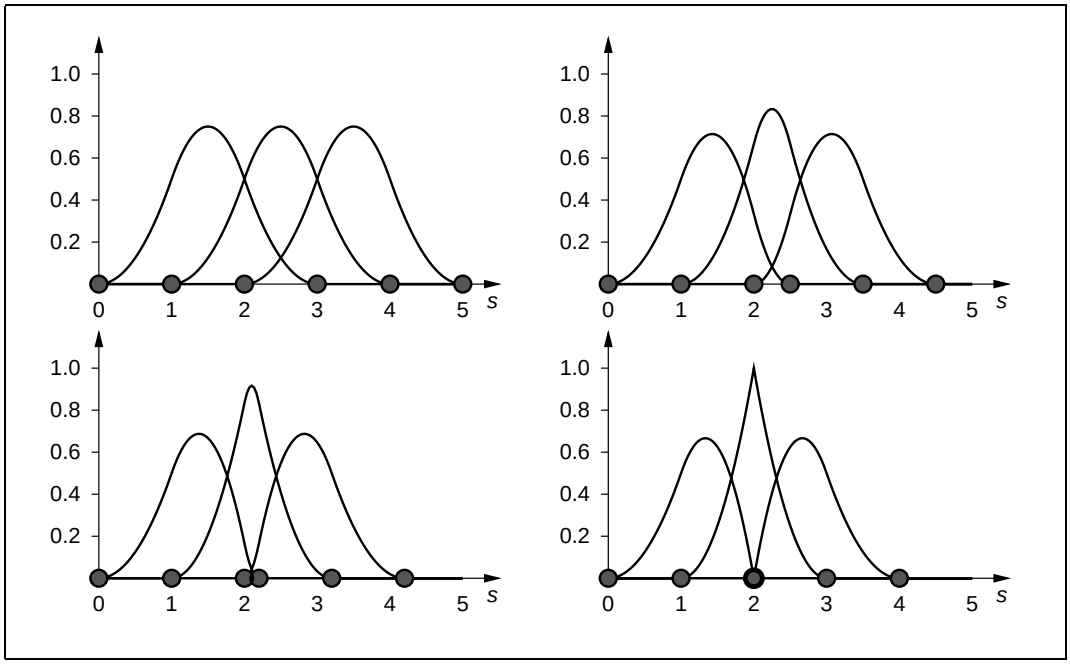


Figure 3.7: Forming multiple knots. *A double knot is introduced into a quadratic B-spline basis at $s = 2$. The resulting basis functions are the limit reached as the knot initially at $s = 3$ approaches $s = 2$.*

which is precisely the "root-mean-square" value[2] of $x(s)$ over the range $0 \leq s \leq L$. The functional norm is especially useful for measuring the difference between two functions $x_1(s), x_2(s)$ as $\|x_1 - x_2\|$, for instance when it is necessary to measure how closely a function x_1 is approximated by another function x_2.

The norm has a corresponding "inner product," denoted $< \cdot, \cdot >$, which is bilinear and is applied to a pair of functions x, y as $< x, y >$. The relationship between an inner product and a norm is that $< x, x >= \|x\|^2$, so in the L_2 case the inner product between two functions works out to be:

$$< x, y >= \frac{1}{L} \int_0^L x(s)y(s)\, ds. \tag{3.8}$$

[2]Conventionally the $\frac{1}{L}$ scaling factor in the definition of the norm would be omitted; we include it so that $\|x\|$ is truly a root-mean-square measure.

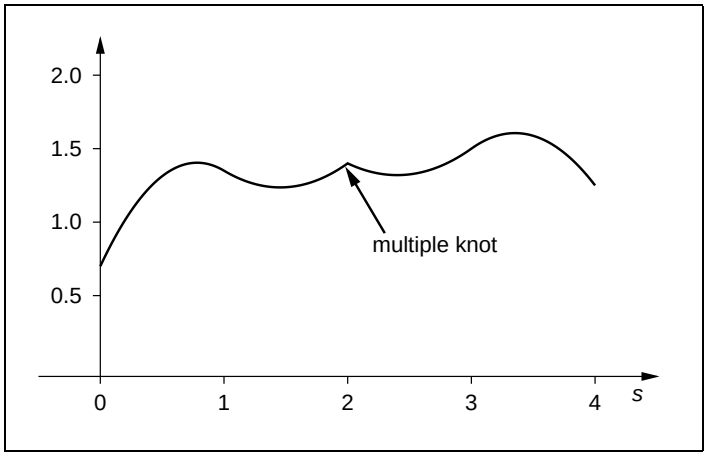


Figure 3.8: Reduced continuity. *Using a basis with a double knot as in figure 3.7, in which a double knot has been introduced at $s = 2$, constructed functions show a gradient discontinuity at $s = 2$ — compare with figure 3.6.*

The inner product will be used later to express function approximations concisely.

Since we are representing functions compactly as vectors $\mathbf{Q}^x$ of spline weights, it is natural to express norms and inner products in terms of these vectors, that is, to define $\|\cdot\|$ for weight vectors such that

$$\|\mathbf{Q}^x\| = \|x\|.$$

Now from (3.2)

$$\|x\|^2 = (\mathbf{Q}^x)^T \frac{1}{L} \left(\int_0^L \mathbf{B}(s)\mathbf{B}(s)^T \, ds \right) \mathbf{Q}^x$$

so the norm must be defined as:

$$\|\mathbf{Q}^x\| = \sqrt{(\mathbf{Q}^x)^T \mathcal{B} \mathbf{Q}^x} \tag{3.9}$$

where

$$\mathcal{B} = \frac{1}{L} \int_0^L \mathbf{B}(s)\mathbf{B}(s)^T \, ds \tag{3.10}$$

defines the *metric matrix* for a given class of B-spline function, and the inner product is

$$< \mathbf{Q}_1^x, \mathbf{Q}_2^x > \equiv (\mathbf{Q}_1^x)^T \mathcal{B} \mathbf{Q}_2^x. \tag{3.11}$$

The $\mathcal{B}$-matrices are sparse. This reflects the fact that each weight Q_n^x affects the function $x(s)$ only over a short sub-interval — the "support" of the corresponding basis function B_n — and this lends efficiency to least-squares approximation algorithms. In the periodic, quadratic case the $\mathcal{B}$-matrices are sparse circulants of order 5, and in general they have order $2d-1$ (the number of non-zero elements in each row). For a given polynomial order d, therefore, the sparsity is most significant when the number N_B of basis functions is large. For periodic quadratic splines with $N_B = 8$ the $\mathcal{B}$-matrix is:

$$\mathcal{B} = \frac{1}{8}\begin{pmatrix} 0.55 & 0.217 & 0.008 & 0.0 & 0.0 & 0.0 & 0.008 & 0.217 \\ 0.217 & 0.55 & 0.217 & 0.008 & 0.0 & 0.0 & 0.0 & 0.008 \\ 0.008 & 0.217 & 0.55 & 0.217 & 0.008 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.008 & 0.217 & 0.55 & 0.217 & 0.008 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.008 & 0.217 & 0.55 & 0.217 & 0.008 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.008 & 0.217 & 0.55 & 0.217 & 0.008 \\ 0.008 & 0.0 & 0.0 & 0.0 & 0.008 & 0.217 & 0.55 & 0.217 \\ 0.217 & 0.008 & 0.0 & 0.0 & 0.0 & 0.008 & 0.217 & 0.55 \end{pmatrix} \tag{3.12}$$

— note the circulant structure (repeating rows), characteristic of the periodic case. For a non-periodic quadratic function, $\mathcal{B}$ is still sparse, no longer a circulant, but now pentadiagonal, as shown here for the case $N_B = 6$ ($L = 4$):

$$\mathcal{B} = \frac{1}{4}\begin{pmatrix} 0.2 & 0.117 & 0.017 & 0.0 & 0.0 & 0.0 \\ 0.117 & 0.333 & 0.208 & 0.008 & 0.0 & 0.0 \\ 0.017 & 0.208 & 0.55 & 0.217 & 0.008 & 0.0 \\ 0.0 & 0.008 & 0.217 & 0.55 & 0.208 & 0.017 \\ 0.0 & 0.0 & 0.008 & 0.208 & 0.333 & 0.117 \\ 0.0 & 0.0 & 0.0 & 0.017 & 0.117 & 0.2 \end{pmatrix} \tag{3.13}$$

and would be heptadiagonal for cubic splines.

Armed with the inner product, some approximation problems become straightforward. The simplest tutorial example is the problem of approximating a spline function represented as $\mathbf{Q}^x$ in terms of two other spline functions $\mathbf{Q}_1^x, \mathbf{Q}_2^x$. The least-squares approximation can be expressed, using inner products, as

$$\hat{\mathbf{Q}}^x = \begin{pmatrix} \mathbf{Q}_1^x & \mathbf{Q}_2^x \end{pmatrix} \begin{pmatrix} \langle \mathbf{Q}_1^x, \mathbf{Q}_1^x \rangle & \langle \mathbf{Q}_1^x, \mathbf{Q}_2^x \rangle \\ \langle \mathbf{Q}_2^x, \mathbf{Q}_1^x \rangle & \langle \mathbf{Q}_2^x, \mathbf{Q}_2^x \rangle \end{pmatrix}^{-1} \begin{pmatrix} \langle \mathbf{Q}_1^x, \mathbf{Q}^x \rangle \\ \langle \mathbf{Q}_2^x, \mathbf{Q}^x \rangle \end{pmatrix} \tag{3.14}$$

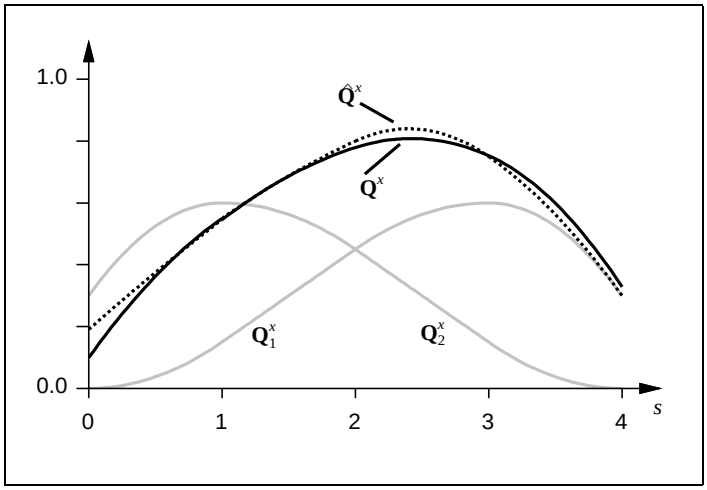


Figure 3.9: Spline approximation: *$\hat{\mathbf{Q}}^x$ is the least-squares approximation of a spline $\mathbf{Q}^x$ in terms of two other splines $\mathbf{Q}_1^x, \mathbf{Q}_2^x$. The solution can be concisely expressed in terms of inner products — see text.*

— see figure 3.9. A development of this method appears later when, in the interests of economy and of stability over time, it is required to express spline curves in terms of a relatively small number of parameters. The reduction of the parameter set is expressed as a projection operation rather like the approximation $\mathbf{Q}^x \rightarrow \hat{\mathbf{Q}}^x$ above, but for curves instead of functions.

Another important type of problem is to approximate some function $f(s)$, not necessarily a spline, as a spline function $x(s)$, represented, as usual, by weights $\mathbf{Q}^x$. Again, the solution can be derived neatly using inner products to give:

$$\mathbf{Q}^x = \mathcal{B}^{-1} \frac{1}{L} \int \mathbf{B}(s) f(s)\, ds \tag{3.15}$$

and an example is shown in figure 3.10. Functional approximation of this kind can be developed to construct approximations of curves in image data, and this is close to what will be required for active contour algorithms. Of course it is not possible to evaluate integrals over data exactly, so in practice data must be sampled. The simplest approximation of a sampled function as a spline is:

$$\mathbf{Q}^x = \mathcal{B}^{-1} \frac{1}{N} \sum_{n=1}^{N} \mathbf{B}(s_n) f(s_n).$$

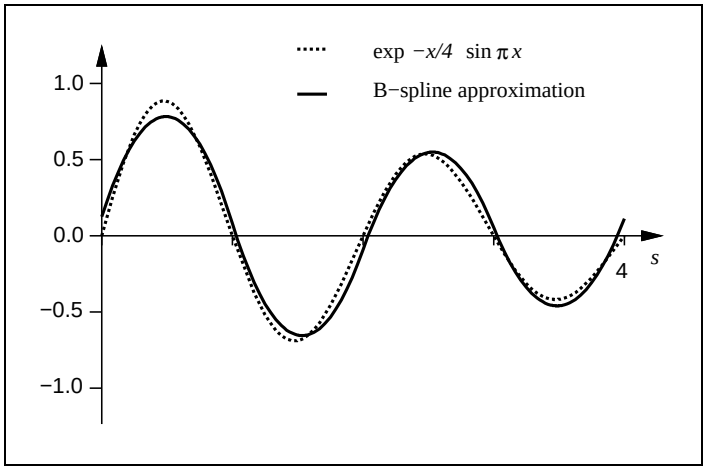


Figure 3.10: Spline approximation: *least-squares spline approximation to a damped sine wave ($L = 4, N_B = 6$).*

The application to image curves is discussed in chapter 6.

3.5 B-spline parametric curves

Spline functions were introduced to serve as a tool for constructing curves in the plane, which they do in the following manner. Parametric spline curves

$$\mathbf{r}(s) = (x(s), y(s))$$

have coordinates $x(s),\ y(s)$ each of which is a spline function of the curve parameter s. First it is necessary to choose an appropriate interval $0 \leq s \leq L$ covering L spans and an appropriate basis $B_0, B_1, \ldots, B_{N_B-1}$ of N_B B-spline functions or basis functions. If the interval $[0, L]$ is taken to be periodic the resulting parametric curve will be closed. Alternatively, an open curve requires a B-spline basis over a finite interval as in figure 3.6. For each basis function B_n a *control point* $\mathbf{q}_n = (q_n^x, q_n^y)^T$ must now be defined and the curve is a weighted vector sum of control points

$$\mathbf{r}(s) = \sum_{n=0}^{N_B-1} B_n(s)\mathbf{q}_n \quad \text{for} \quad 0 \leq s \leq L, \tag{3.16}$$

a smooth curve that follows approximately the "control polygon" defined by linking control points by lines (figure 3.11). The component functions of $\mathbf{r}(s)$ do, of course, turn out to be spline functions, for instance:

$$x(s) = \sum_{n=0}^{N_B-1} B_n(s) q_n^x \quad \text{for} \quad 0 \leq s \leq L, \tag{3.17}$$

— a weighted sum of basis functions with weights q_n^x. The example curve in figure 3.11 uses the basis functions B_n for regular, periodic, quadratic splines that were defined earlier in (3.6) on page 45 and, as before, $N_B = L$ so that the number of control points of the curve is equal to the number of its spans.

3.6 Curves with vertices

It is often necessary to introduce a vertex or hinge at a certain point along a parametric curve, to fit around sharp corners on an object outline. One straightforward way of doing this is to allow two or more consecutive control points to coincide to form a "multiple control point." When n consecutive control points coincide, the order of continuity of the curve is reduced by $n - 1$. A quadratic spline, for instance, has continuous first derivative but discontinuous second derivative when all control points are distinct. A hinge (discontinuous first derivative) is formed therefore when 2 consecutive control points coincide, as in figure 3.12. Unfortunately, introducing a hinge in this way generates spurious linearity constraints (see figure). What is more, the parameterisation of the curve behaves badly in the vicinity of the hinge in the sense that $\mathbf{r}'(s) = \mathbf{0}$ so that the parameter s is changing infinitely fast as the curve passes through the hinge. This would have the effect in the curve-fitting algorithms to be described in chapter 6 of giving undue weight to the region of the curve around the hinge. A good alternative is to use multiple knots — not quite as simple but having good geometric behaviour. The formation of multiple knots in a B-spline basis was explained earlier, in section 3.3. Parametric curves defined with the new basis inherit its reduced continuity. For example, a hinge can be formed in a quadratic, parametric curve by introducing a double knot into the underlying quadratic B-spline basis, as in figure 3.13. Alternatively a triple knot introduces a break in a quadratic B-spline curve.

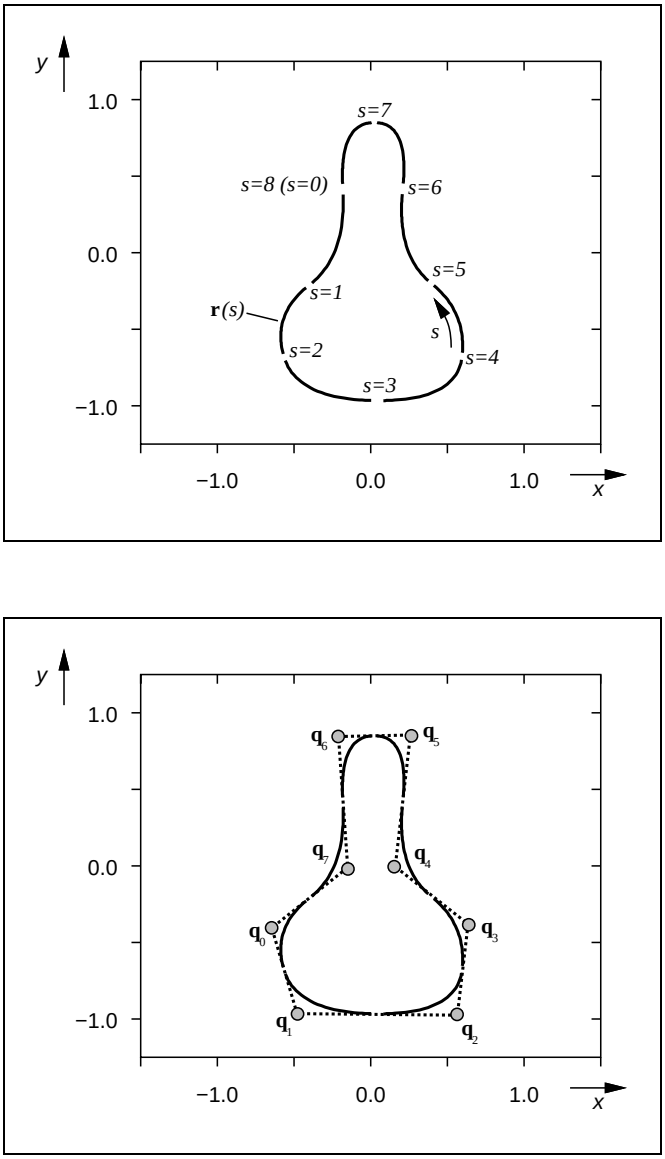


Figure 3.11: *A quadratic ($d = 3$), parametric spline curve $\mathbf{r}(s)$ is shown (top) that is regular, and closed. It has 8 knots, so $L = 8$, and $0 \leq s \leq 8$, and s is treated as periodic. The curve is a smooth approximation to its "control polygon" (bottom) formed from a sequence of control points $\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_7$. Note that $N_B = L$ for regular closed spline curves — here $N_B = L = 8$.*

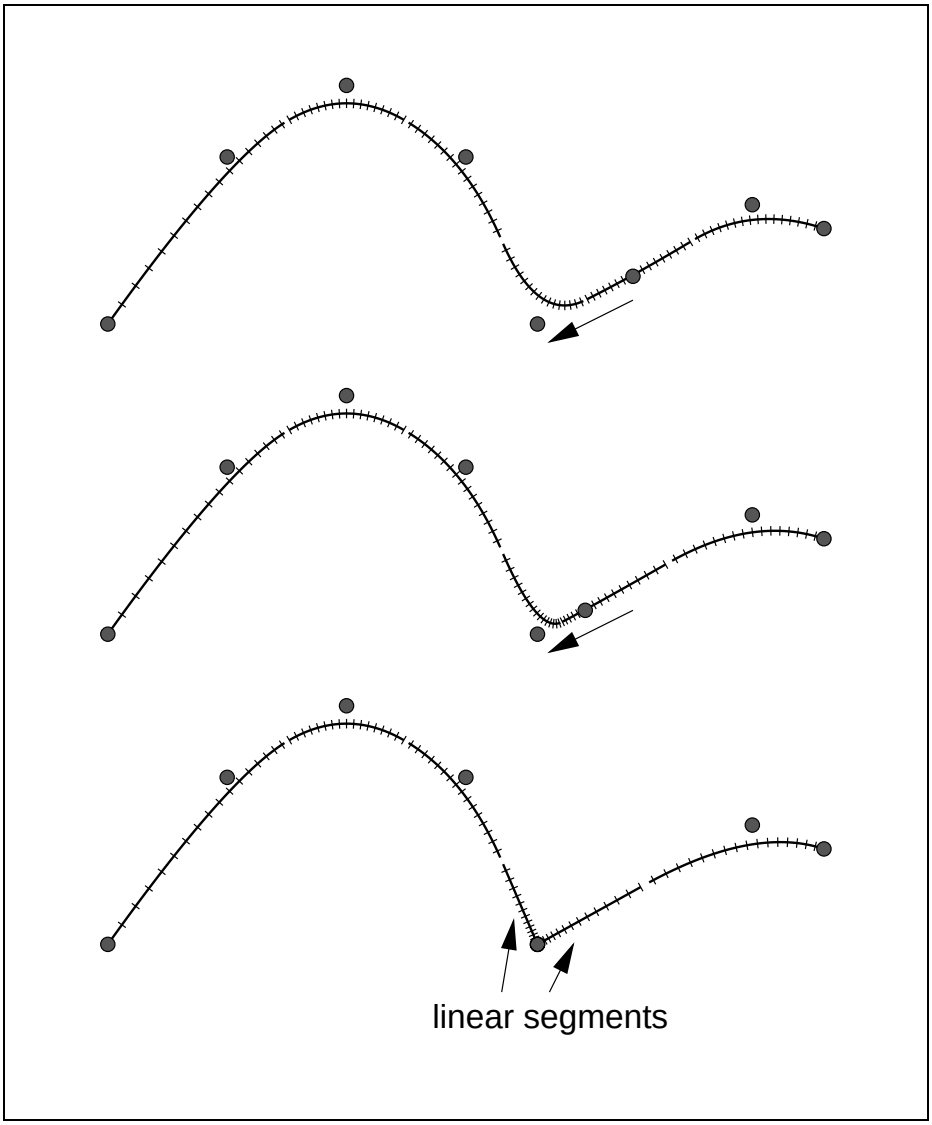
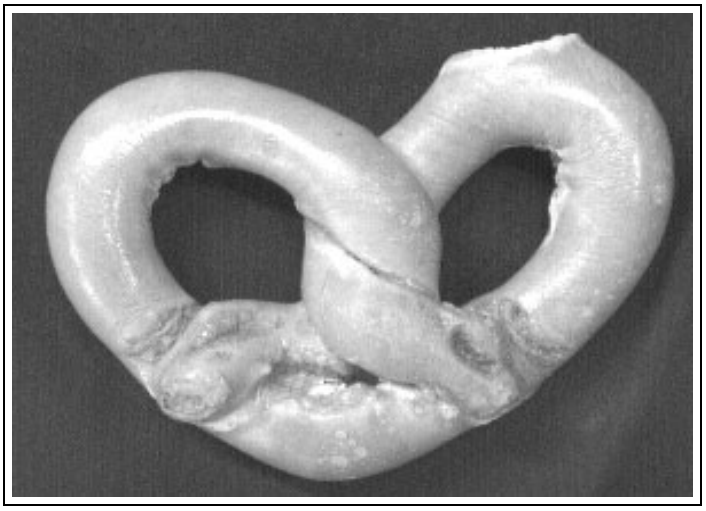


Figure 3.12: Multiple control points. *On a quadratic, parametric spline curve, the curve tangent becomes discontinuous when two control points coincide, forming a "hinge." However, forming a hinge in this fashion turns out to be unsatisfactory, partly because of the segments on either side of the vertex which are constrained to be linear, and partly because of problems with uneven parameterisation — see text.*



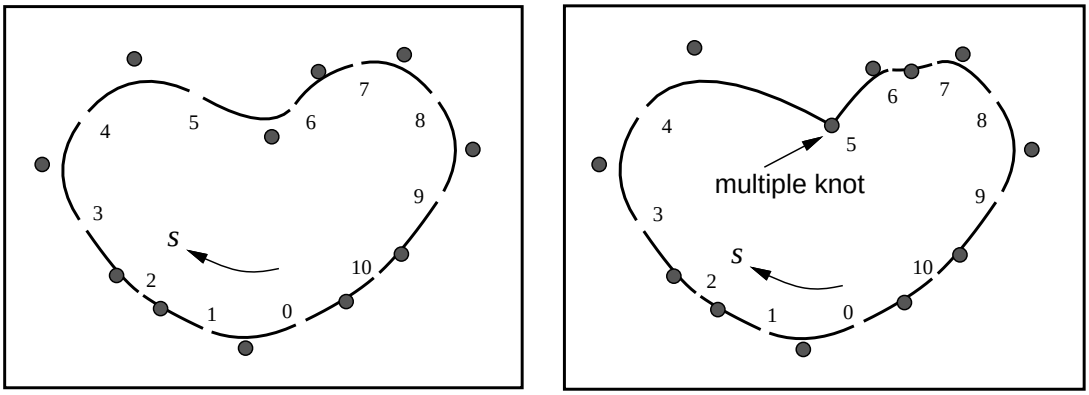


Figure 3.13: *The outline of the pretzel at the top is heart-shaped with a vertex at its apex. A regular, parametric spline curve (left) does not follow the outline as closely as one with a multiple knot (right). In the case of a quadratic spline curve, a tangent discontinuity forms at the control point corresponding to the double knot.*

3.7 Control vector

Dealing with control points explicitly is cumbersome so, as a first step towards a more compact notation, let us first define a space $\mathcal{S}_Q$ of *control vectors* $\mathbf{Q}$ consisting of

control point coordinates, first all the x-coordinates, then all the y-coordinates:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}^x \\ \mathbf{Q}^y \end{pmatrix} \quad \text{where} \quad \mathbf{Q}^x = \begin{pmatrix} q_0^x \\ \cdots \\ \cdots \\ q_{N_B-1}^x \end{pmatrix} \tag{3.18}$$

and similarly for $\mathbf{Q}^y$. Then the coordinate functions can be written as

$$x(s) = \mathbf{B}(s)^T \mathbf{Q}^x,$$

where $\mathbf{B}(s)$ is a vector of B-spline basis functions as defined earlier, and similarly for $y(s)$, so that

$$\mathbf{r}(s) = U(s)\mathbf{Q} \quad \text{for} \quad 0 \le s \le L \tag{3.19}$$

where

$$U(s) = I_2 \otimes \mathbf{B}(s)^T = \begin{pmatrix} \mathbf{B}(s)^T & \mathbf{0} \\ \mathbf{0} & \mathbf{B}(s)^T \end{pmatrix}, \tag{3.20}$$

a matrix of size $2 \times 2N_Q$. (Note that $\otimes$ denotes the "Kronecker product" of two matrices, a notation that will be used again. See appendix A for details. The matrix I_m denotes an $m \times m$ identity.)

3.8 Norm for curves

Now that we have set up a representation of curves as parametric splines, the next step is therefore to extend the norm and inner product to curves, for use in curve approximation. We can define a norm $\|\cdot\|$ for B-spline curves which is induced by the Euclidean distance measure in the image plane:

$$\|\mathbf{Q}\|^2 = \frac{1}{L} \int_{s=0}^{L} |\mathbf{r}(s)|^2 \, ds \tag{3.21}$$

or equivalently, from (3.19) and (3.10),

$$\|\mathbf{Q}\|^2 = \mathbf{Q}^T \mathcal{U} \mathbf{Q}, \tag{3.22}$$

where the metric matrix for curves $\mathcal{U}$ is defined in terms of the metric matrix $\mathcal{B}$ for B-spline functions:

$$\mathcal{U} \equiv \frac{1}{L} \int_0^L U(s)^T U(s) \, ds \tag{3.23}$$

or, equivalently,

$$\mathcal{U} = I_2 \otimes \mathcal{B} = \begin{pmatrix} \mathcal{B} & 0 \\ 0 & \mathcal{B} \end{pmatrix}.$$

Of course, the norm also implies an inner product for curves:

$$\langle \mathbf{Q}_1, \mathbf{Q}_2 \rangle = \mathbf{Q}_1^T \mathcal{U} \mathbf{Q}_2.$$

The curve norm is particularly meaningful when used as a means of comparison between two curves, using the distance $\|\mathbf{Q}_1 - \mathbf{Q}_2\|$. This is the basis for approximation of visual data by curves, and is illustrated in figure 3.14.

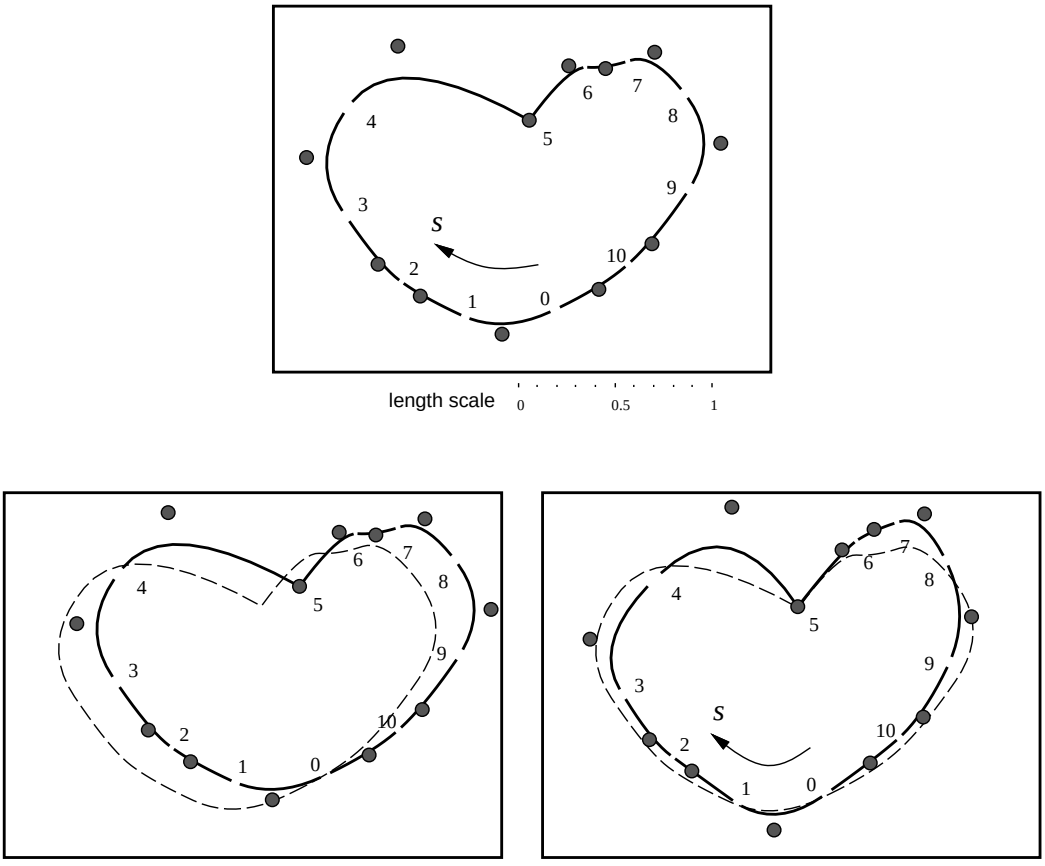
There are potentially simpler norms than the one above, the obvious candidate being the Euclidean norm $|\mathbf{Q}|^2 \equiv \mathbf{Q}^T\mathbf{Q}$ of the control vector. Compared with the L_2 norm $\|\cdot\|$ defined above, the Euclidean norm is simpler to compute because the banded matrix $\mathcal{U}$ is replaced by the identity matrix. However, attractive as this short cut may be, the simpler norm does not work satisfactorily. This is made clear by the counter-example of figure 3.15 in which decreasing the displacement between two curves produces an increase in the Euclidean norm. This example makes it clear that the Euclidean norm is not suitable for ranking the closeness of curve approximations.

Invariance to re-parameterisation

It is important to note that the curve norm, as a measure of the difference between a pair of curves, does not allow for possible re-parameterisation of one of the curves. For example, a curve $\mathbf{r}(s),\ 0 \le s \le 1$ could be reparameterised to give a new curve $\mathbf{r}^*(s) = \mathbf{r}(1-s),\ 0 \le s \le 1$. Geometrically, the two curves are identical; the difference between them is simply a reversal of parameterisation. We would like ideally to measure shape differences in a way that is invariant to re-parameterisation, so that the vector difference function $\mathbf{r}^* - \mathbf{r}$ would ideally have a norm of zero. However, the L_2 norm will not behave in this way:

$$\|\mathbf{r}^*(s) - \mathbf{r}(s)\|^2 = \frac{1}{L}\int |\mathbf{r}(1-s) - \mathbf{r}(s)|^2\, ds \neq 0,$$

in general. As a result, any curve-fitting algorithm that uses the norm will be disrupted if the parameterisation of the target curve fails to match that of the template, and this is illustrated in figure 3.16.



average displacement: norm= 0.22 average displacement: norm= 0.08

Figure 3.14: Measuring average displacement between curves. *The pretzel curve (top) is compared, using the norm as a measure, with two modified curves: a translation (left) and a modest distortion (right). The values given for the norm in each case represent average (root-mean-square) displacements for the curves, in length units as shown on the scale.*

A general solution to the problem of parameterisation invariance would require a search over possible parameterisations. The proximity of a curve $\mathbf{r}(s)$ to a second curve $\mathbf{r}^*(s)$ could be evaluated as

$$\min_g \|\mathbf{r}(s) - \mathbf{r}^*(g(s))\|$$

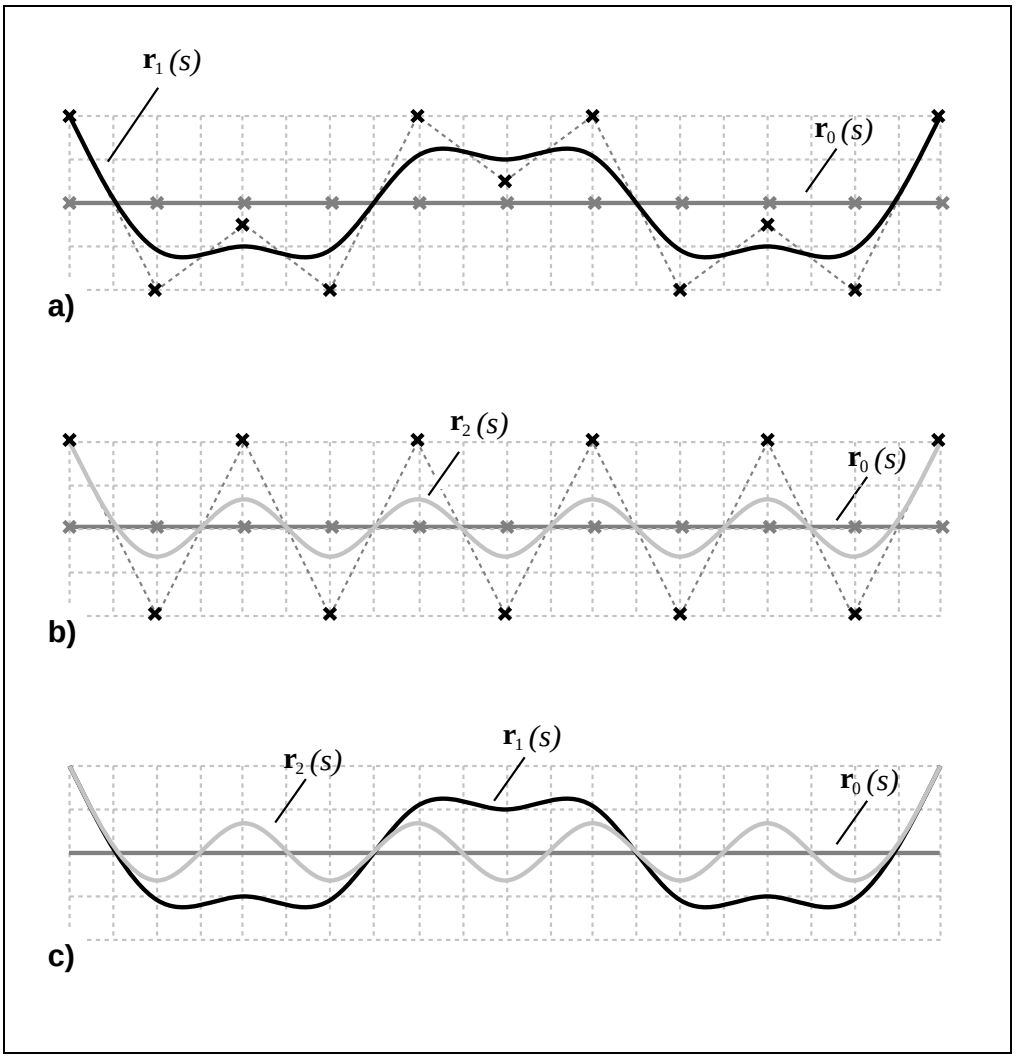
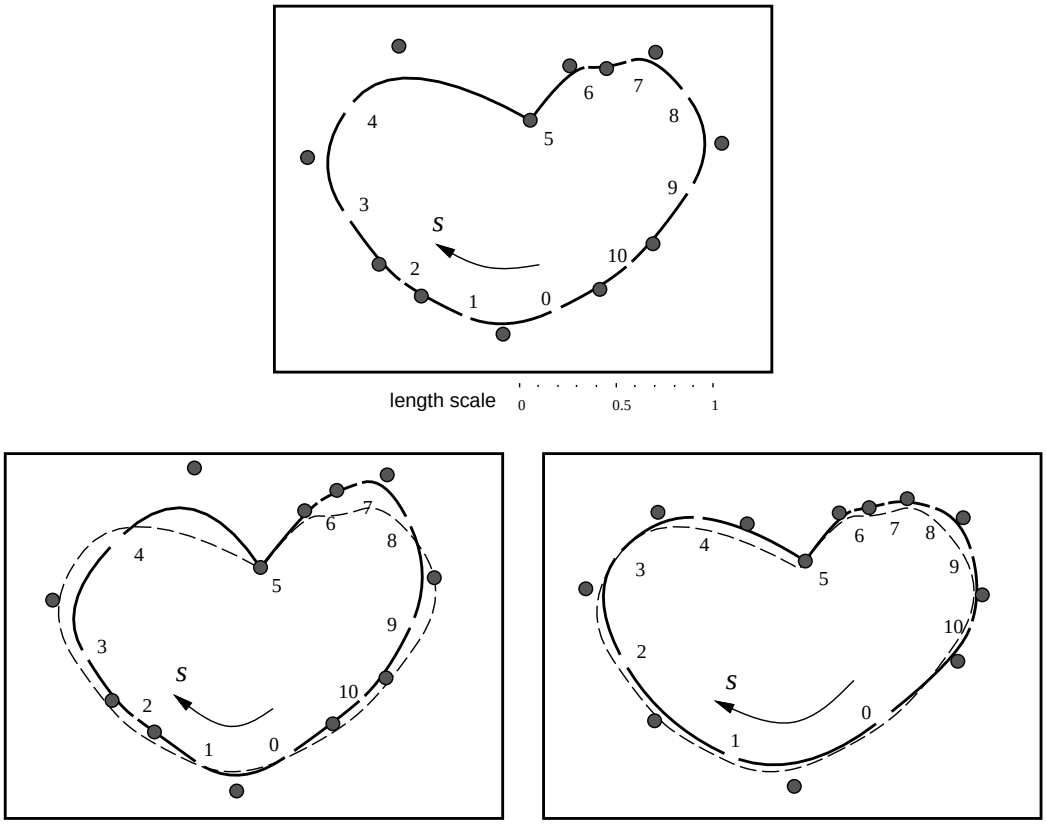


Figure 3.15: Euclidean distance between control vectors is a poor measure of curve displacement. *In a) and b) two cubic spline curves* $\mathbf{r}_1, \mathbf{r}_2$ *are shown together with their control points (crosses) and control polygons (dotted). Each is to be compared with a standard curve* $\mathbf{r}_0$*. It is clear that, pointwise,* $\mathbf{r}_2$ *is closer to* $\mathbf{r}_0$ *than* $\mathbf{r}_1$ *is. Any reasonable curve metric should reflect that and, sure enough, in the* L_2 *norm,* $\|\mathbf{r}_2 - \mathbf{r}_0\| < \|\mathbf{r}_1 - \mathbf{r}_0\|$*. However it is clear from the shape of the control polygons that, in Euclidean distance,* $\mathbf{r}_2$ *is actually* further *from* $\mathbf{r}_0$ *than* $\mathbf{r}_1$ *is. Euclidean distance between control vectors is therefore ruled out as a curve metric.*



average displacement: norm= 0.08 average displacement: norm= 0.34

Figure 3.16: Norm-difference is sensitive to re-parameterisation. *A substantial change in parameterisation (right), though it gives rise to little change in shape, nonetheless registers a larger average displacement than the more distorted shape (left).*

where the minimum is explored over some space of re-parameterisation functions g, using an appropriate optimisation procedure. Suitably powerful optimisation procedures have been developed (see the bibliographic notes at the end of this chapter) but they are computationally costly. A more economical approach is to use a distance measure $d(\mathbf{r}, \mathbf{r}^*)$ that is invariant to *minor* re-parameterisation of the curve $\mathbf{r}$. This is developed later, in chapter 6. The distance measure is based on the *normal* displace-

ment of $\mathbf{r}$ relative to $\mathbf{r}^*$, that is, omitting any tangential component of displacement — the sliding of one curve along the other. It will be shown that such a distance measure is approximately invariant and it remains to achieve a rough alignment of two curves sufficient for the invariant distance approximation to be effective. One way of doing this uses moments, as described below.

3.9 Areas and moments

Applications in computer vision often require the computation of gross properties of a curve. Curve moments — area, centroid, and higher moments are useful for computing approximate curve position and orientation, and to obtain gross shape information sufficient for some coarse discrimination between objects.

Generally, moments have two roles in active contours. The first is initialisation, in which a spline template is positioned sufficiently close to the tracked object to "lock" onto it. At this stage moments may also be used for coarse shape discrimination to confirm the identity of the object being tracked. The second role is in interpreting the position and orientation of a tracked object, for example the 3D visual mouse in figure 1.16 on page 20. For example, if hand motion is restricted to 3D translation and rotation in the image plane, those four parameters can be recovered from the zeroth moment (area) the first moment (centroid) and the second moment (inertia).

Centroid

A conventional definition for the centroid of a curve is

$$\overline{\mathbf{r}} = \frac{1}{L}\int_0^L \mathbf{r}(s)\,ds$$

which can be computed straightforwardly from the spline-vector $\mathbf{Q}$ using inner products:

$$\overline{\mathbf{r}} = \begin{pmatrix} \langle \mathbf{1}_x, \mathbf{Q} \rangle \\ \\ \langle \mathbf{1}_y, \mathbf{Q} \rangle \end{pmatrix} \quad \text{where} \quad \mathbf{1}_x = \begin{pmatrix} \mathbf{1} \\ \\ \mathbf{0} \end{pmatrix} \quad \text{and} \quad \mathbf{1}_y = \begin{pmatrix} \mathbf{0} \\ \\ \mathbf{1} \end{pmatrix}. \tag{3.24}$$

This simple definition of centroid is computationally convenient but has the drawback that it is not invariant to re-parameterisation of the curve, as figure 3.17 shows. This is because s is not generally true arclength; it is simply a convenient spline parameter.

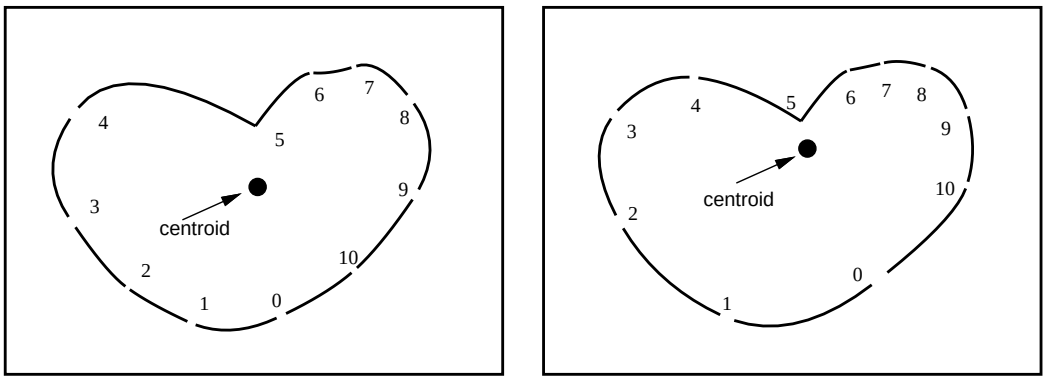


Figure 3.17: Simple centroid is not invariant. *The centroid of the pretzel curve (left), calculated as in (3.24), shifts substantially when the curve is reparameterised (right), even though there is almost no shape change.*

The length of an infinitesimal segment of curve is not ds but $|\mathbf{r}'(s)|ds$, so that an invariant centroid of the curve would be

$$\overline{\mathbf{r}} = \frac{\int_0^L \mathbf{r}(s)|\mathbf{r}'(s)|\,ds}{\int_0^L |\mathbf{r}'(s)|\,ds}.$$

The square root implicit in $|\mathbf{r}'(s)|$ means that this invariant centroid cannot be computed directly in terms of the spline-vector $\mathbf{Q}$. An alternative invariant centroid for closed curves is the centroid of area described below, which can be computed directly, as a cubic function of $\mathbf{Q}$. For many purposes, non-invariant moments are adequate. For example, in a hand-tracking application the parameterisation of the tracked curve is, typically, strongly stabilised by a template. The parameterisation of the tracked curve does not, in practice, deviate much from the standard parameterisation inherited from the template. In that case the 2D translational motion of the tracked object can be recovered satisfactorily from the non-invariant centroid.

Invariant moments

Suppose an active contour is to be initialised from an area of pixels detected by image processing based on brightness, colour or motion. The vector $\mathbf{Q}$ for the initial configuration of the tracked curve is set by manipulating it to bring the moments of the area enclosed by the curve into close agreement with the moments of the active area.

The simplest available parameterisation-invariant measure for a closed curve is the area

$$\int \left|\mathbf{r}(s), \mathbf{r}'(s)\right| \, ds$$

where $|\mathbf{x}, \mathbf{y}|$ denotes the determinant of the matrix whose columns are $\mathbf{x}, \mathbf{y}$. This is neatly expressible as a quadratic form in $\mathbf{Q}$:

$$A(\mathbf{Q}) = \mathbf{Q}^T \mathcal{A} \mathbf{Q}, \tag{3.25}$$

reminiscent of the norm in (3.22) but in place of the symmetric matrix $\mathcal{U}$ we have:

$$\mathcal{A} \equiv \begin{pmatrix} \mathcal{B}' & 0 \\ 0 & -\mathcal{B}' \end{pmatrix} \quad \text{where} \quad \mathcal{B}' = \int_0^L \mathbf{B}(s)\mathbf{B}'^T(s)\,ds. \tag{3.26}$$

(Details of efficient computation of $\mathcal{A}$ are given in appendix A.2.) As with the matrix $\mathcal{U}$, $\mathcal{A}$ is $2N_Q \times 2N_Q$ where

$$N_Q = 2N_B, \tag{3.27}$$

the dimension of the spline space. The matrix $\mathcal{A}$ is sparse which makes the computation of the area quadratic form relatively efficient. One direct application for curve area computation is in visual navigation, and a picturesque example is given in figure 3.18.

The centroid $\overline{\mathbf{r}}$ of the area enclosed by a closed B-spline curve, which is invariant to curve re-parameterisation, is given by

$$\overline{\mathbf{r}} = \frac{1}{A(\mathbf{Q})} \int_0^L \left|\mathbf{r}(s), \mathbf{r}'(s)\right| \mathbf{r}(s)\, ds. \tag{3.28}$$

In principle this is a useful measure for positioning, but is moderately costly — $O(N_Q^3)$ — to compute exactly. This improves to $O(N_X^3)$ if curves are restricted to a "shape-space" of reduced dimension N_X, and this is discussed in the next chapter. Similarly, the second moment

$$\mathcal{I} = \frac{1}{A(\mathbf{Q})} \int_0^L \left|\mathbf{r}(s), \mathbf{r}'(s)\right| \mathbf{r}(s)\mathbf{r}^T(s)\, ds \tag{3.29}$$

is invariant and useful in principle for orienting a shape, but the computational cost is $O(N_Q^4)$, again reduced if a shape-space is used.

Bibliographic notes

This chapter has outlined a framework for representing curves in the image plane. It has been common both in robotics and in computer vision to represent curves algebraically as $f(x, y) = 0$ where f is a polynomial (Faverjon and Ponce, 1991; Petitjean et al., 1992; Forsyth et al., 1990). Although such representations are often attractive mathematically, for the purpose of constructing proofs, they are cumbersome from the computational point of view. Practical systems using curve approximation are better founded on B-splines. Tutorials on splines can be found in graphics books such as (Foley et al., 1990) or in books on computer-aided design such as (Faux and Pratt, 1979). Some essential details and algorithms are given also in the appendix of this

Figure 3.18: *While a travelling video camera approaches a car, a B-spline curve is locked onto the outline of the windscreen in successive video frames. The computed windscreen area $a(t)$, increasing over time, can be used to estimate time-to-collision as $a(t)/\dot{a}(t)$.* (Figure reproduced from (Cipolla and Blake, 1992a).)

book. A more complete book on splines, oriented toward computer graphics is (Bartels et al., 1987) and a mathematical source on spline functions (but not curves) is (de Boor, 1978).

Splines, common in computer graphics, have also been used in computer vision for some years, for shape-warping (Bookstein, 1989), representing corners and edges in static scenes (Medioni and Yasumoto, 1986; Arbogast and Mohr, 1990) and for shape approximation (Menet et al., 1990) and tracking (Cipolla and Blake, 1990).

Shape approximation using spline curves is an application of the "normal equations" for approximation problems (Press et al., 1988). Equivalently it uses a "pseudo-inverse" (Barnett, 1990) of which the B-spline metric matrix $\mathcal{B}$ is a component. Function norms are a standard mathematical tool for functional approximation (Kreysig, 1988) and in signal processing (Papoulis, 1991) and image processing (Gonzales and Wintz, 1987) for least-squares restoration problems.

Measures of curve difference that are more economical to compute than the L_2 norm can be made by replacing the metric matrix with the identity to give the Euclidean distance between control vectors, as done for polygons in (Cootes and Taylor, 1992). However, such measures do have some undesirable properties, as explained earlier. Curve matching using norms is not invariant to re-parameterisation; matching algorithms do exist that deal with re-parameterisation, for example ones developed for stereoscopic image matching (Ohta and Kanade, 1985; Witkin et al., 1986) but they are computationally expensive, too much so for use in real-time tracking systems. This is discussed again in chapter 6.

Chapter 4

Shape-space models

In practice, it is very desirable to distinguish between the *spline*-vector $\mathbf{Q} \in \mathcal{S}_Q$ that describes the basic shape of an object and the *shape*-vector which we denote $\mathbf{X} \in \mathcal{S}$, where $\mathcal{S}$ is a *shape-space*. Whereas $\mathcal{S}_Q$ is a vector space of B-splines and has dimension $N_Q = 2N_B$, the shape-space $\mathcal{S}_X$ is constructed from an underlying vector space of dimension N_X which is typically considerably smaller than N_Q. The shape-space is a linear parameterisation of the set of allowed deformations of a base curve. The necessity for the distinction is made clear in figure 4.1. To obtain a spline that does justice to the geometric complexity of the face shape, thirteen control points have been used. However, if all of the resulting 26 degrees of freedom of the spline-vector $\mathbf{Q}$ are manipulated arbitrarily, many uninteresting shapes are generated that are not at all reminiscent of faces. Restricting the displacements of control points to a lower-dimensional shape-space is more meaningful if it preserves the face-like quality of the shape. Conversely, using the unconstrained control-vector $\mathbf{Q}$ leads to unstable active contours and this was illustrated in figure 2.4 on page 31.

The requirement that a shape-space be a *linear* parameterisation is made for the sake of computational simplicity. The curve-fitting and tracking procedures described in the book are substantially simplified by linearity and in many cases exact algorithms are available only for linear parameterisations. Linearly parameterised, image-based models work well for rigid objects however, and for simpler non-rigid ones. Linearity can certainly be a limitation when the allowed motions of an object become more complex, for example a three-dimensional object with articulated parts. Articulation can in fact be dealt with in linearly parameterised, image-based models but only at the cost of relaxing certain geometric constraints. This is explored further in the

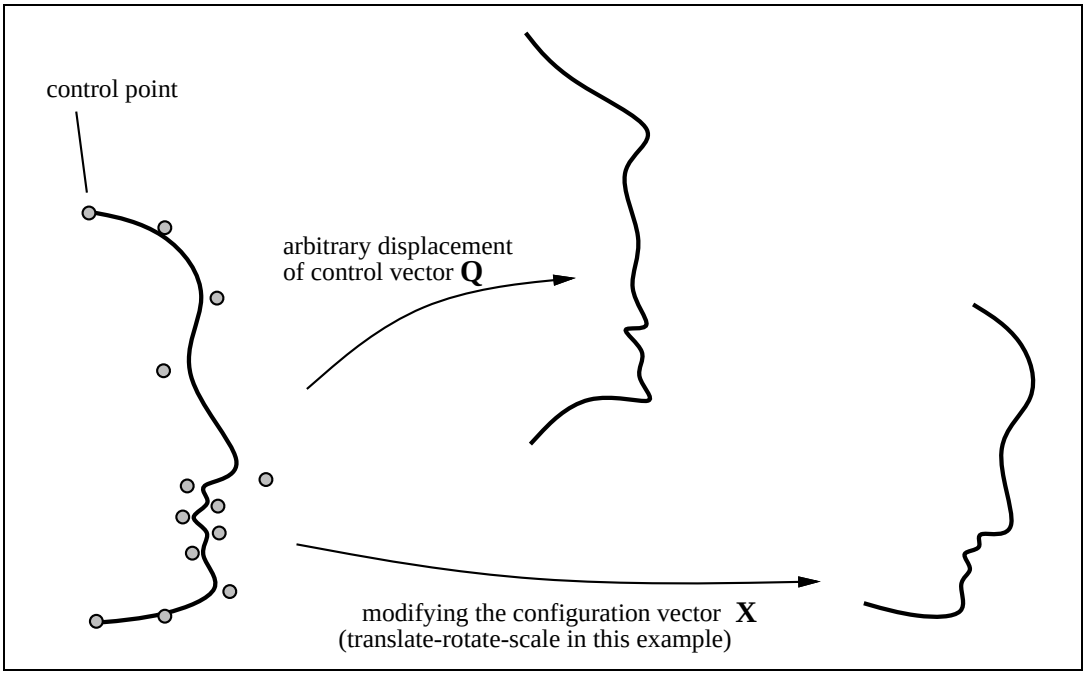


Figure 4.1: Configuration vector. *Arbitrary manipulation of the spline-vector* $\mathbf{Q}$ *of a spline curve is likely to be too general to be practically interesting. In this example a face curve ceases to look face-like. What is far more interesting is a restricted class* $\mathcal{S}$ *of transformations, parameterised by a relatively low-dimensional configuration vector* $\mathbf{X}$*. In this case* $\mathbf{X}$ *is a Euclidean similarity transformation which does retain the face-like character.*

discussion of shape-spaces below. A more detailed discussion of the trade-off between image-based models and three-dimensional models is given in appendix C.

4.1 Representing transformations in shape-space

Rigid motion

A simple example of a shape-space is the space of Euclidean similarities of a template curve $\mathbf{r}_0(s)$. This is a space of dimension 4 corresponding exactly to the variation of an image curve as a camera with a zoom lens looks directly down on a planar object that is free to move on a table top. The effect on the curve is that it moves rigidly in

the image plane and may also magnify or diminish in size, but its shape is preserved, as in figure 4.2. Alternatively it could be that the camera is able to translate in three

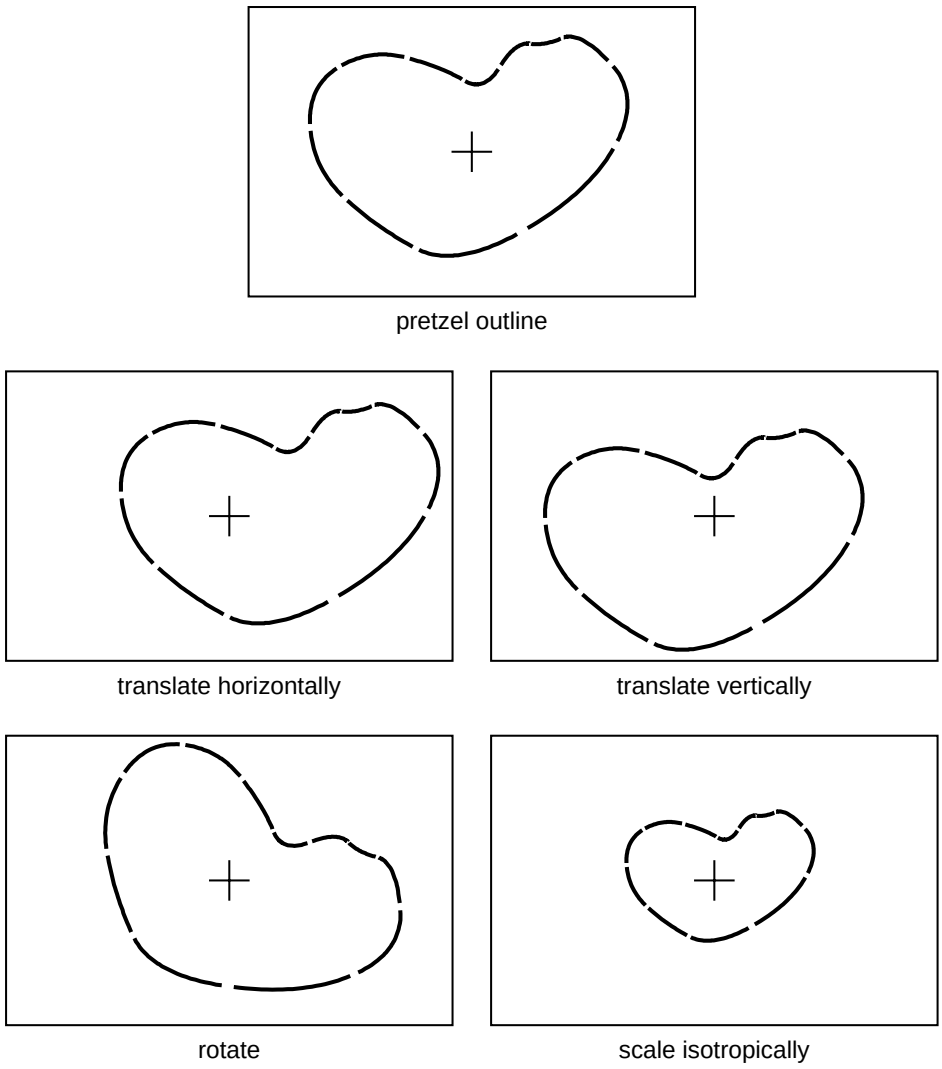


Figure 4.2: Euclidean similarities. *The shape-space of Euclidean similarities has 4 degrees of freedom. They are depicted here as applied to the outline of a pretzel (the pretzel from figure 3.13 on page 57 of the previous chapter).*

dimensions and to rotate about an axis perpendicular to the table top (something that occurs, for example, when a camera is mounted on a "SCARA" arm, popular in robot automation, which translates freely and rotates in the plane of the table). This combination also sweeps out a shape-space of Euclidean similarities.

Another important shape-space is the one that arises when a planar object has complete freedom to move in three dimensions. Its motion has 6 degrees of freedom, three for translation and three for rotation. Provided perspective effects are not too great, the image of a planar object contour is well described as a shape-space of planar affine transformations, a space with dimension 6. It can be thought of as the space of linear transformations of a template. Alternatively, it is the space of transformations which preserves parallelism between lines. The planar affine group of transformations is depicted in figure 4.3. Figure 4.4 illustrates how the planar affine shape-space can enhance active contours when used appropriately. The figure shows tracking of an outstretched hand which, being almost planar, is well modelled by a planar affine space. The increased degree of constraint enhances immunity to distraction from clutter in the background.

The planar affine and Euclidean similarity shape-spaces work efficiently in the sense that the dimension of the shape-space is exactly equal to the number (six/four respectively) of the degrees of freedom of camera movement. Unfortunately this happy state of affairs does not persist in general because transformation groups do not necessarily form vector spaces; it is not always possible to find a vector space which matches exactly the degrees of freedom of camera/object motion. Consider the case of a camera with fixed magnification, viewing a planar object moving rigidly on a table. The image curve translates and rotates rigidly without any change of size. This is now simply the planar Euclidean group which does not however form a vector space. To see this, consider the template $\mathbf{r}_0(s)$ and a copy of it rotated through 180^o to give $-\mathbf{r}_0(s)$; when these two are added vectorially they give $\mathbf{r}_0(s) + (-\mathbf{r}_0(s)) = \mathbf{0}$ which is not a rotated version of the template at all. The rotation operation is therefore not "closed" under addition and therefore cannot form a vector space. Rotation and scaling taken *jointly* do form a vector space, of dimension 2. Combining them with translation gives the Euclidean similarities, of dimension 4. The smallest vector space that encompasses Euclidean transformations is therefore the space of Euclidean similarities. The price of insisting on a linear representation of the Euclidean transformations is that 4 dimensions are needed to represent 3 degrees of freedom; the resulting space is underconstrained by one degree of freedom.

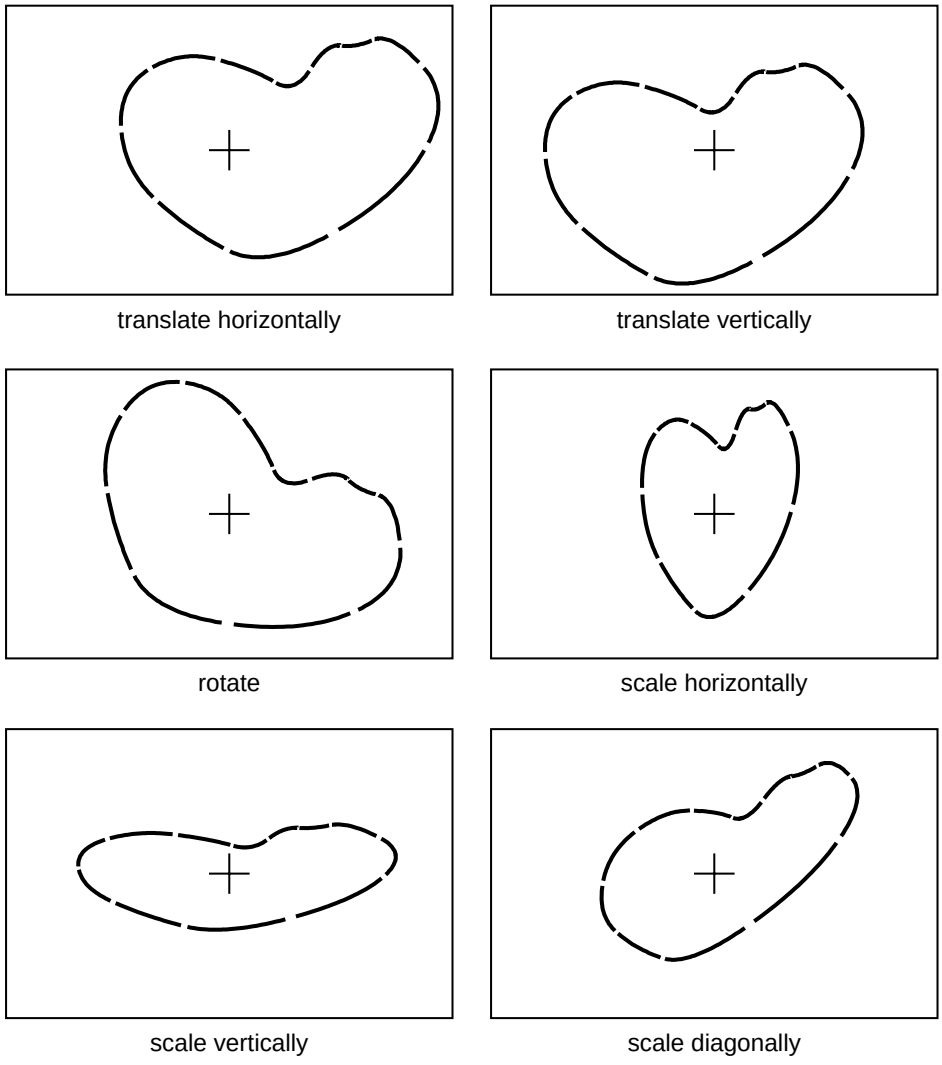


Figure 4.3: Planar affine basis. *The planar affine transformation group has 6 degrees of freedom. A basis for them is depicted here, as applied to the pretzel outline from figure 4.2 on page 71. The first three elements of the basis correspond to the first three for the Euclidean similarities. The last three elements span a subspace that includes the fourth element — scaling — for the Euclidean similarities and two further degrees of freedom for directional scaling. Directional scaling occurs when a planar object, initially co-planar with the image, is allowed to rotate about an axis that lies parallel to the image plane.*

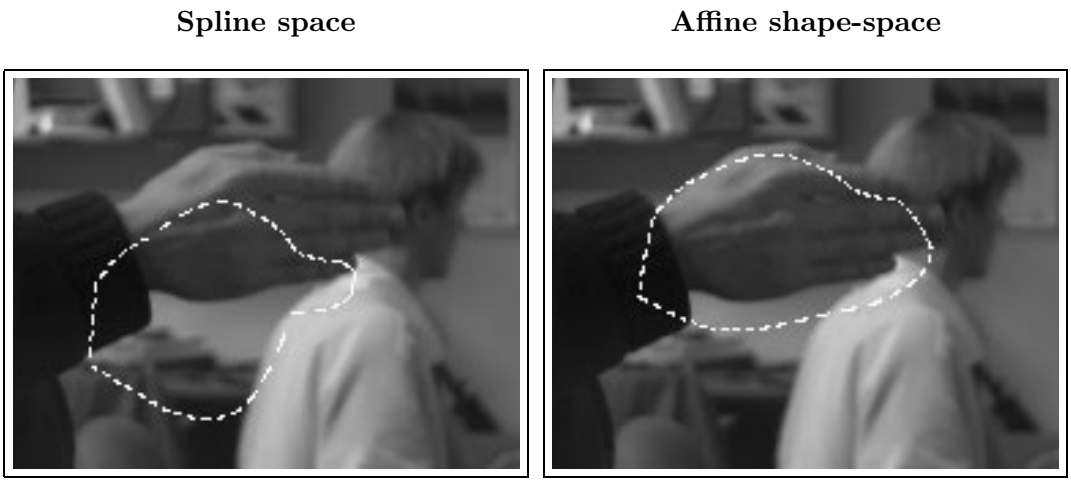


Figure 4.4: Shape-space can impose constraints that allow background clutter to be ignored. *A test sequence of hand motion against background clutter consists of vertical oscillation at around 0.5 Hz. The tracking algorithm developed later in the book (chapter 10) is applied over spline space and over an affine shape-space. The figure depicts snapshots after 9 seconds of tracking. It appears that the use of a planar affine shape-space confers enhanced immunity to background clutter.*

Definition of shape-space

At this stage, a more precise definition of shape-space is called for. A shape-space $\mathcal{S} = \mathcal{L}(W, \mathbf{Q}_0)$ is a linear mapping of a "shape-space vector" $\mathbf{X} \in \mathbb{R}^{N_X}$ to a spline-vector $\mathbf{Q} \in \mathbb{R}^{N_Q}$:

$$\mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0, \tag{4.1}$$

where W is a $N_Q \times N_X$ "shape-matrix." The constant offset $\mathbf{Q}_0$ is a template curve against which shape variations are measured; for instance, a class of shapes consisting of $\mathbf{Q}_0$ and curves close to $\mathbf{Q}_0$ could be expressed by restricting the shape-space $\mathcal{S}$ to "small" $\mathbf{X}$. The image of $\mathbb{R}^{N_X}$ need not necessarily be a vector space itself but is a "coset" — an underlying vector space $\{W\mathbf{X},\ \mathbf{X} \in \mathbb{R}^{N_X}\}$ plus an offset $\mathbf{Q}_0$. We talk of the "basis" $\mathcal{V}$ of a shape-space meaning a basis for the underlying vector space. The matrix W is comprised of columns which are the vectors of the basis $\mathcal{V}$. In fact the two spaces discussed in this chapter — Euclidean similarity and Affine, are vector spaces,

because there exists an $\mathbf{X}$ for which $\mathbf{Q} = W\mathbf{X}$. In chapter 8 we encounter shape-spaces whose images are not vector spaces because the offset $\mathbf{Q}_0$ is linearly independent of the basis $\mathcal{V}$. In fact the simplest shape-space that is not a pure vector space is the space of translations of a template $\mathbf{Q}_0$.

4.2 The space of Euclidean similarities

The Euclidean similarities of a template curve $\mathbf{r}_0(s)$ represented by $\mathbf{Q}_0$ form a 4-dimensional shape-space $\mathcal{S}$ with shape-matrix

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x \end{pmatrix} \tag{4.2}$$

where the N_B-vectors $\mathbf{0}$ and $\mathbf{1}$ are:

$$\mathbf{0} = (0, 0, \ldots, 0)^T, \quad \mathbf{1} = (1, 1, \ldots, 1)^T.$$

The first two columns of W govern horizontal and vertical translations respectively. The third and fourth columns, made up from components of the spline-vector $\mathbf{Q}_0$ for the template, cover rotation and scaling. By convention, we choose $\mathbf{Q}_0$ to have its centroid at the origin ($< \mathbf{Q}_0^x, \mathbf{1} >=< \mathbf{Q}_0^y, \mathbf{1} >= 0$) so that the third and fourth columns are associated with *pure* rotation and scaling, free of translation. In practice the template is obtained by fitting a spline interactively around a standard view of the shape, and translating it so that its centroid lies over the origin.

Some examples of shape representations in the space of Euclidean similarities follow.

1. $\mathbf{X} = (0, 0, 0, 0)^T$ represents the original template shape $\mathbf{Q}_0$
2. $\mathbf{X} = (1, 0, 0, 0)^T$ represents the template translated 1 unit to the right, so that, from (4.1),
$$\mathbf{Q} = \mathbf{Q}_0 + \begin{pmatrix} \mathbf{1} \\ \mathbf{0} \end{pmatrix}$$
3. $\mathbf{X} = (0, 0, 1, 0)^T$ represents the template doubled in size
$$\mathbf{Q} = 2\mathbf{Q}_0$$

4. $\mathbf{X} = (0, 0, \cos\theta - 1, \sin\theta)^T$ represents the template rotated through angle θ:

$$\mathbf{Q} = \begin{pmatrix} \cos\theta\ \mathbf{Q}_0^x - \sin\theta\ \mathbf{Q}_0^y \\ \\ \sin\theta\ \mathbf{Q}_0^x + \cos\theta\ \mathbf{Q}_0^y \end{pmatrix}.$$

As an example, the lotion bottle in figure 4.5 moves rigidly from the template configuration $\mathbf{X} = 0$ in shape-space to the configuration

$$\mathbf{X} = (0.465, 0.047, -0.282, -0.698)^T$$

representing a translation through $(0.465, 0.047)^T$, almost horizontal as the figure shows, a magnification by a factor

$$\sqrt{(1 - 0.282)^2 + 0.698^2} = 1.001$$

and a rotation through

$$\arctan(1 - 0.282, -0.698) = -44.2^o$$

— all consistent with the figure.

4.3 Planar affine shape-space

It was claimed that for a planar shape just six *affine* degrees of freedom are required to describe, to a good approximation, the possible shapes of its bounding curve. The planar affine group can be viewed as the class of all *linear* transformations that can be applied to a template curve $\mathbf{r}_0(s)$:

$$\mathbf{r}(s) = \mathbf{u} + M\mathbf{r}_0(s), \tag{4.3}$$

where $\mathbf{u} = (u_1, u_2)^T$ is a two-dimensional translation vector and M is a 2×2 matrix, so that $M, \mathbf{u}$ between them represent the 6 degrees of freedom of the space. This class can be represented as a shape-space with template $\mathbf{Q}_0$ and shape-matrix:

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & \mathbf{0} & \mathbf{0} & \mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^y & \mathbf{Q}_0^x & \mathbf{0} \end{pmatrix}. \tag{4.4}$$

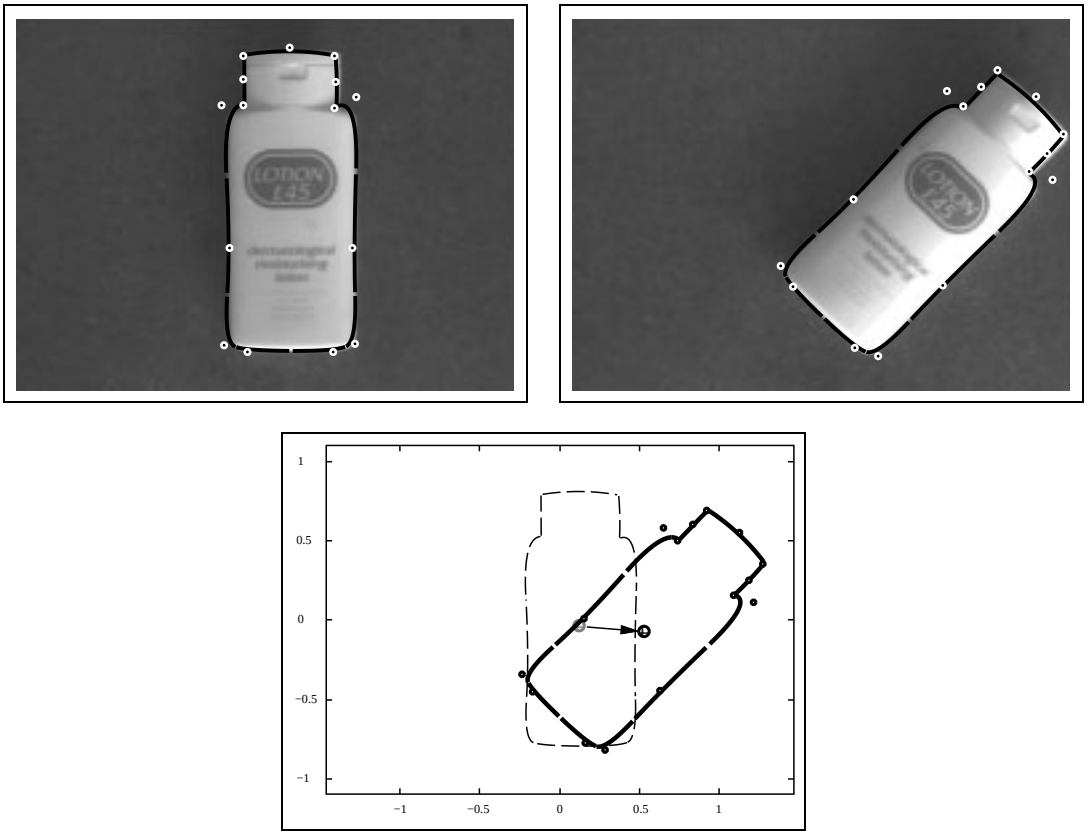


Figure 4.5: Euclidean similarities. *The outline of a bottle in a standard position (left) is taken as a template for a shape-space of Euclidean similarities in which any new outline (right) can be described. In this case the new outline is displaced and rotated relative to template (bottom) and this is apparent from the shape-space representation — see text.*

(A derivation is given below.) The first two columns of W represent horizontal and vertical translation. As before, by convention, the template $\mathbf{r}_0(s)$ represented by $\mathbf{Q}_0$ is chosen with its centroid at the origin. Then the remaining four affine motions (figure 4.3), which do not correspond one-for-one the last four columns of W, can however be expressed as simple linear combinations of those columns. Recall that the shape-space transformation is $\mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0$ so that the elements of $\mathbf{X}$ act as weights on the columns of W. The interpretation of those weights in terms of planar

transformations (4.3) of the template is:

$$\mathbf{X} = (u_1, u_2, M_{11} - 1, M_{22} - 1, M_{21}, M_{12})^T. \tag{4.5}$$

Some examples of transformations are:

1. $\mathbf{X} = (0, 0, 0, 0, 0, 0)^T$ represents the original template shape $\mathbf{Q}_0$
2. $\mathbf{X} = (1, 0, 0, 0, 0, 0)^T$ represents the template translated 1 unit to the right,
3. $\mathbf{X} = (0, 0, 1, 1, 0, 0)^T$ represents the template doubled in size
4. $\mathbf{X} = (0, 0, \cos\theta - 1, \cos\theta - 1, -\sin\theta, \sin\theta)^T$ represents the template rotated through angle θ
5. $\mathbf{X} = (0, 0, 1, 0, 0, 0)^T$ represents the template doubled in width

In practice it is convenient to arrange for the elements of the affine basis to have similar magnitudes to improve numerical stability. If the control-vector $\mathbf{Q}_0$ is expressed in pixels, for computational simplicity, the magnitudes of the last four columns of the shape-matrix may be several hundred times larger than those of the first two, and it is then necessary to scale the translation columns to match.

Derivation of affine basis. Using (3.19), (4.3) can be rewritten:

$$\mathbf{r}(s) - \mathbf{r}_0(s) = \mathbf{u} + (M - I)U(s)\mathbf{Q}_0.$$

Now using the definition (3.20) of $U(s)$ and noting that $\mathbf{B}(s)^T\mathbf{1} = 1$ (3.5), this becomes:

$$\begin{aligned}
\mathbf{r}(s) - \mathbf{r}_0(s) &= \begin{pmatrix} u_1\mathbf{B}^T(s)\mathbf{1} \\ u_2\mathbf{B}^T(s)\mathbf{1} \end{pmatrix} + \begin{pmatrix} (M_{11}-1)\mathbf{B}^T(s)\mathbf{Q}_0^x + M_{12}\mathbf{B}^T(s)\mathbf{Q}_0^y \\ M_{21}\mathbf{B}^T(s)\mathbf{Q}_0^x + (M_{22}-1)\mathbf{B}^T(s)\mathbf{Q}_0^y \end{pmatrix} \\
&= u_1U(s)\begin{pmatrix} \mathbf{1} \\ \mathbf{0} \end{pmatrix} + u_2U(s)\begin{pmatrix} \mathbf{0} \\ \mathbf{1} \end{pmatrix} + (M_{11}-1)U(s)\begin{pmatrix} \mathbf{Q}_0^x \\ \mathbf{0} \end{pmatrix} \\
&+ M_{12}U(s)\begin{pmatrix} \mathbf{Q}_0^y \\ \mathbf{0} \end{pmatrix} + M_{21}U(s)\begin{pmatrix} \mathbf{0} \\ \mathbf{Q}_0^x \end{pmatrix} + (M_{22}-1)U(s)\begin{pmatrix} \mathbf{0} \\ \mathbf{Q}_0^y \end{pmatrix}.
\end{aligned}$$

From (3.19),

$$\mathbf{r}(s) - \mathbf{r}_0(s) = U(s)(\mathbf{Q} - \mathbf{Q}_0)$$

and comparing this with the expression for $\mathbf{r}(s) - \mathbf{r}_0(s)$ above shows that $\mathbf{Q} - \mathbf{Q}_0$ belongs to a vector space of dimension 6, for which the rows of W in (4.4) form a basis, and furthermore, given that $\mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0$, $\mathbf{X}$ is composed of elements of M and $\mathbf{u}$ as in (4.5).

4.4 Norms and moments in a shape-space

Given that it is generally preferred to work in a shape-space $\mathcal{S}$, a formula for the curve norm is needed that applies to the shape-space parameter $\mathbf{X}$. We require a consistent definition so that, for a given space, $\|\mathbf{Q}_1 - \mathbf{Q}_2\| = \|\mathbf{X}_1 - \mathbf{X}_2\|$. The L_2 norm in shape-space $\mathcal{S}$ is said to be "induced" from the norm over $\mathcal{S}_Q$, which was in turn induced from the L_2 norm over the space of curves $\mathbf{r}(s)$. From (4.1), this is achieved by defining:

$$\|\mathbf{X}\| = \sqrt{\mathbf{X}^T \mathcal{H} \mathbf{X}}, \tag{4.6}$$

where

$$\mathcal{H} = W^T \mathcal{U} W. \tag{4.7}$$

The norm over $\mathcal{S}$ has a geometric interpretation:

$$\|\mathbf{X}\| = \|\mathbf{Q} - \mathbf{Q}_0\|$$

is the average displacement of the curve parameterised by $\mathbf{X}$ from the template curve. We can also now define a natural mapping from $\mathcal{S}_Q$ onto the shape-space $\mathcal{S}$. Of course there is in general no inverse of the mapping W in (4.1) from $\mathcal{S}_Q$ to $\mathcal{X}$ but, providing W has full rank (its columns are linearly independent), a pseudo-inverse W^+ can be defined:

$$\mathbf{X} = W^+(\mathbf{Q} - \mathbf{Q}_0) \quad \text{where} \quad W^+ = \mathcal{H}^{-1} W^T \mathcal{U}. \tag{4.8}$$

It turns out (see chapter 6) that W^+ can be naturally interpreted as an error-minimising *projection* onto shape-space.

In the case of spline space, it was argued in the previous chapter, the Euclidean norm $|\cdot|$ defined by $|\mathbf{Q}|^2 \equiv \mathbf{Q}^T\mathbf{Q}$ is not as natural as the L_2-norm $\|\mathbf{Q}\|$, although the two can have approximately similar values in practice, especially when curvature is small. Their approximate similarity derives from the fact that the metric matrix $\mathcal{U}$ is banded. However, the matrix $\mathcal{H}$ above is dense and so the Euclidean norm $|\mathbf{X}|$

in shape-space does not approximate the induced L_2-norm, and in fact $|\mathbf{X}|$ has no obvious geometric interpretation. Therefore, while one might get away with using the Euclidean norm in spline space, it is of no use at all in shape-space — only the L_2-norm will do.

Computing Area

As with the norm, the area form $A(\mathbf{X})$ can be expressed in a shape-space as a function

$$A(\mathbf{X}) = (W\mathbf{X} + \mathbf{Q}_0)^T \mathcal{A} (W\mathbf{X} + \mathbf{Q}_0)$$

that is quadratic in $\mathbf{X}$, and whose quadratic and linear terms involve just $N_X(N_X + 3)/2$ coefficients so that, in the case of Euclidean similarity, there are just 14 independent coefficients.

Centroid and inertia

The centroid $\overline{\mathbf{r}}$ of the area enclosed by a closed B-spline curve ((3.28) on page 66) is a symmetric cubic function of the configuration $\mathbf{X}$. Such a function has $O((N_X)^3)$ terms which is obviously large for larger shape-spaces, but works out to be just 20 terms in the case of Euclidean similarities — quite practical to compute. (The exact formula for the number of terms is $N_X(N_X + 1)(N_X + 2)/6$.) The invariant second moment or inertia matrix ((3.29) on page 66) could be expressed in terms of a symmetric quartic form which, surprisingly, has only 23 terms in the case of Euclidean similarity ($N_X = 4$) but of course this number is $O((N_X)^4)$ in general. In cases where the size of the configuration space is too large for efficient computation of invariant moments, the alternative is to compute them by numerical integration.

Finally, note that there is an important special case in which invariant moments are easily computed. The special properties of the affine space mean that moments can be computed efficiently under affine transformations. For instance, although the invariant second moment for a 6-dimensional shape-space turns out generally to be a quartic polynomial with 101 terms, in the affine space it can be computed simply as a product of three 2×2 matrices:

$$\mathcal{I} = M \mathcal{I}_0 M^T$$

where $\mathcal{I}_0$ is the inertia matrix of the template, computed numerically from (3.28) on page 66. Similarly, for area:

$$A = (\det M) A_0.$$

Note that $\mathcal{I}$ represents only 3 constraints on M and one of those is duplicated by the area A. So $\mathcal{I}$, $\overline{\mathbf{r}}$ and A between them give only 5 constraints on the 6 affine degrees of freedom. It would be necessary to compute higher moments to fix all 6 constraints. On the other hand, those moments up to second-order are sufficient to fix a vector in the space of Euclidean similarities.

Using moments for initialisation

For the Euclidean similarities, a shape-vector $\mathbf{X}$ can be recovered from the moments up to second order, as follows.

1. The displacement of the centroid $\overline{\mathbf{r}}$ gives the translational component of $\mathbf{X}$.
2. The scaling is given by $\sqrt{A/A_0}$.
3. The rotation is the angle θ through which the largest eigenvector of $\mathcal{I}$ rotates.

This procedure is illustrated in figure 4.6 below. In the illustration given here, moments were computed over a foreground patch segmented from the background on the basis of colour. An effective method for certain problems such as vehicle tracking, in which the foreground moves over a stationary background, is to use image motion. So-called "optical flow" is computed over the image, and a region of moving pixels is delineated. Either a snake may be wrapped around this region directly or, in a shape-space, moments can be computed for initialisation as above. Background subtraction methods are also useful here — see chapter 5 for an explanation.

4.5 Perspective and weak perspective

The next task is to set up notation for perspective projection in order to show that, under modest approximations, the set of possible shapes of image contours do indeed form affine spaces. Standard camera geometry is shown in figure 4.7 and leads to the following relationship between a three-dimensional object contour $\mathbf{R}(s)$ and its two-dimensional image $\mathbf{r}(s)$:

$$\mathbf{r}(s) = \frac{f}{Z(s)} \begin{pmatrix} X(s) \\ Y(s) \end{pmatrix} \quad \text{where} \quad \mathbf{R}(s) = \begin{pmatrix} X(s) \\ Y(s) \\ Z(s) \end{pmatrix}. \tag{4.9}$$

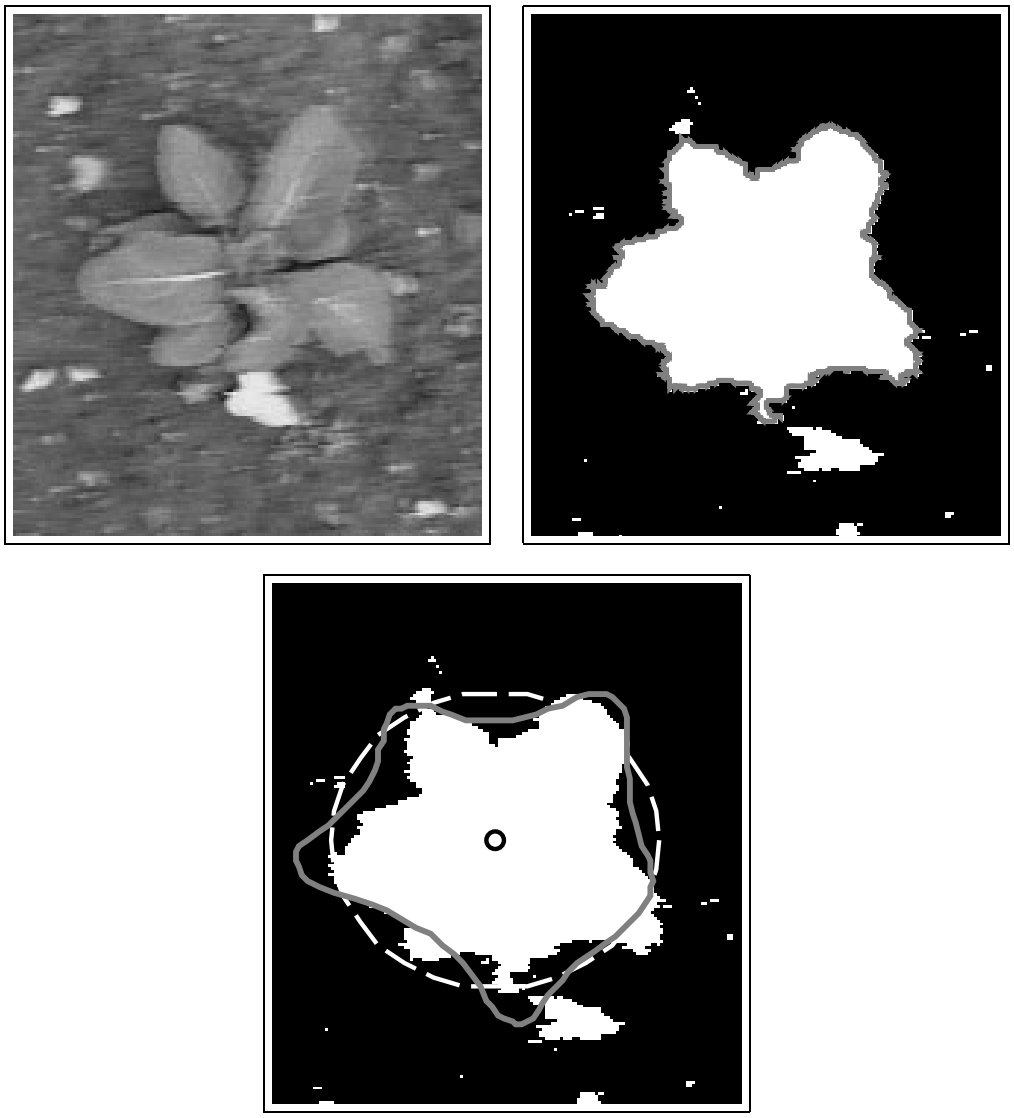


Figure 4.6: A contour is initialised using moments. *An image (left) (colour version in figure 1.6 on page 10) is processed using straightforward colour segmentation to obtain an interior region (right). Moments of the outline are calculated; the second moment is shown as an ellipse (bottom) and used to initialise a curve in a shape-space of Euclidean similarities (grey curve).*

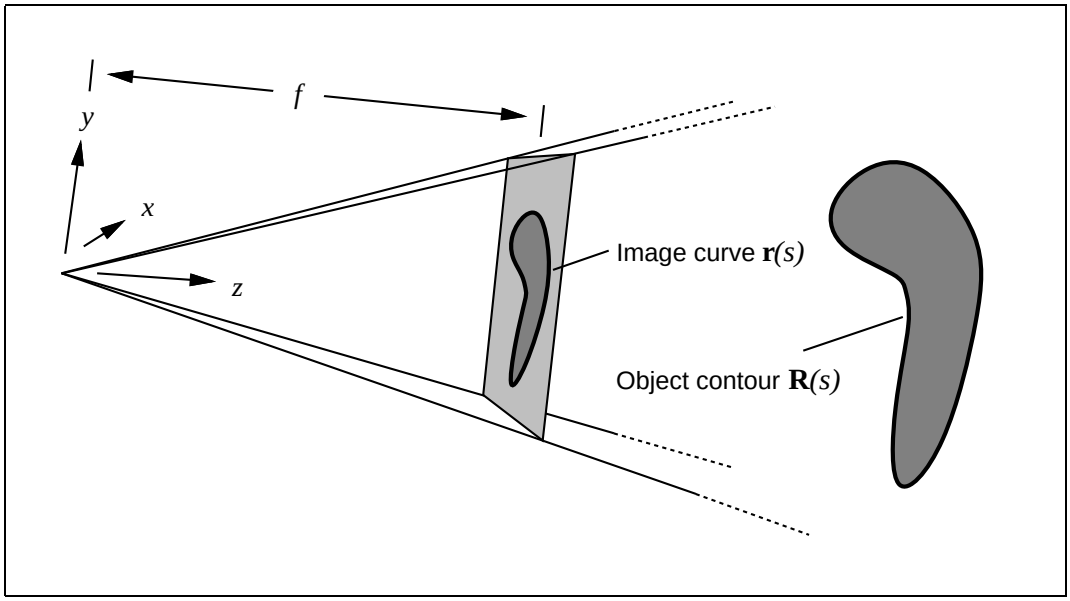


Figure 4.7: *It is a universally accepted fiction that, contrary to the layout of real cameras, the centre of projection in a mathematical camera is taken to lie* behind *the image plane. The focal length is f, measured in the same units as x, y and z, say mm for convenience.*

The $1/Z$ term is intuitively reasonable as it represents the tendency of objects to appear smaller as they recede from the camera. However, it makes the projection function non-linear which is problematic given that shape-spaces, being vector spaces, imply linearity. Fortunately there are well-established methods for making good linear approximations to perspective. The most general of these is the *weak perspective* projection.

Note that a good approximate value for f can be obtained simply by using the nominal value usually printed on the side of a lens housing, which we denote f_∞. To a first approximation, $f = f_\infty$, but a better one, taking into account the working distance Z_c, is

$$f = f_\infty \left(1 - \frac{f_\infty}{Z_c}\right)^{-1}. \tag{4.10}$$

Since image positions x, y available to a computer are measured in units of pixels relative to one corner of the camera array, a scale factor is needed to convert pixel units into length units (mm). This can be done quite effectively by taking a picture,

as in figure 4.8, of a ruler lying in a plane parallel to the image plane. Moving a

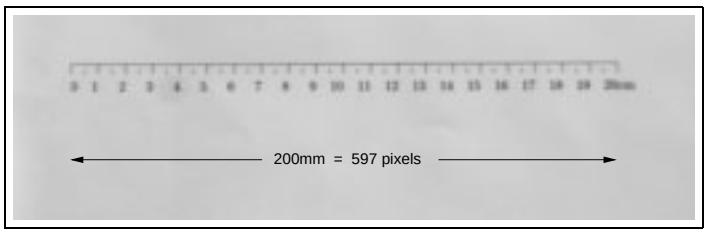


Figure 4.8: A simple camera calibration procedure. *The ruler is set at a distance 1060 mm from the camera iris.*

cursor over the image shows that 597 pixels corresponds to 200 mm at a distance of $Z_c = 1060$ mm from the camera iris, and the nominal focal length is $f_\infty = 25$ mm. From (4.10) we have $f = 25.6$ mm, and the scaling factor for distance x on the image plane is

$$\frac{200}{597}\frac{f}{Z_c}\,\text{mm/pixel} = \frac{200}{597}\frac{25.6}{1060}\,\text{mm/pixel} = 8.09 \times 10^{-3}\,\text{mm/pixel}.$$

(This puts the width of the entire physical camera array of 768 pixels at $768 \times 8.09 \times 10^{-3}$ mm or 6.21 mm which is a very reasonable figure). Of course, for cameras whose pixels are not square, this procedure must be repeated in the vertical direction to calculate the vertical scaling factor.

Finally, note that more precise calculations, including allowances for minor mechanical defects such as asymmetry of lens placement with respect to the image array, can be made precisely using automatic but somewhat involved "camera calibration" procedures. However, the simple procedure above has proved sufficient for active contour interpretation, in most cases.

Weak Perspective

The weak perspective approximation is valid provided that the three-dimensional object is bounded so that its diameter is small compared with the distance from camera to object. Taking $\mathbf{R}_c = (X_c, Y_c, Z_c)^T$ to be a point close to the object — think of it as the object's centre, replace $\mathbf{R}(s)$ in the projection formula by $\mathbf{R}_c + \mathbf{R}(s)$ and then the assumption about object diameter can be written as

$$|\mathbf{R}(s)| \ll Z_c \;\forall\, s. \tag{4.11}$$

A useful alternative form of the assumption is that the subtended angle of the *image* contour is much less than 1 radian when viewed from any aspect.

That assumption can now be used in the perspective equation (4.9) to give the weak perspective projection:

$$\mathbf{r}(s) = \frac{f}{Z_c}\left[\begin{pmatrix} X_c \\ Y_c \end{pmatrix} + \begin{pmatrix} X(s) \\ Y(s) \end{pmatrix} - \frac{Z(s)}{Z_c}\begin{pmatrix} X_c \\ Y_c \end{pmatrix}\right] \tag{4.12}$$

which is linear in $\mathbf{R}(s)$ and approximates perspective projection to first order in $|\mathbf{R}(s)|/Z_c$. The tendency of image size to diminish as an object recedes is present in the f/Z_c term, now approximated to an "average" value for a given object. As individual points of $\mathbf{R}(s)$ recede they tend to move towards the centre of the image and the third term expresses this. In typical views, the approximation works well, as figure 4.9 shows. If, in addition to the camera having a large field of view, the object also fills that field of view, then errors in the weak perspective approximation become significant. That is not a situation that commonly arises in object tracking however. If the camera is mounted on a pan-tilt head, the camera's field of view is likely to be narrow in order to obtain the improved resolution that the movable head allows. Alternatively, when the camera is fixed, the image diameter is likely to be several times smaller than the field of view to allow for object movement. Since the field of view of a wide-angle camera lens is of the order of 1 radian, it follows that object diameter is likely to be considerably less than 1 radian, precisely the condition for the weak perspective approximation to hold good.

Orthographic projection

For a camera with a narrow field of view (substantially less than one radian) it can further be assumed, in addition to the assumption (4.11) about object diameter, that

$$|X_c| \ll Z_c \quad \text{and} \quad |Y_c| \ll Z_c \tag{4.13}$$

— simply the condition that the contour centre is close enough to the centre of the image for the object actually to be visible. In that case, the third term in (4.12) is negligible and image perspective is well approximated by the *orthographic* projection

$$\mathbf{r}(s) = \frac{f}{Z_c}\begin{pmatrix} X_c + X(s) \\ Y_c + Y(s) \end{pmatrix}. \tag{4.14}$$

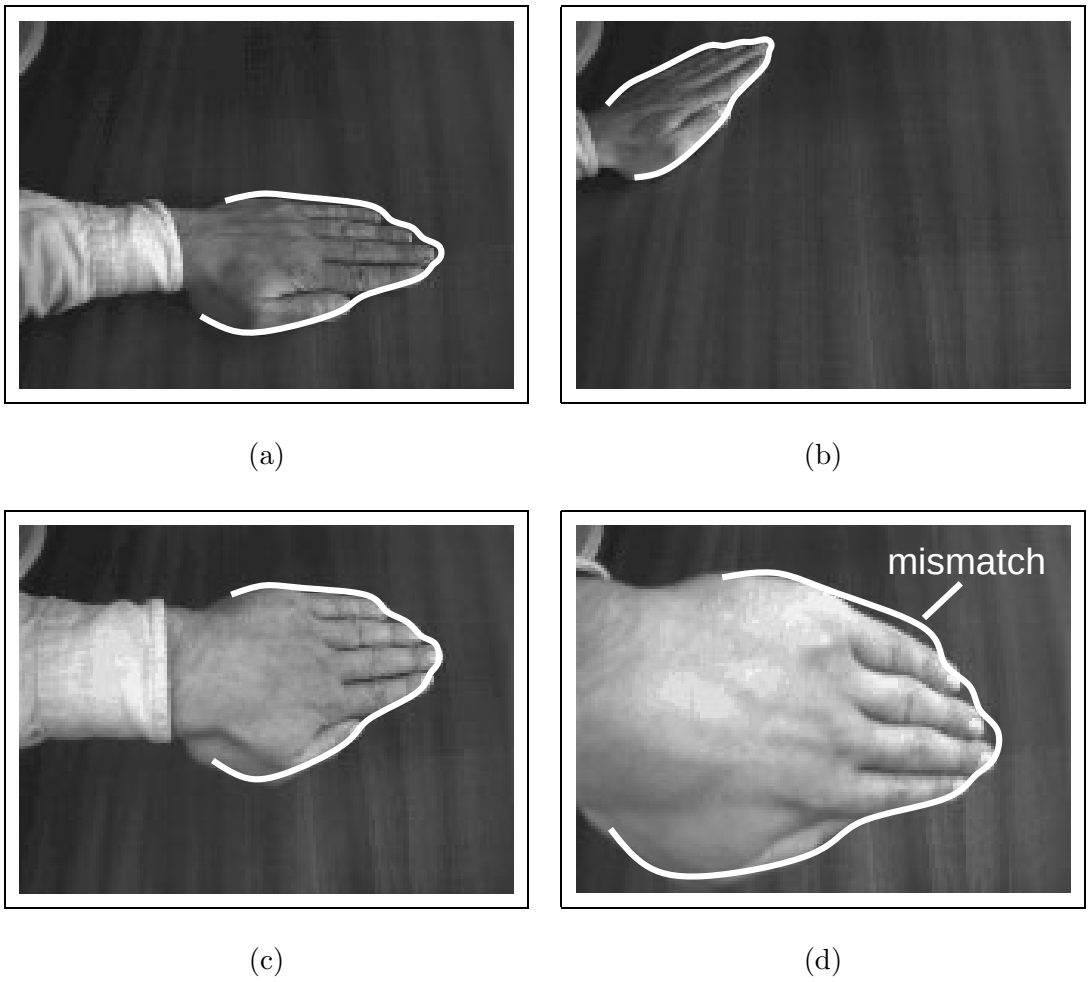


Figure 4.9: The weak perspective approximation is normally accurate. *(a–c) This image of a hand being tracked in a camera with a wide field of view shows that the weak perspective image of the hand outline closely approximates the true perspective image. (d) Under extreme viewing conditions, when perspective effects are strong, approximation error may be appreciable, visible here as the mismatch of the curve around the fingers.*

Suppose the object contour $\mathbf{R}_c + \mathbf{R}(s)$ derives from a contour $\mathbf{R}_0(s)$ in a base coordinate frame which has then been rotated to give $\mathbf{R}(s) = R\mathbf{R}_0(s)$ and translated through $\mathbf{R}_c$, so that $R, \mathbf{R}_c$ are parameters for three-dimensional motion. Suppose also that the object is planar so (without loss of generality) $Z_0(s) = 0$. Then the orthographic projection equation becomes

$$\mathbf{r}(s) = \mathbf{u} + \frac{f}{Z_c} R_{2\times 2} \begin{pmatrix} X_0(s) \\ Y_0(s) \end{pmatrix}$$

where $\mathbf{u}$ is the orthographic projection of the three-dimensional displacement vector $\mathbf{R}_c$ and $R_{2\times 2}$ is the upper-left 2×2 block of the rotation matrix R. Finally, take $M = (f/Z_c)R_{2\times 2}$ and adopt the convention that $Z_c = f$ in the standard view so that

$$\mathbf{r}_0(s) = (X_0(s), Y_0(s))^T$$

is the image template. This gives a general planar affine transformation as in (4.3), so the image of a planar object moving rigidly in three dimensions does indeed sweep out a planar affine shape-space.

If the orthographic constraint (4.13) is relaxed again, to allow general weak perspective, it turns out that, when $\mathbf{R}(s)$ is planar, $\mathbf{r}(s)$ still inhabits the planar affine shape-space. Later we return to weak perspective for a general analysis of planar affine configurations, in particular to work out the three-dimensional pose of an object from its affine coordinates. This is used, for example, to calculate the three-dimensional position and attitude of the hand in the mouse application of figure 1.16 on page 20. That general method of pose calculation will work even when the camera is positioned obliquely relative to table-top coordinates and when the hand moves over the whole of a wide field of view.

4.6 Three-dimensional affine shape-space

Shape-space for a non-planar object is derived as a modest extension of the planar case. The object concerned should be visualised as a piece of bent wire, rather than a smooth three-dimensional surface. Smooth surfaces are, of course, of great interest but shape-space treatment is more difficult because of the complex geometrical behaviour of silhouettes. The bent wire model also implies freedom from hidden lines; the approach described here deals with parallax effects arising from three-dimensional

shape but not with the problem of "occlusion" for which additional machinery is needed.

Clearly the 6-dimensional planar affine shape-space cannot be expected to suffice for non-planar surfaces and this is illustrated in figure 4.10. The new shape-space is

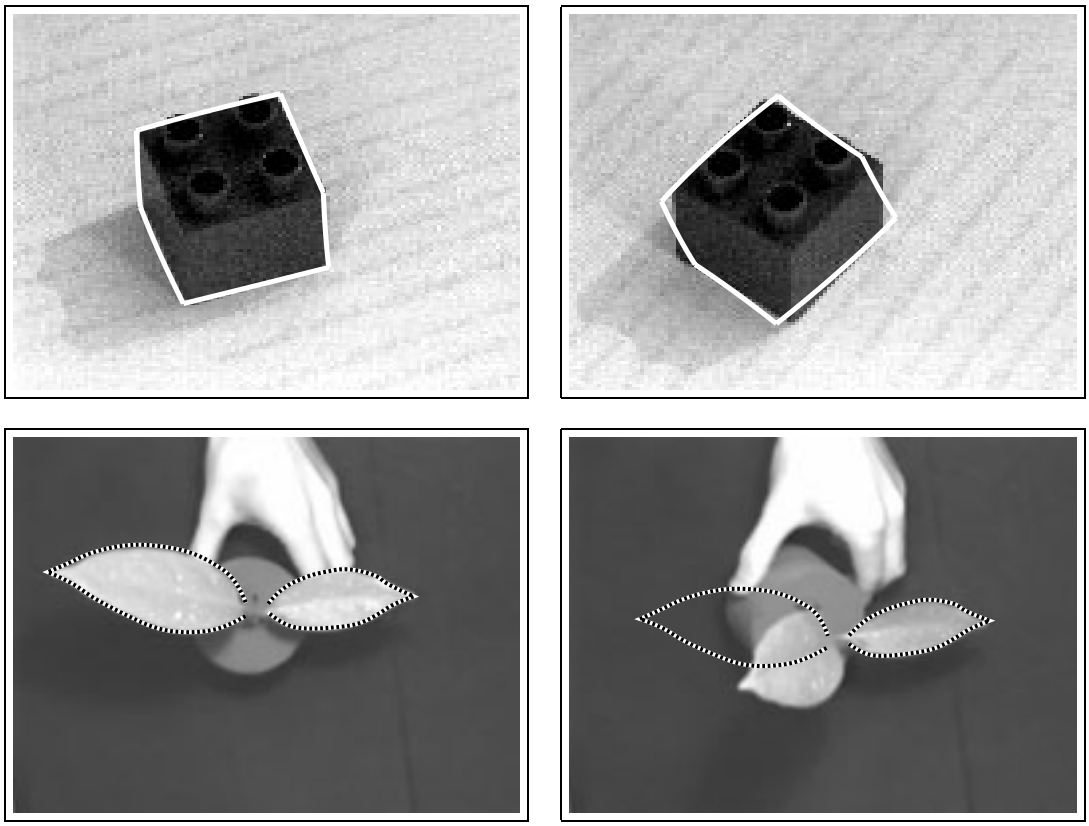


Figure 4.10: The views of a general three-dimensional contour cannot be encompassed by a planar affine shape-space. *A planar affine space of contours has been generated from the outline of the first view of the cube. The outlines of subsequent views do not however lie in the space, as evidenced by the visible mismatch in the fitted contours. (Figure reprinted from (Curwen, 1993).) A similar effect is observed with the leaves.*

"three-dimensional affine" with 8 degrees of freedom, made up of the six-parameter planar affine space and a two-parameter extension. Consider the object to be a three-

dimensional curve

$$\mathbf{R}_0(s) = (X_0(s), Y_0(s), Z_0(s))^T$$

which is projected orthographically as in (4.14) to give an image curve

$$\mathbf{r}(s) = \mathbf{u} + \frac{f}{Z_c} R_{2\times 3} \begin{pmatrix} X_0(s) \\ Y_0(s) \\ Z_0(s) \end{pmatrix}$$

and this can be expressed as the standard planar affine transformation $(\mathbf{u}, M)$ of (4.3) with an additional depth-dependent term:

$$\mathbf{r}(s) = \mathbf{u} + M\mathbf{r}_0(s) + \mathbf{v}Z_0(s) \tag{4.15}$$

where

$$R_{2\times 3} = \frac{Z_c}{f} (M \mid \mathbf{v}) . \tag{4.16}$$

The three-dimensional shape-space therefore consists of the two-dimensional one for the planar affine space generated by template $\mathbf{Q}_0$, with two added components to account for the depth variation that is not visible in the template view. The two additional basis elements are:

$$\mathcal{V}' = \left\{ \begin{pmatrix} \mathbf{Q}_0^z \\ \mathbf{0} \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ \mathbf{Q}_0^z \end{pmatrix} \right\}.$$

The extra two elements are tacked onto the planar affine W-matrix (4.4) to form the W-matrix for the three-dimensional case:

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & \mathbf{0} & \mathbf{0} & \mathbf{Q}_0^y & \mathbf{Q}_0^z & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^y & \mathbf{Q}_0^x & \mathbf{0} & \mathbf{0} & \mathbf{Q}_0^z \end{pmatrix}. \tag{4.17}$$

Just as equation (4.5) provided a conversion from the planar affine shape-space to the real-world transformation, the three-dimensional affine shape-space components have the following interpretation:

$$\mathbf{X} = (u_1, u_2, M_{11} - 1, M_{22} - 1, M_{21}, M_{12}, v_1, v_2). \tag{4.18}$$

The expanded space now encompasses the outlines of all views of the three-dimensional outline as figure 4.11 shows. Automatic methods for determining $\mathbf{Q}_0^z$ from example views are discussed in chapter 7.

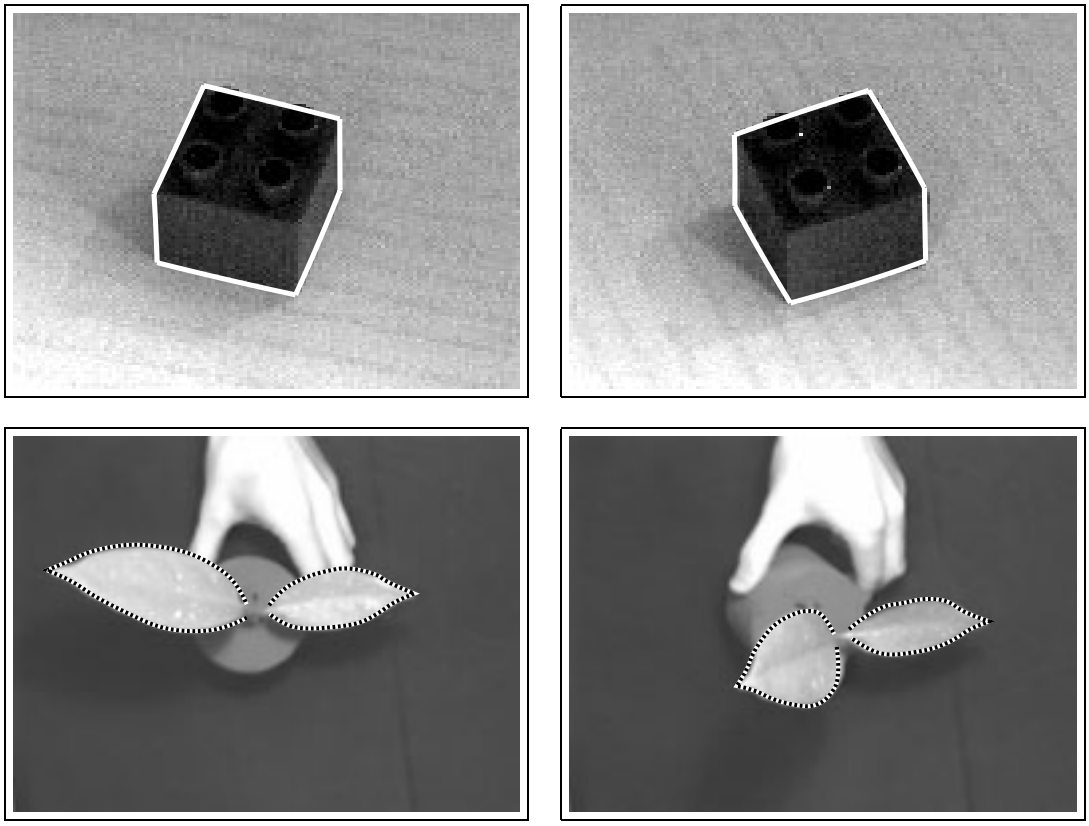


Figure 4.11: Three-dimensional affine shape-space. *The outlines of views of a cube which could not be contained in a planar affine shape-space now fall within a suitably constructed 3D affine space. (Figure reprinted from (Curwen, 1993).) Similarly, the non-planar arrangement of leaves is happily encompassed by a 3D affine space.*

4.7 Key-frames

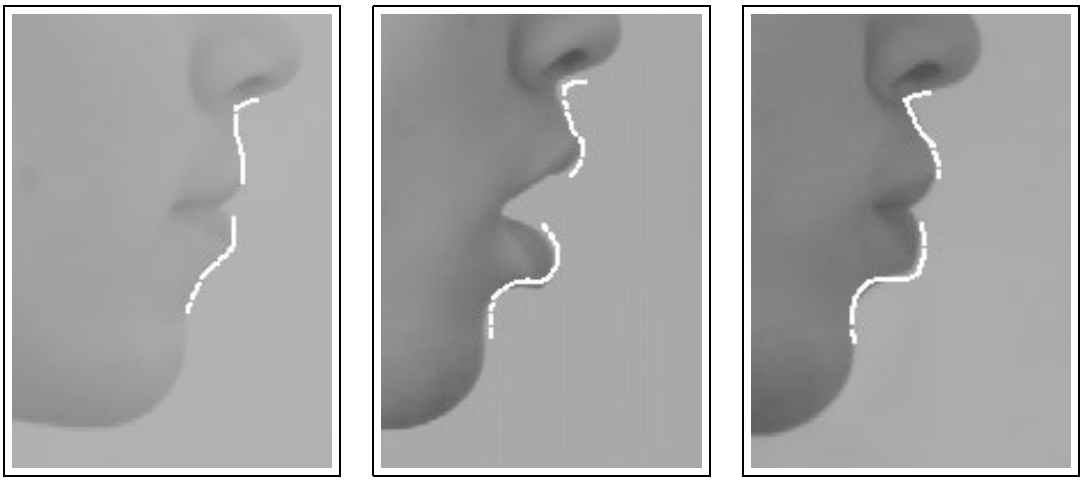
Affine spaces are appropriate shape-spaces for modelling the appearance of three-dimensional rigid body motion. In many applications, for instance facial animation, speech-reading and cardiac ultrasound, as described in chapter 1, motion is decidedly non-rigid. In the absence of any prior analytical description of the motion, the most effective strategy is to *learn* a shape-space from a training set of sample motion. A general approach to this, based on statistical modelling, is described in chapter 8.

In the meantime, a simpler methodology is presented here, based on "key-frames" or representative image outlines of the moving shape. Often, an effective shape-space can be built by linear combination of such key-frames.

As an example, in figure 4.12, a sequence of three frames is shown which can be used to build a simple shape-space in which the first frame $\mathbf{Q}_0$ acts as the template and the shape-matrix W is constructed from the two key-frames $\mathbf{Q}'_1, \mathbf{Q}'_2$:

$$W = \begin{pmatrix} \mathbf{Q}_1^x & \mathbf{Q}_2^x \\ \mathbf{Q}_1^y & \mathbf{Q}_2^y \end{pmatrix}. \tag{4.19}$$

where $\mathbf{Q}_i = \mathbf{Q}'_i - \mathbf{Q}_0$. This two-dimensional shape-space is sufficient to span all linear



Template $\mathbf{Q}_0$ *Key-frame: opening* $\mathbf{Q}_1$ *Key-frame: protrusion* $\mathbf{Q}_2$

Figure 4.12: Key-frames. *Lips template followed by two key-frames, representing interactively tracked lips in characteristic positions. The key-frames are combined linearly with appropriate rigid degrees of freedom, to give a shape-space suitable for use in a tracker for non-rigid motion.*

combinations of the three frames. What is more, the shape-space coordinates have clear interpretations, for example:

- $\mathbf{X} = (0, 0)^T$ represents the closed mouth;

- $\mathbf{X} = (1/2, 0)^T$ represents the half-open mouth;
- $\mathbf{X} = (1/4, 1/2)^T$ represents the mouth, half-protruding and slightly open.

A little more ambitiously, the same three frames can be used to build a more versatile shape-space that allows for translation, zooming and rotation of any of the expressions from the simple two-dimensional shape-space. Minimally, this should require 2 parameters for expression plus 4 for Euclidean similarity, a total of 6 parameters. However, the linearity of shape-space leads to a wastage of 2 degrees of freedom and the shape-space is 8-dimensional with template $\mathbf{Q}_0$ as before and shape-matrix

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y & \mathbf{Q}_1^x & -\mathbf{Q}_1^y & \mathbf{Q}_2^x & -\mathbf{Q}_2^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x & \mathbf{Q}_1^y & \mathbf{Q}_1^x & \mathbf{Q}_2^y & \mathbf{Q}_2^x \end{pmatrix}. \tag{4.20}$$

This is based on the shape-matrix for Euclidean similarities (4.2) on page 75, extended to all three frames. Again, expressions are naturally represented by the shape-vector, for example:

- $\mathbf{X} = (u, 0, 0, 0, 1, 0, 0, 0)^T$ represents the fully open mouth, shifted to the right by u;
- $\mathbf{X} = (0, 0, \cos\theta - 1, \sin\theta, 0, 0, \frac{1}{2}\cos\theta, \frac{1}{2}\sin\theta)^T$ represents the closed mouth, half-protruding and rotated through an angle θ.

Of course this technique, illustrated here for 2 key-frames under Euclidean similarity, does apply to an arbitrary number of key-frames, and a general space of rigid transformations spanned by a set $\{T^j,\ j = 1, \ldots, N_r\}$. In that case any contour corresponding to the appearance of i^{th} key-frame is composed of a linear combination of contours

$$\mathbf{Q}_i^1, \mathbf{Q}_i^2, \ldots, \mathbf{Q}_i^{N_r},$$

for an N_r-dimensional space of rigid transformations. Then the W-matrix is composed of columns which are vectors $\mathbf{Q}_i^j,\ i = 0, 1, \ldots,\ j = 1, 2, \ldots, N_r$. To avoid introducing linear dependencies into the W-matrix, it is best to omit translation from the space of rigid transformations and treat it separately, as in the two key-frame example above. Then the W-matrix for the composite shape-space of rigid and non-rigid transformations is

$$W = \begin{pmatrix} \mathbf{1} & \mathbf{0} & & & & & & \\ & & \mathbf{Q}_0^1 & \mathbf{Q}_0^2 & \ldots & \mathbf{Q}_1^1 & \mathbf{Q}_1^2 & \ldots \\ \mathbf{0} & \mathbf{1} & & & & & & \end{pmatrix}. \tag{4.21}$$

One final caveat is in order. With N_r degrees of transformational freedom (excluding translation) and N_k key-frames, there are a total of $N_r + N_k$ degrees of freedom in the system. However the linear representation as a shape-space with a W-matrix as above has dimension $N_r \times (N_k + 1)$, a "wastage" of $N_k(N_r - 1)$ degrees of freedom. The two key-frame example above has $N_r = 2, N_k = 2$ so the wastage is just 2 degrees of freedom, in a shape-space of total dimension 8 (including translation). With more key-frames and larger transformational spaces such as three-dimensional affine ($N_r = 6$), the wastage is more severe — 5 degrees of freedom per key-frame. In such cases, the constructed shape-space is likely to be too big for efficient or robust contour fitting. However, it is often possible to construct a smaller space by other means such as "PCA" (chapter 8) and use the large shape-space constructed as above for interpretation. In particular, shape displacements can be factored into components due to rigid and non-rigid transformations respectively, and this is explained at the end of chapter 7.

4.8 Articulated motion

When an object (e.g. a hand) is allowed, in addition to its freedom to move rigidly, to support articulated bodies (fingers), more general shape-spaces are needed. Clearly, one route is to take a kinematic model in the style used in robotics for multi-jointed arms and hands and use it as the basis of a configuration space. The advantage is that the resulting configuration space represents legal motions efficiently because the configuration space has minimal dimension. The drawback is that the resulting measurement models (see next chapter) are non-linear. This is due to trigonometric non-linearities as in the previous section on rigid motion but exacerbated by the hinges added onto the base body. The result is that classical linear Kalman filtering is no longer usable, though non-linear variants exist which are not however probabilistically rigorous. Furthermore, linear state-spaces admit motion models which apply globally throughout the space. In a non-linear space, motion models could perhaps be represented as a set of local linear models in tangent spaces placed strategically over a manifold. This is hard enough to represent and the task of learning such models seems a little daunting.

As with rigid motion, there is a way to avoid the non-linearities by generating appropriate shape-spaces. Again, there is some inefficiency in doing this and the resulting space underconstrains the modelled motion. The degree of wastage depends

on the precise nature of the hinging of appendages, and this is summarised in the below. Proofs are not given here, but there is a more detailed discussion in appendix C.

Two-dimensional hinge

For a body in two dimensions, or equivalently a three-dimensional body constrained to lie on a plane, each additional hinged appendage increments the dimension of shape-space by 2, despite adding only one degree of kinematic freedom. Hence the wastage amounts to 1 degree of freedom per appendage.

Two-dimensional telescopic appendage

Still in two dimensions, each telescopic element added to the rigid body increments the shape-space dimension by 2, causing a wastage of one degree of freedom, as for the hinge.

Hinges on a planar body in three dimensions

The rigid planar body above, with its co-planar hinged appendages, is now allowed to move out of the ground plane, so that it can adopt any three-dimensional configuration. Each hinged appendage now adds 4 to the dimension of shape-space, resulting in the wastage of 3 degrees of freedom.

Universal joints on a rigid three-dimensional body

Given a three-dimensional rigid body, whose shape-space is 3D affine, each appendage attached with full rotational freedom (via a ball joint, for instance) increments the dimension of shape-space by 6. Such an appendage introduce 3 kinematic degrees of freedom, so the wastage is 3.

Hinges on a rigid three-dimensional body

For appendages attached to the three-dimensional body by planar hinges, with just 1 kinematic degree of freedom, the dimension of shape-space increases by 4, so again the wastage is 3 degrees of freedom per appendage.

Note that the above results hold regardless of how the appendages are attached — whether directly to the main body (parallel), or in a chain (serial) or a combination of the two.

Bibliographic notes

This chapter has explained how shape-spaces can be constructed for various classes of motion. The value of shape-spaces of modest dimensionality was illustrated in (Blake et al., 1993) as a cure to the instability that can arise in tracking with high-dimensional representations of curves such as the original finite-element snake (Kass et al., 1987) or unconstrained B-splines (Menet et al., 1990; Curwen and Blake, 1992). Shape-spaces are linear, parametric models in image-space, but non-linear models or *deformable templates* are also powerful tools (Fischler and Elschlager, 1973; Yuille, 1990; Yuille and Hallinan, 1992). Linear shape-spaces have been used effectively in recognition (Murase and Nayar, 1995). Shape-spaces discussed so far have been image-based but a related topic is the use of three-dimensional parametric models for tracking, either rigid (Harris, 1992b) or non-rigid (Terzopoulos and Waters, 1990; Terzopoulos and Metaxas, 1991; Lowe, 1991; Rehg and Kanade, 1994).

Initialisation from moments is discussed in (Blake and Marinos, 1990; Wildenberg, 1997) and the use of 3rd moments to recover a full planar affine transformation is described in (Cipolla and Blake, 1992a). In some circumstances, region-based optical flow computation (Buxton and Buxton, 1983; Horn and Schunk, 1981; Nagel, 1983; Horn, 1986; Nagel and Enkelmann, 1986; Enkelmann, 1986; Heeger, 1987; Bulthoff et al., 1989) can be used to define the region for snake initialisation. This has been shown to be particularly effective with traffic surveillance (Koller et al., 1994).

Shape-spaces are based on perspective projection and its linear approximations in terms of vector spaces (Strang, 1986). Mathematically, projective geometry is a somewhat old-fashioned topic and so the standard textbook (Semple and Kneebone, 1952) is rather old-fashioned too. More accessible, is a graphics book such as (Foley et al., 1990) for the basics of camera geometry and perspective transformations. Computer vision has been concerned with *camera calibration* (Tsai, 1987) in which test images of grids are analysed to deduce the projective parameters for a particular camera, including both *extrinsic* parameters (the camera-to-world transformation) and *intrinsic* parameters such as focal length.

The most general linear approximation to perspective is known variously as *paraperspective* (Aloimonos, 1990) or *weak perspective* (Mundy and Zisserman, 1992) and can be particularly effective if separate approximations are constructed for different neighbourhoods of an image (Lawn and Cipolla, 1994). The gamut of possible appearances of three-dimensional contours under a particular weak perspective transformation forms an affine space (Ullman and Basri, 1991; Koenderink and van Doorn,

1991). This idea led to a series of studies on using affine models to analyse motion, including (Harris, 1990; Demey et al., 1992; Bergen et al., 1992a; Reid and Murray, 1993; Bascle and Deriche, 1995; Black and Yacoob, 1995; Ivins and Porrill, 1995; Shapiro et al., 1995).

The first three chapters of (Faugeras, 1993) are an excellent introduction to projective and affine geometry and to camera calibration.

Articulated structures are most naturally described in terms of non-linear kinematics (Craig, 1986) in which the non-linearities arise from the trigonometry of rotary joints. Such a model has been incorporated into a hand tracker, for instance (Rehg and Kanade, 1994), in which the articulation the fingers is full treated. Articulated structures can be embedded in linear shape-spaces but this can be very "inefficient," in the sense of section 4.1, that kinematic constraints have to be relaxed — see appendix C.

Finally, smooth silhouette curves and their shape-spaces are beyond the scope of this book. However, it can be shown that a shape-space of dimension 11 is appropriate. This shape-space representation of the curve is an approximation, valid for sufficiently small changes of viewpoint. Its validity follows from results in the computer vision literature about the projection of silhouettes into images (Giblin and Weiss, 1987; Blake and Cipolla, 1990; Vaillant, 1990; Koenderink, 1990; Cipolla and Blake, 1992b).

Chapter 5

Image processing techniques for feature location

The use of image-filtering operations to highlight image features was illustrated in chapter 2. Figure 2.1 on page 27 illustrated operators for emphasising edges, valleys and ridges, and it was shown how the emphasised image could be used as a landscape for a snake. However, for efficiency, the deformable templates described in the next two chapters are driven towards a distinguished feature curve $\mathbf{r}_f(s)$ rather than over the entire image landscape F that is used in the snake model. This is rather like making a quadratic approximation to the external snake energy:

$$E_{\text{ext}} \propto -F(\mathbf{r}) \propto \int \left(\mathbf{r}(s) - \mathbf{r}_f(s)\right)^2 \, ds, \tag{5.1}$$

where $\mathbf{r}_f(s)$ lies along a ridge of the feature-map function F. The increase in efficiency comes from being able to move directly to the curve $\mathbf{r}_f$, rather than having to iterate towards it as in the original snake algorithm described in section 2.1.

It is therefore necessary to extract $\mathbf{r}_f(s)$ from an image. One way of doing this is to mark high strength values on the feature maps and group them to form point sets to which spline curves could be fitted. An example of feature curves grouped in this way was given, for edge-features, in figure 3.1 on page 42. However, the wholesale application of filters across entire images is excessively computationally costly. At any given instant, an estimate is available of the position of a tracked image-contour and this can be used to define a "search-region," in which the corresponding image feature is likely to lie. Image processing can then effectively be restricted to this search region,

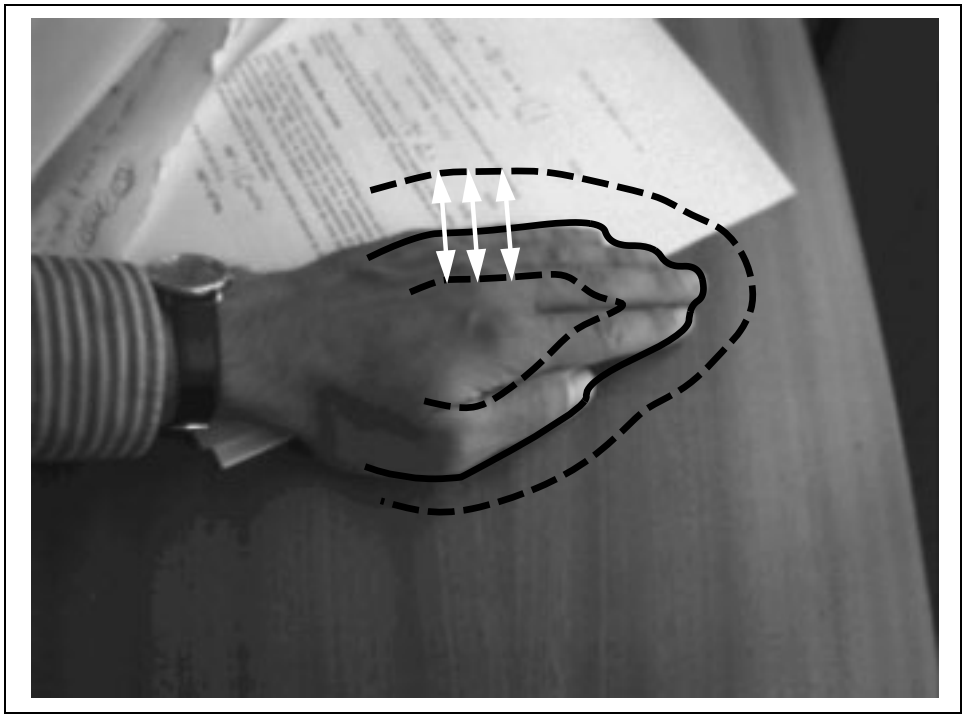


Figure 5.1: Search region. *It is computationally efficient to restrict image processing operations to lie within a "region of interest" (dashed lines), one either side of the currently estimated contour position (solid line). Image processing operations are then performed along certain lines passing through the estimated contour. In this example, the lines are normals to the estimated curve, three of which are shown as arrowed white lines.*

as in figure 5.1. The search region displayed in the figure is formed there by sweeping normal vectors of a chosen length along the entire contour. Features can then be detected by performing image filtering along each of the sampled normals, and this is very efficient. If normals are constructed at points $s = s_i,\ i = 1, \ldots, N$, along the curve $\mathbf{r}(s)$, this will give a sequence of sampled points $\mathbf{r}_f(s_i), i = 1, \ldots, N$ along the feature curve $\mathbf{r}_f(s)$. It is of course possible that more than one feature may be found on each normal, but for now it is assumed that just one — the favourite feature — is retained.

5.1 Linear scanning

In order to perform one-dimensional image processing, image intensity is sampled at regularly spaced intervals along each image normal. An arbitrarily placed normal line generally intersects image pixels in an irregular fashion, as in figure 5.2. This is well

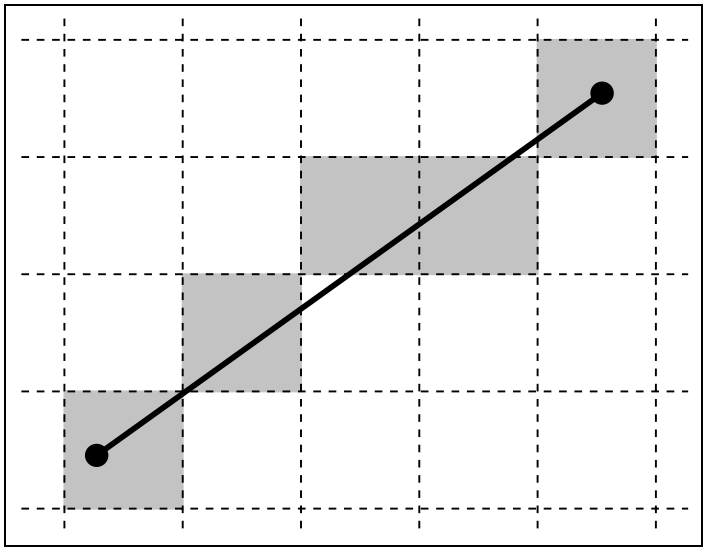


Figure 5.2: Irregular image sampling. *Listing the intensities of pixels crossed by a normal line would result in a non-uniform sampling of intensity that would suffer abrupt variations as the line moved over the image.*

known to produce undesirable artifacts in Computer graphics — "jaggies" in static images and twinkling effects in moving ones, for which the usual cure is "anti-aliasing." Unlike graphics, in which the task is to map from a mathematical line onto pixels, the problem here is to generate the opposite mapping, from image to line. This calls for a sampling scheme of its own.

An effective sampling scheme, spatially regular and temporally smooth (when the line moves) involves interpolation as follows. A sequence of regularly spaced sample points are chosen along the line. The intensity I at a particular sample point (x, y) is computed as a weighted sum of the intensities at 4 neighbouring pixels, as in figure 5.3. A pixel with centre sited at integer coordinates (i, j) has intensity $I_{i,j}$. The intensity

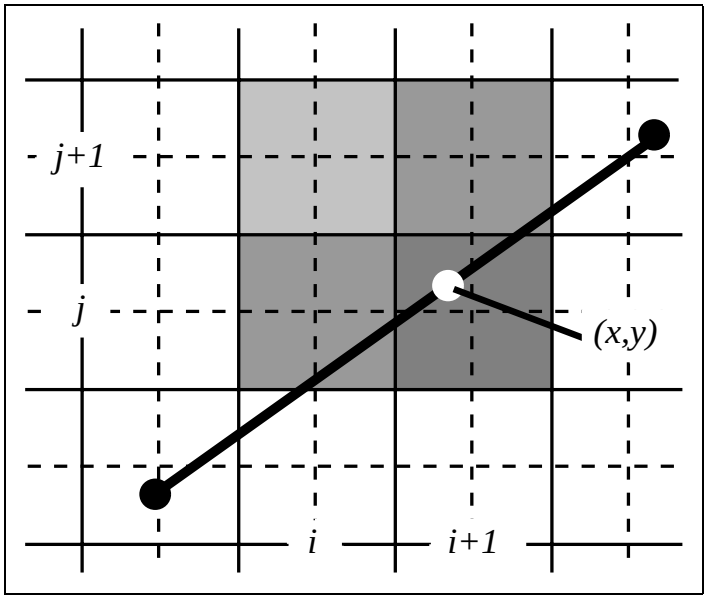


Figure 5.3: Interpolated image sampling. *The intensity at a chosen sampling point* (x, y) *is computed as a weighted sum of the intensities at the four immediately adjacent pixels.*

I at (x, y) is then computed by bilinear interpolation:

$$I = \sum_{i,j} w_{i,j} I_{i,j} \tag{5.2}$$

with weights

$$w_{i,j} = \begin{cases} (1 - |x - i|)(1 - |y - j|) & \text{if } |x - i| < 1 \text{ and } |y - j| < 1 \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

so that at most four pixels, the ones whose centres are closest to (x, y), have non-zero weights, as the figure depicts.

5.2 Image filtering

Analysis of image intensities now concentrates on the one-dimensional signals along normals. The intensity $I(x)$ along a particular normal is sampled regularly at $x = x_i$

and intensities are stored in an array $I_i = I(x_i),\ i = 1, \ldots, N$. A variety of feature detection operators can be applied to the line, popular ones being edges, valleys and ridges. Features are located by applying an appropriate operator or mask $C_n,\ -N_C \leq n \leq N_C$, by discrete convolution, to the sampled intensity signal $I_n,\ 1 \leq n \leq N_I$, to give a feature-strength signal

$$E_n = \sum_{m=-N_C}^{N_C} C_m I_{n+m}.$$

Maxima of that signal are then located, and marked wherever the value at that maximum exceeds a preset threshold (chosen to exclude spurious, noise-generated maxima). This is illustrated for edges in figure 5.4 and for valleys in figure 5.5.

Corners

Effective operators for corners also exist and have been used for visual tracking. However, corners do not quite fit into the search paradigm described here. Being discrete points, a corner is likely to be missed by search along normals unless it happens to lie exactly on some normal; more generally it will be located in the gap between two adjacent normals. This problem does not arise with edges because they are extended and should generally intersect one or more of the normals. If corners are to be used they must be located by an exhaustive search over the region of interest, which is rather more expensive computationally than a search that is restricted to normals.

One operator, the "Harris" corner detector, works by computing a discrete approximation to the moment matrix

$$S(x, y) = \int G(x', y')[\nabla I(x + x', y + y')][\nabla I(x + x', y + y')]^T \, dx' \, dy'$$

at each image point (x, y), where $\nabla I = (\partial I/\partial x, \partial I/\partial y)^T$, the image-gradient vector at a point, and G is a two-dimensional Gaussian mask for smoothing, typically 2–4 pixels in diameter. The trace $\mathrm{tr}(S)$ and the determinant $\det(S)$ are examined at each point (x, y). Wherever $\mathrm{tr}(S)$ exceeds some threshold, signaling a significantly large image gradient, and also the ratio $\mathrm{tr}(S)/2\sqrt{\det(S)}$ is sufficiently close to its lower bound of 1, a corner feature is marked. The Harris detector responds reliably to objects that are largely polyhedral, marking corners that are likely to be stable to changing viewpoint. Natural shapes (figure 5.6) may however fire the corner detector in locations that are

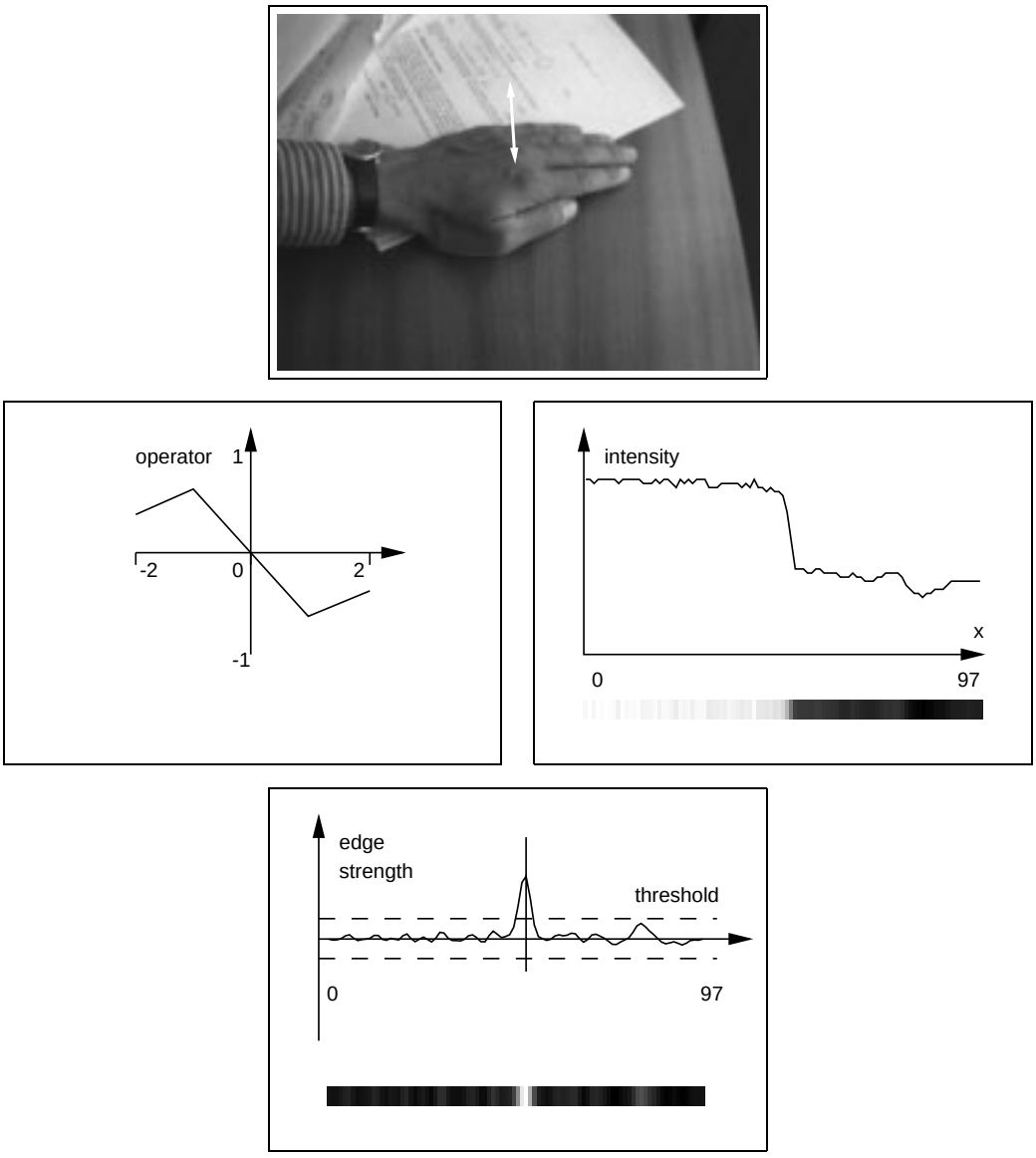


Figure 5.4: Operator for edge detection *The problem is to search along a line in an image (top) to find edges — locations where contrast is high. An operator (left, shown on an expanded length scale) is convolved with the image intensity function along the line (right). One edge is found, corresponding to a maximum of the feature-strength function (bottom).*

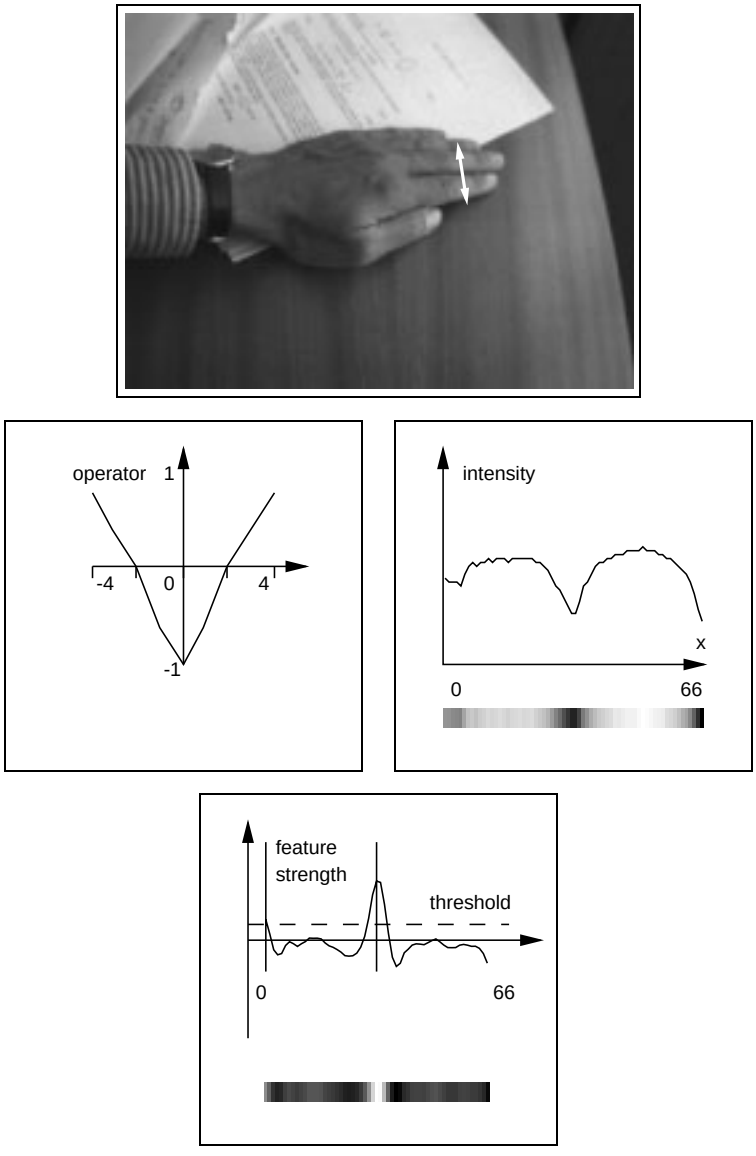


Figure 5.5: Operator for valley detection *The problem is to search along a line in an image (top) to find valleys — locations of minimum intensity. The valley operator (left) convolved with the intensity signal (right) produces a ridge (bottom) corresponding to the dark line between adjacent fingers.*

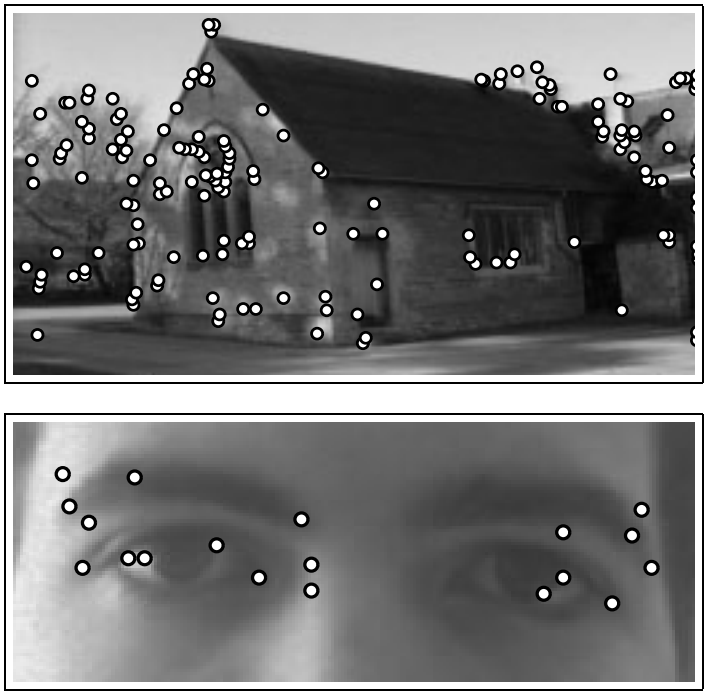


Figure 5.6: Corner detection. *The corner detector fires reliably at sharp, polyhedral corners and junctions. It also fires on certain other features such as tighter curves, but with less spatial accuracy and reliability. (Figures courtesy of Andrew Fitzgibbon).*

hard to predict and may be unstable under changing viewpoint. Not surprisingly, it works best where there are well-defined geometrical features, such as on buildings. On a natural object such as a face, the response is a mixture of reasonably reliable features at eye corners and other less reliable responses on curves.

5.3 Using colour

Colour in images is a valuable source of information especially where contrast is weak, as in discriminating lips from facial skin. Colour information is commonly presented as a vector $\mathbf{I} = (r, g, b)$ of red-green-blue values, at each pixel. The most economical way to treat the colour vector is to reduce it to a single value by computing a suitable scalar function of $\mathbf{I}$. The scalar I can then be treated as a single intensity value

and subjected to the same image-feature operators as were used above, for processing monochrome images. Two scalar functions are described here: one that is general, the *hue* function and one that is customised — learned from training data, the *Fisher linear discriminant*.

The hue function is used commonly to represent colour in graphics, vision and colorimetry and corresponds roughly to the polar angle for polar coordinates in r, g, b-space. It separates out what is roughly a correlate of spectral colour — i.e. colour wavelength — from two other colour components: intensity (overall brightness) and saturation ("colouredness" as opposed to whiteness). This explains why it is particularly effective when contrast is low so that changes in intensity $(r + g + b)$ are hard to perceive. Hue is defined as follows

$$\text{hue}(r, g, b) = \arctan\left((2r - g - b), \sqrt{3}(g - b)\right). \tag{5.4}$$

Note that there is a linear "hexcone" approximation to the hue function for applications where it is important to compute it fast, for example to convert entire video images.

The Fisher linear discriminant is an alternative scalar function that attempts to represent as much of the relevant colour information as possible in a single scalar value. Its efficiency is optimal in certain sense, but this comes at the cost of re-deriving the function, in a learning step, for each new application. Learning is straightforward and works as follows. First, a foreground area F and a background area B are delineated in a training image. For instance, in figure 5.7, F would be the lip region and B would be part of the immediate surround. The Fisher discriminant function is defined as a simple scalar product:

$$\text{fisher}(\mathbf{I}) = \mathbf{f} \cdot \mathbf{I} \quad \text{where} \quad \mathbf{I} = (r, g, b)^T. \tag{5.5}$$

The function is learned from the foreground and background areas F and B by the algorithm of figure 5.8, which determines the coefficient vector $\mathbf{f}$. The effect of the algorithm is to choose the vector $\mathbf{f}$ in colour (r, g, b) space which best separates the background and foreground populations of colours. When the Fisher function is used in place of intensity, feature detection works effectively, as figure 5.7 shows.

5.4 Correlation matching

The oldest idea in visual matching and tracking, and one that is widely used in practical tracking systems, is correlation. Given a template T in the form of a small

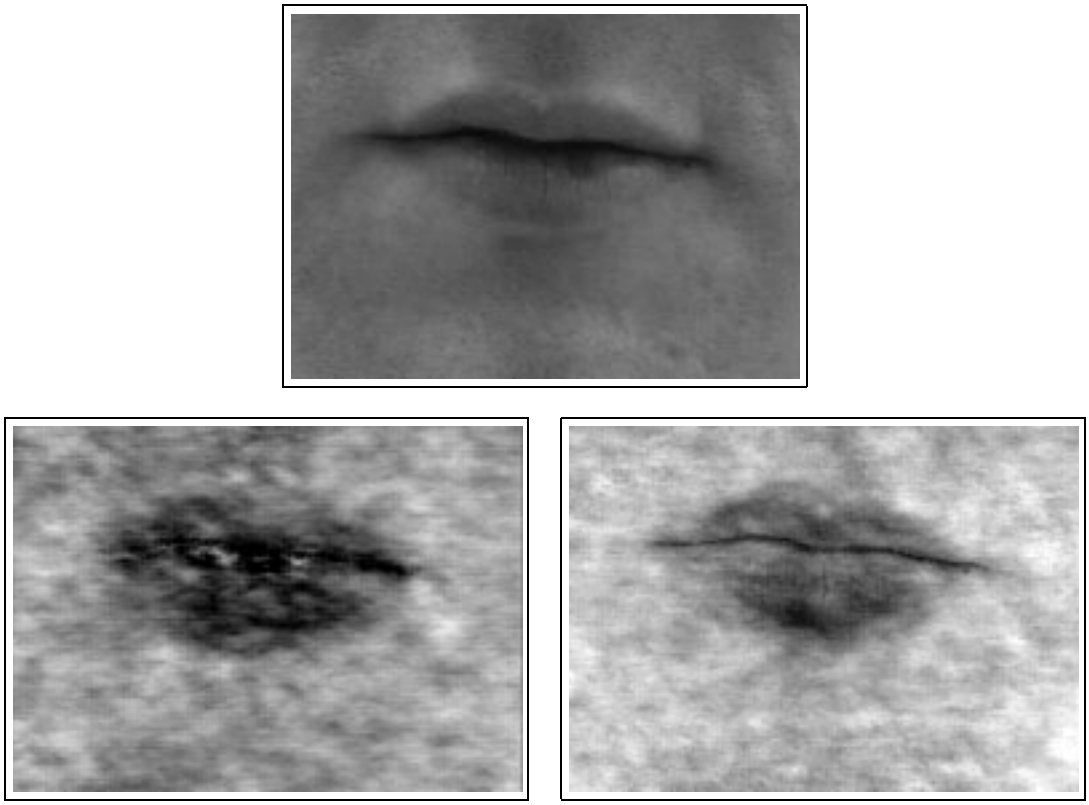


Figure 5.7: Detecting colour boundaries. *Contrast between lip and skin can be lost if only intensity (top) is extracted from a colour image; this is especially so for the lower lip in this example. The hue function (left) shows improved contrast, but is rather noisy, and this tends to disturb the operation of an active contour by generating spurious clutter. A better solution is the Fisher discriminant function (right). In this example, colour is to be used to locate lip boundaries. A colour training image is used to learn the Fisher discriminant function. (Figures courtesy of Robert Kaucic.)*

array of image intensities, the aim is to find the likely locations of that template in some larger test image I. In one dimension for example, with image $I(x)$ and template $T(x)$, $0 \leq x \leq \Delta$, the problem is to find the offset x' for which $T(x)$ best matches $I(x + x')$ over the range $0 \leq x \leq \Delta$ of the template. This is most naturally done by

1. **Calculate** the mean pixel values in each class

$$\overline{I}_{\mathrm{F}} = \frac{1}{N_{\mathrm{F}}} \sum_{(x,y)\in \mathrm{F}} \mathbf{I}(x,y)$$
$$\overline{I}_{\mathrm{B}} = \frac{1}{N_{\mathrm{B}}} \sum_{(x,y)\in \mathrm{B}} \mathbf{I}(x,y)$$

2. **Determine** the within class scatter matrices

$$S_{\mathrm{F}} = \sum_{(x,y)\in \mathrm{F}} (\mathbf{I}(x,y) - \overline{\mathbf{I}}_{\mathrm{F}})(\mathbf{I}(x,y) - \overline{\mathbf{I}}_{\mathrm{F}})^T$$
$$S_{\mathrm{B}} = \sum_{(x,y)\in \mathrm{B}} (\mathbf{I}(x,y) - \overline{\mathbf{I}}_{\mathrm{B}})(\mathbf{I}(x,y) - \overline{\mathbf{I}}_{\mathrm{B}})^T$$

3. **Find** the Fisher discriminant vector

$$\mathbf{f} = S^{-1}(\overline{\mathbf{I}}_{\mathrm{F}} - \overline{\mathbf{I}}_{\mathrm{B}}) \quad \text{where} \quad S = S_{\mathrm{F}} + S_{\mathrm{B}}.$$

Figure 5.8: Learning the Fisher discriminant function. *A discriminant vector* $\mathbf{f}$ *is computed from the foreground* F *and background* B *populations of pixels, aiming to separate foreground optimally from background.*

minimising a difference measure such as

$$M(x') = \int_{x=0}^{\Delta} (I(x+x') - T(x))^2 \, dx \tag{5.6}$$

with respect to x'. In practice this theoretical measure must be computed as a sum over image pixels.

An illustration is given in figure 5.9 in which the problem is to locate the position of the eyebrow along a particular line. A template T is available that represents an ideal distribution of intensity along a cross-section located in a standard position. The task is to find the position on the line which best corresponds to the given intensity

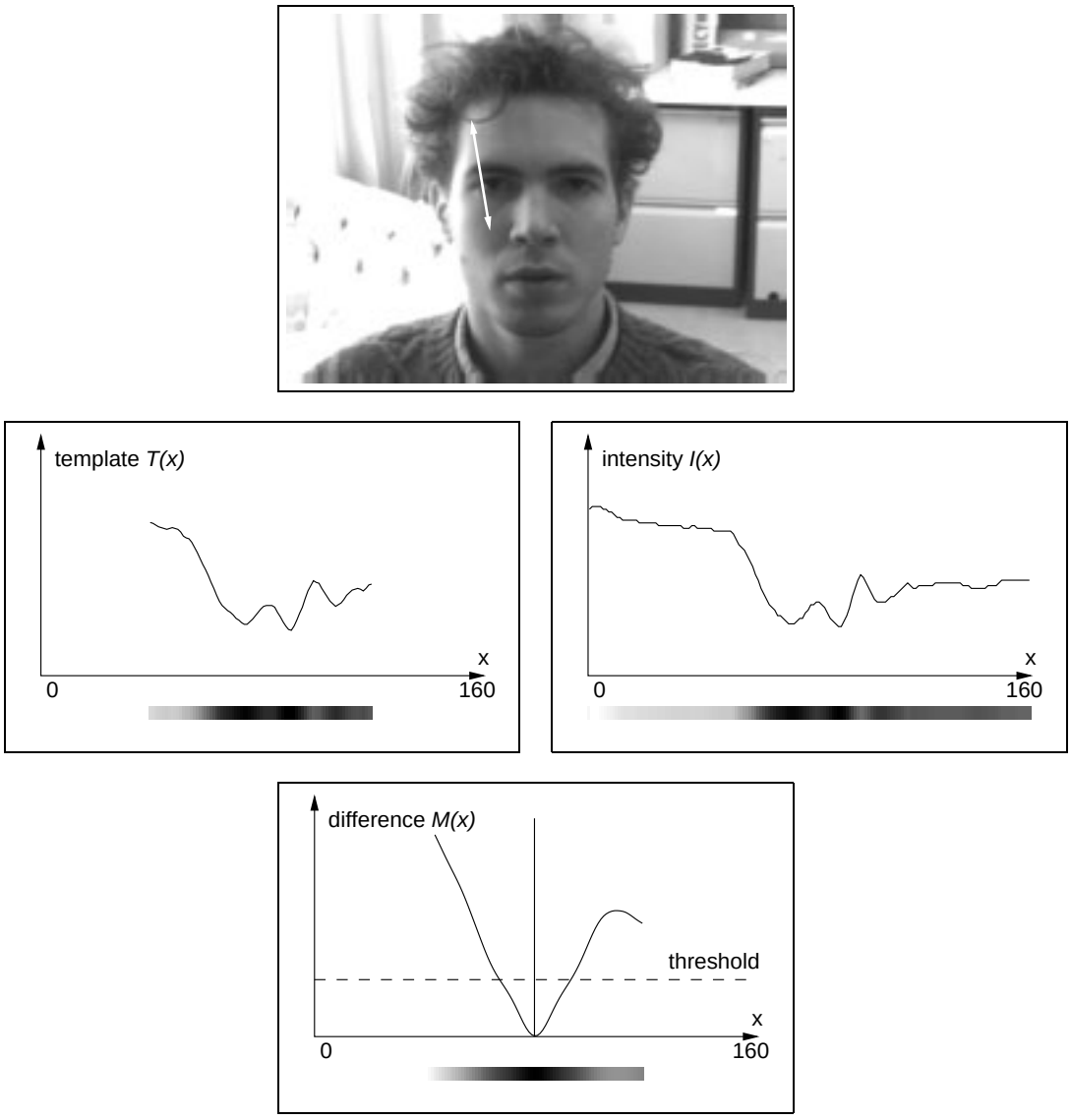


Figure 5.9: One-dimensional correlation matching. *The problem is to search along a line in an image (top) to find the position of an eye. A template (left) of the cross-section of an eye is to be matched with the image intensity function along the line (right). The correct position is marked by the minimum of the matching function (bottom). Figure courtesy of Robert Kaucic.*

template, and this is achieved by minimising $M(x)$ as above. The nomenclature "correlation" derives from the idea that in the case that $T(x)$ and $I(x)$ span the same x-interval, and are periodic, minimising $M(x)$ is equivalent to maximising the "mathematical correlation"

$$\int_{x=0}^{\Delta} I(x+x')T(x)\,dx. \tag{5.7}$$

Correlation matching can be used to considerable effect as a generalised substitute for edge and valley location. Position of the located feature along each normal can be reported in the same way as for edges and used for curve matching. This can be particularly effective in problems where image contrast is poor. In the lip-tracking application of figure 1.10 on page 14, tracking without lip make-up proved possible only when edge detection was replaced by correlation.

There are numerous variations on the basic theme (see bibliographic notes, at the end of the chapter). One variation is to pre-process $I(x)$, for example to emphasise its spatial derivative, which tends to generate a sharper valley in $M(x')$, which can then be located more accurately. More modern approaches dispense with intensities altogether, representing I and T simply as lists of the positions of prominent features. The problem then is to match those lists, using discrete algorithms. This has the advantage of efficiency because of the data-compression involved in reducing the intensity arrays to lists. It is also more robust for two reasons. First, intensity profiles vary as ambient illumination changes whereas the locations of features are approximately invariant to illumination. Secondly, there is the additional flexibility that different amounts of offset x' can be associated with different features, for example when performing stereo matching over a large image region, whereas in the correlation framework x' is fixed. For these reasons, feature-based matching is considered superior to correlation for many problems.

Correlation matching is often used in two dimensions with a template $T(x, y)$ matched to an offset image $I(x + x', y + y')$, as in figure 5.10. This is considerably more computation-intensive than in one dimension as it involves a double integral over x, y and also a two-dimensional search to minimise M with respect to x', y'. Various techniques such as Discrete Fourier Transforms and "pyramid" processing at multiple spatial scales can be used to improve efficiency. In higher dimensions than two, for example when rotation and scaling are to be allowed in addition to translation, exhaustive correlation becomes prohibitively expensive and alternative algorithms are needed. One approach is to generate the offsets in higher dimensions in a more sparing

Figure 5.10: Matching eyes by image correlation. *The template (top) is a reversed copy of the right eye in the face image (bottom). When the template is correlated with the image, its centre collocates accurately with the centre of the left eye. (Figures courtesy of Steve Smith.)*

fashion, using gradient descent for instance. Numerous authors have shown that this can be very successful.

5.5 Background subtraction

A widely used technique for separating moving objects from their backgrounds is based on subtraction. It is used as a pre-process in advance of feature detection to suppress background features to prevent them distracting fitting and tracking processes. It is particularly suited for applications such as surveillance where the background is often largely stationary.

An image of $I_B(x, y)$ of the background is stored before the introduction of a foreground object. Then, given an image $I(x, y)$ captured with the object present, feature detection is restricted to areas of $I(x, y)$ that are labelled as foreground because

they satisfy

$$|I(x,y) - I_B(x,y)| > \sigma,$$

where σ is a suitable chosen noise threshold. As figure 5.11 shows, background features tend to be successfully inhibited by this procedure. Cancellation can disrupt the

Background $I_B(x,y)$

Image $I(x,y)$

Background suppressed

Figure 5.11: Background subtraction. *The difference between an image (right) and a stored background (left) is computed to suppress background features (bottom), though some background features do "print through" the foreground.*

foreground, as the figure shows, where the background intensity happens to match

the foreground too closely. This results in some loss of genuine foreground features, a cost which is eminently justified by the effectiveness of background suppression.

Finally, it should be noted that the expense of computing the entire difference image ΔI can be largely saved by computing differences "lazily" just of those pixels actually required for interpolation along normals in (5.2). This is an important consideration for real-time tracking systems.

Bibliographic notes

The Bresenham algorithm (Foley et al., 1990) is routinely used in graphics to convert a mathematical line to a sequence of pixels and "anti-aliasing" is employed to achieve an interpolated pattern of pixel intensities which varies smoothly, without flicker, as the line is moved. The interpolated sampling scheme described in this chapter is similar in spirit, but uses a different sampling pattern which performs the inverse function of mapping from an array of pixels to an arbitrary point on a mathematical line.

A general reference on feature detection and the use of convolution masks is (Ballard and Brown, 1982). A much-consulted study of trade-offs in the design of operators for edge detection is (Canny, 1986) and the design of operators for ridges and valleys is described in (Haralick, 1980); the discussions relate to two-dimensional image processing whereas in this chapter the simpler one-dimensional problem is addressed. Effective detectors for corners exist (Kitchen and Rosenfeld, 1982; Zuniga and Haralick, 1983; Noble, 1988) and have been used to good effect in motion analysis, e.g. (Harris, 1992a) and tracking (Reid and Murray, 1996). Operators that respond to regions rather than curves are also important, for example texture masks (Jain and Farrokhnia, 1991) which can be used effectively in snakes that settle on texture boundaries (Ivins and Porrill, 1995).

Correlation matching is based on the idea of "mathematical correlation" which is central to the processing of one-dimensional signals (Bracewell, 1978). It is also used in two-dimensional processing of images to locate patterns (Ballard and Brown, 1982), track motion (Bergen et al., 1992b) and register images for stereo vision (Lucas and Kanade, 1981). Two-dimensional correlation can be computed efficiently using pyramid architectures (Burt, 1983). A notable variation on the correlation approach is to allow the correlation offset to vary spatially, adding considerable flexibility at some computational expense (Witkin et al., 1986). Successful applications of correlation in higher dimensions have used gradient descent (Sullivan, 1992) which may also be

coupled with feature detection (Bascle and Deriche, 1995) for added robustness to illumination variations and specularity. Very efficient algorithms can be constructed in the case of affine shape-spaces, by pre-processing image deformation maps (Hager and Belhumeur, 1996). One-dimensional correlation along normals has proved a useful tool in matching contours (Cootes et al., 1993; Rowe and Blake, 1996a).

The hue model for colour is used in vision (Ballard and Brown, 1982) and in graphics (Foley et al., 1990) in the form of a linear "hexcone" approximation which can be computed efficiently. The Fisher discriminant function is a general technique in pattern recognition (Bishop, 1995) that has also been used to good effect in vision to discriminate faces from one another (Belhumeur et al., 1996). A Bayesian treatment of colour segmentation for tracking is given in (Crisman, 1992). With careful modelling of the physics of reflection, colour segmentation can be even made robust in an outdoor environment with its varying illumination (Plá et al., 1993).

Background subtraction/cancellation assists greatly in the generation of benign training sets, by suppressing clutter, and is a valuable technique in learning dynamics (chapter 11). A variety of techniques exists based on linear and non-linear (morphological) filtering, and on statistical hypothesis testing (Baumberg and Hogg, 1994; Murray and Basu, 1994; Koller et al., 1994; Rowe and Blake, 1996b).

Chapter 6

Fitting spline templates

Chapters 3 and 4 dealt with the geometry and representation of curves and classes of curves — the shape-spaces. Now it is time to look at some image data to see how shapes can be approximated by members of those classes. The norm and inner product machinery developed earlier proves useful together with the image-processing techniques of chapter 5. Curve approximation techniques are built up step by step in this chapter until the necessary tools are assembled for basic B-spline snakes and deformable templates.

6.1 Regularised matching

Generally, measurements made from images are "noisy" — prone to unpredictable variations from a number of sources. At the finest grain there is the effect of electrical noise on the video signal from the camera and optical noise such as the flickering of fluorescent lights. Coarser effects are the interactions of lighting with object surfaces, causing specularities or highlights and shadows which vary in a manner that is too hard to model. Since we have to live with such disturbances it is imperative that algorithms for analysing images are designed to be intrinsically robust to them. This is standard practice in image processing, where "regularisation" is used to clean poor quality images by imposing prior constraints on the likely appearance of valid images. This section describes the application of regularisation to curves reconstructed from image data. First basic curve-fitting machinery is developed.

Projection

Shape-space was introduced as a means of reducing shape-variability and, in the context of the problem of fitting an image-feature curve, acts as a way of encouraging smoothness. Suppose the image feature were expressed in the form of a spline curve $\mathbf{r}_f$ where $\mathbf{r}_f(s) = U(s)\mathbf{Q}_f$. If the fitted spline $\mathbf{Q}$ is restricted to shape-space, and using the energy landscape for a quadratically approximated feature map ((5.1) on page 97), the fitting problem is

$$\min_{\mathbf{X}} \quad \|W\mathbf{X} + \mathbf{Q}_0 - \mathbf{Q}_f\|^2.$$

The solution $\mathbf{X} = \hat{\mathbf{X}}$ is given (proof below) by the pseudo-inverse $W^+ = \mathcal{H}^{-1}W^T\mathcal{U}$ that was defined in the previous chapter:

$$\hat{\mathbf{X}} = W^+(\mathbf{Q}_f - \mathbf{Q}_0). \tag{6.1}$$

The fact that $\hat{\mathbf{X}}$ minimises the error in approximating the curve $\mathbf{Q}$ within shape-space $\mathcal{S}$ motivates the interpretation of W^+ as an operator for *projection* onto shape-space. It also explains why, in the example of figure 6.1, this operator manages to smooth noisy data while staying close to the gross shape of that data.

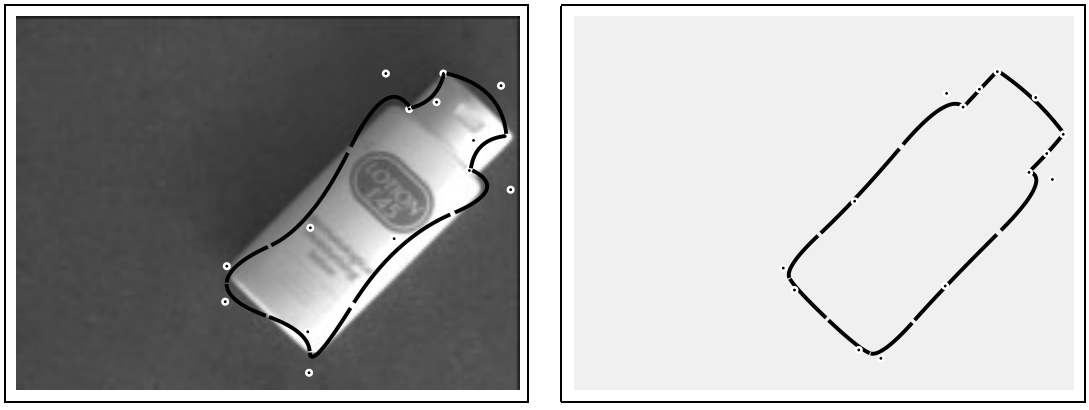


Figure 6.1: Curve approximation in shape-space. A distorted bottle shape (left, black curve) does not fit the bottle outline at all well. Projecting the distorted curve onto the lotion bottle shape-space of figure 4.5 on page 77, finds the closest "valid" curve shape (right). Clearly this removes the distortion.

Regularisation

Further tolerance to noise is procured by biasing the fitted curve towards a mean shape $\overline{\mathbf{r}}(s)$ to a degree determined by a regularisation constant α. In the simplest form of the problem, the fitted curve is the solution of

$$\min_{\mathbf{r}(s)} \quad \alpha\|\mathbf{r}-\overline{\mathbf{r}}\|^2 + \|\mathbf{r}-\mathbf{r}_f\|^2 \tag{6.2}$$

where $\mathbf{r}_f$ is a member of a class $\mathcal{S}_Q$ of B-spline curves, and the possible fitted curves $\mathbf{r}$ are constrained to lie in some shape-space $\mathcal{S} \subset \mathcal{S}_Q$. The problem can be expressed conveniently as

$$\min_{\mathbf{X}} \quad \alpha\|\mathbf{X}-\overline{\mathbf{X}}\|^2 + \|\mathbf{Q}-\mathbf{Q}_f\|^2 \quad \text{with} \quad \mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0.$$

The idea of the regularising term is that it tends to pull any fitted curve towards the mean $\overline{\mathbf{r}}$ but this is rarely satisfactory as it stands. For example, it may be desirable in practice for $\overline{\mathbf{r}}$ to influence the *shape* of the fitted curve but not its position or orientation. A more general regulariser is needed therefore, using a weight matrix $\overline{S}$ (which must be positive semi-definite), so that the fitting problem becomes:

$$\min_{\mathbf{X}} \quad (\mathbf{X}-\overline{\mathbf{X}})^T\overline{S}(\mathbf{X}-\overline{\mathbf{X}}) + \|\mathbf{Q}-\mathbf{Q}_f\|^2 \quad \text{with} \quad \mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0. \tag{6.3}$$

For instance, the regulariser $\alpha\|\mathbf{X}-\overline{\mathbf{X}}\|^2$ would be obtained by setting $\overline{S} = \alpha\mathcal{H}$. To achieve the desired invariance of the regulariser over some subspace $\mathcal{S}_s \subset \mathcal{S}$ of transformations, for instance the Euclidean similarities, $\overline{S}$ must be restricted by means of a projection operation E^d to operate over deformations outside the invariant subspace $\mathcal{S}_s$:

$$\overline{S} = \alpha {E^d}^T \mathcal{H} E^d. \tag{6.4}$$

The projection operator can be expressed in terms of the shape-matrix W_s for the subspace and its pseudo-inverse shape-matrix W_s^+:

$$E^d = I - E^s \quad \text{where} \quad E^s = W^+ W_s W_s^+ W. \tag{6.5}$$

The solution $\mathbf{X} = \hat{\mathbf{X}}$ to the fitting problem (6.3) is obtained in two stages, a projection onto shape-space

$$\mathbf{X}_f = W^+(\mathbf{Q}_f - \mathbf{Q}_0), \tag{6.6}$$

followed by weighted summation

$$\hat{\mathbf{X}} = \left(\overline{S} + \mathcal{H}\right)^{-1} \left(\overline{S}\,\overline{\mathbf{X}} + \mathcal{H}\mathbf{X}_f\right). \tag{6.7}$$

This is illustrated in the example of figure 6.2, in which the shape-space $\mathcal{S}$ is taken to be the entire spline space ($\mathcal{S} = \mathcal{S}_Q$) and the invariant subspace $\mathcal{S}_s$ is the space of Euclidean similarities which allows the *shape* of $\overline{\mathbf{X}}$ to influence the fit while its position and orientation are ignored. As $\alpha \to 0$, the influence of the template diminishes and the fitted curve moves closer to the data, as expected.

Proof that projection is realised by the pseudo-inverse. Writing $\mathbf{Q}' = \mathbf{Q}_f - \mathbf{Q}_0$, the fitting problem above is to minimise

$$\begin{aligned} \|W\mathbf{X} - \mathbf{Q}'\|^2 &= (W\mathbf{X} - \mathbf{Q}')^T\mathcal{U}(W\mathbf{X} - \mathbf{Q}') \\ &= \mathbf{X}^T W^T\mathcal{U}W\mathbf{X} - \mathbf{X}^T W^T\mathcal{U}\mathbf{Q}' - \mathbf{Q}'^T\mathcal{U}W\mathbf{X} + \text{ const} \\ &= (\mathbf{X} - \hat{\mathbf{X}})^T W^T\mathcal{U}W(\mathbf{X} - \hat{\mathbf{X}}) + \text{ const}, \end{aligned}$$

(completing the square) in which

$$\hat{\mathbf{X}} = (W^T\mathcal{U}W)^{-1}W^T\mathcal{U}\mathbf{Q}' = \mathcal{H}^{-1}W^T\mathcal{U}\mathbf{Q}',$$

which gives the required result (6.1).

Derivation of the shape-space regularisation formula. First a "projection lemma" is needed, essentially Pythagoras' theorem applied to shape-space, that

$$\|\mathbf{Q} - \mathbf{Q}_f\|^2 = \|\mathbf{X} - \mathbf{X}_f\|^2 + \|W\mathbf{X}_f + \mathbf{Q}_0 - \mathbf{Q}_f\|^2, \tag{6.8}$$

and this is illustrated in figure 6.3. (Note that $\|W\mathbf{X} - W\mathbf{X}_f\| = \|\mathbf{X} - \mathbf{X}_f\|$ by definition of the norm over $\mathcal{S}$). Now the problem of (6.3) becomes the minimisation of

$$(\mathbf{X} - \overline{\mathbf{X}})^T\overline{S}(\mathbf{X} - \overline{\mathbf{X}}) + (\mathbf{X} - \mathbf{X}_f)^T\mathcal{H}(\mathbf{X} - \mathbf{X}_f)$$

and this is simplified by "completing the square" to give

$$(\mathbf{X} - \hat{\mathbf{X}})^T(\overline{S} + \mathcal{H})(\mathbf{X} - \hat{\mathbf{X}}) + c$$

where $\hat{\mathbf{X}}$ is as defined above and c is a constant independent of $\mathbf{X}$, so that the minimising shape is $\mathbf{X} = \hat{\mathbf{X}}$ as in 6.7.

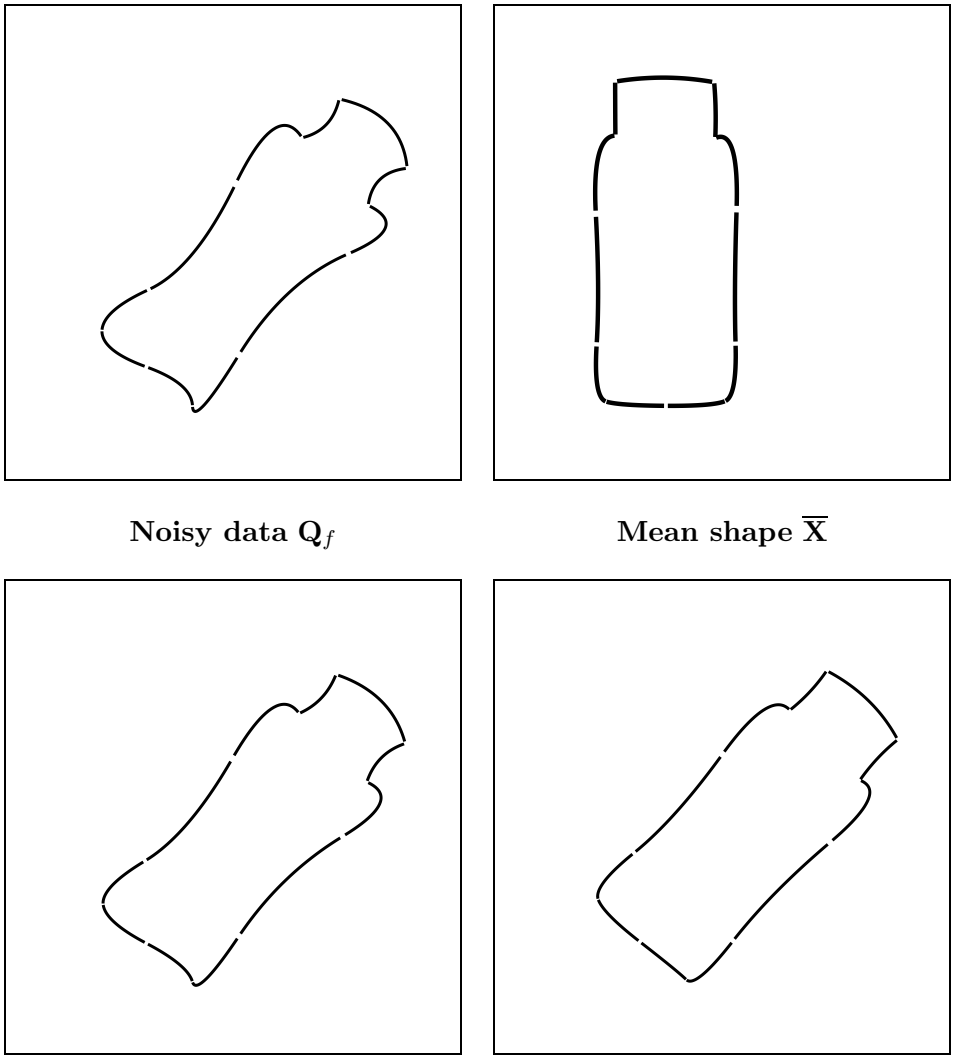


Figure 6.2: Curve-fitting with regularisation. *A regularisation parameter α controls the trade-off from high noise resistance but biased towards a mean shape (α large) to more accurate fitting but with greater sensitivity to noisy data (alpha small). The regulariser is constructed to be invariant to Euclidean similarity transformations, so the position and orientation of the mean shape have no influence on the fitted curve.*

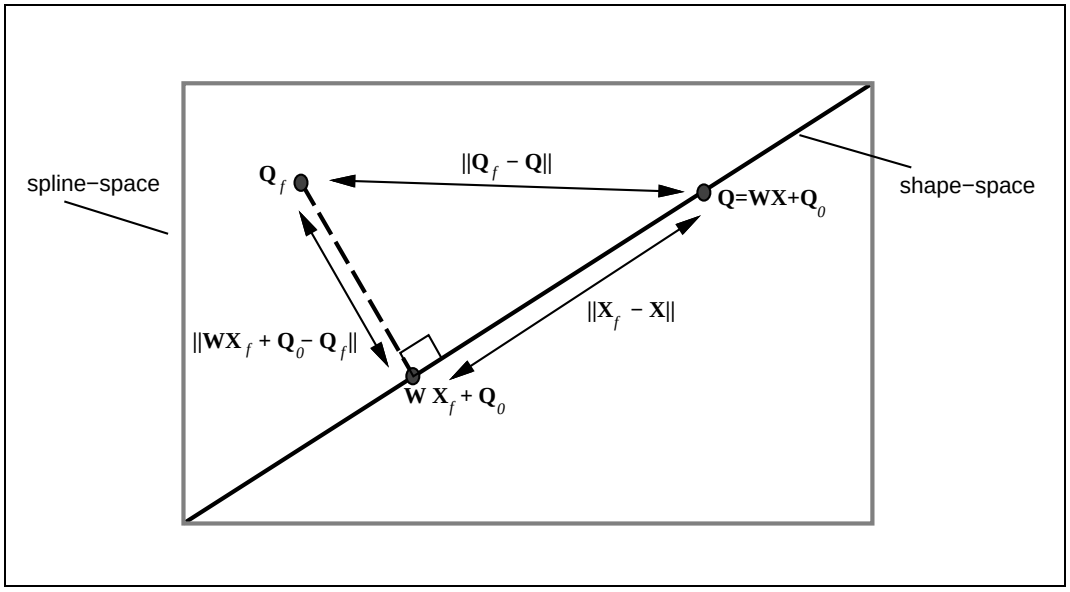


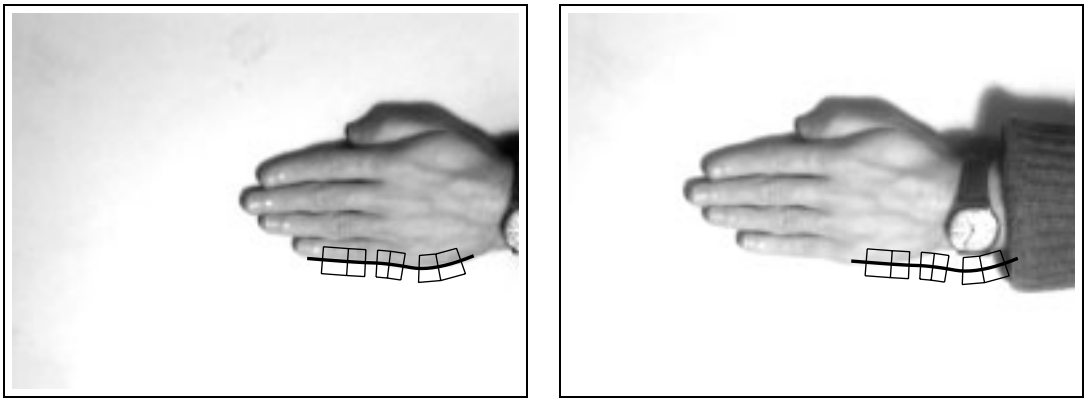
Figure 6.3: *Projection lemma.*

6.2 Normal displacement in curve fitting

In chapter 3 we saw that the measure of curve difference using the norm is sensitive to parameterisation. Two curves with similar shapes will nonetheless register a substantial difference according to the norm unless the parameterisations of those curves also match. One example, figure 3.16 on page 62, showed two similarly shaped curves with a norm-difference that was substantial simply because the parameterisation of one of the curves had been shifted.

Curve fitting based on the norm will therefore suffer from sensitivity to parameterisation. This is particularly a problem when the data-curve is derived from an image because, as the previous section showed, the parameterisation of an image curve is inherited from a spline curve $\mathbf{r}(s)$, either the template curve itself or some initial estimate of the image curve. This anchors the parameterisation of the image curve $\mathbf{r}_f(s)$ close to that of $\mathbf{r}(s)$. The result is that the fitted curve exhibits "reluctance" to move away from $\mathbf{r}(s)$, as figure 6.4 illustrates.

A solution to this problem, proposed in chapter 3, is to redefine the fitting problem to take re-parameterisation explicitly into account when measuring the difference



(a) (b)

Figure 6.4: Inherited parameterisation leads to reluctance in curve fitting. *Curve fitting using norm-difference is applied repeatedly to a sequence of images starting with (a) and ending with (b). In each case, the fitted curve obtained from the previous image is used as the initial estimate for fitting in the current image. At each step, the inherited parameterisation of the feature curve follows closely the parameterisation of the estimated curve. The result is a reluctance to move, as illustrated.*

between the fitted curve $\mathbf{r}$ and the data $\mathbf{r}_f$. A re-parameterisation function $g(s)$ is defined such that $g(s) = s$ gives the original inherited parameterisation, mapping points on curve $\mathbf{r}$ to points on data-curve $\mathbf{r}_f$. Then the image-data curve $\mathbf{r}_f(s)$ is to be compared with the re-parameterised version of the curve $\mathbf{r}(g(s))$. In place of $\|\mathbf{r} - \mathbf{r}_f\|$ in (6.2), the alternative curve-displacement measure $d(\mathbf{r}, \mathbf{r}_f)$ is:

$$d^2 = \min_g \frac{1}{L} \int D^2 \, ds \quad \text{where} \quad D^2(s) = (\mathbf{r}(g(s)) - \mathbf{r}_f(s))^2, \tag{6.9}$$

defined as a minimum over all possible reparameterisations.

Local invariance

Computation of the curve-displacement measure $d(\mathbf{r}, \mathbf{r}_f)$ is feasible if we are content with local, rather than global minimisation. According to the rules of the "calculus of variations", a local minimum of d is achieved when

$$\frac{\partial D^2}{\partial g} = 0 \quad \text{for all } s,$$

that is, when

$$[\mathbf{r}_f(s) - \mathbf{r}(g(s))] \cdot \mathbf{r}'(g(s)) = 0.$$

This says that, for the optimal parameterisation, the vector $\mathbf{r}_f(s) - \mathbf{r}(g(s))$ joining corresponding points on the two curves is perpendicular to the tangent vector $\mathbf{r}'(g(s))$; it is in the direction of the *normal* vector $\mathbf{n}(g(s))$ to the curve $\mathbf{r}$, so that the distance between corresponding points is

$$D(s) = |\mathbf{r}_f(s) - \mathbf{r}(g(s))| = [\mathbf{r}_f(s) - \mathbf{r}(g(s))] \cdot \mathbf{n}(g(s)).$$

Now, assuming that the curves are sufficiently similar that the extent of re-parameterisation is small ($g(s) \approx s$), it follows first that

$$\mathbf{n}(g(s)) \approx \mathbf{n}(s)$$

since the implicit smoothness of the B-spline ensures the curvature is also small. In addition, using a first-order Taylor expansion,

$$[\mathbf{r}(s) - \mathbf{r}(g(s))] \cdot \mathbf{n}(s) \approx (g(s) - s)\mathbf{r}'(s) \cdot \mathbf{n}(s) = 0,$$

giving finally

$$D(s) \approx [\mathbf{r}_f(s) - \mathbf{r}(s)] \cdot \mathbf{n}(s),$$

the "normal displacement" between corresponding points on the two curves (with the original parameterisation).

The use of normal displacement, which is a standard technique from Computer Vision, can be explained intuitively. The total displacement $\mathbf{r}_f(s) - \mathbf{r}(s)$ at a point can be expressed as the vector sum of components along the curve tangent and normal respectively (figure 6.5). The tangential component corresponds approximately to displacement *along* the curve $\mathbf{r}(s)$ without actually travelling any distance away from the curve. It reflects the variation of parameterisation between curves. If the tangential component is eliminated, what remains of the total displacement is the normal component, representing purely the distance between curves.

In terms of the original problem of finding a measure of distance between the fitting curve $\mathbf{r}(s)$ and a feature curve $\mathbf{r}_f(s)$, we have shown that the distance

$$d(\mathbf{r}, \mathbf{r}_f) \approx \frac{1}{L} \int [(\mathbf{r}(s) - \mathbf{r}_f(s)) \cdot \mathbf{n}(s)]^2 \, ds \tag{6.10}$$

is a suitably invariant measure provided that the displacement between the two curves is small. This is now used to define a new norm $\| \cdot \|_{\overline{\mathbf{n}}}$ with respect to an estimated or

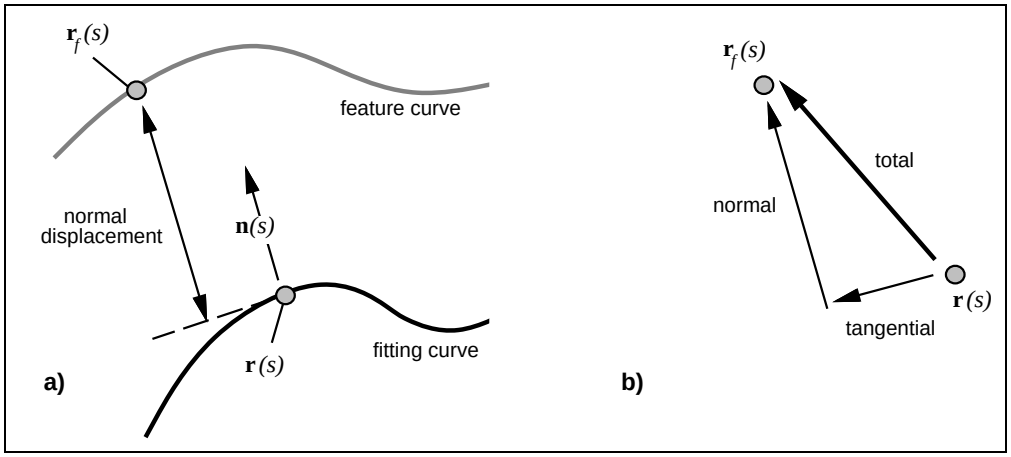


Figure 6.5: Normal Displacement. *a) Displacement along the normal from one curve to another, as shown, forms the basis for a measure of difference between curves that is approximately invariant to re-parameterisation. b) Total displacement can be factored vectorially into two components, tangential and normal.*

template curve $\overline{\mathbf{r}}(s)$, whose normals are $\overline{\mathbf{n}}(s)$:

$$\|\mathbf{r}\|_{\overline{\mathbf{n}}}{}^2 \equiv \frac{1}{L}\int [\mathbf{r}(s)\cdot\overline{\mathbf{n}}(s)]^2\, ds. \tag{6.11}$$

It has the property that the norm-difference approximates the invariant distance measure, that is

$$\|\mathbf{r}-\mathbf{r}_f\|_{\overline{\mathbf{n}}} \approx d(\mathbf{r},\mathbf{r}_f),$$

provided both curves $\mathbf{r}$ and $\mathbf{r}_f$ are sufficiently close to the estimated curve $\overline{\mathbf{r}}$. Norms $\|\mathbf{Q}\|_{\overline{\mathbf{n}}}$ in spline space and $\|\mathbf{X}\|_{\overline{\mathbf{n}}}$ in shape-space can be defined by inducing them from the curve norm $\|\mathbf{r}\|_{\overline{\mathbf{n}}}$, just as $\|\mathbf{Q}\|$ and $\|\mathbf{X}\|$ were induced from $\|\mathbf{r}\|$ originally.

The next step is to take account of the practicalities of image measurement by expressing the invariant difference discretely. Suppose normal vectors are sampled at regularly spaced points $s = s_i,\ i = 1,\ldots,N$, with inter-sample spacing h, along the entire curve $\mathbf{r}(s)$, so that, in the case of an open curve,

$$s_1 = 0,\quad s_{i+1} = s_i + h,\quad \text{and}\quad s_N = L.$$

Then the norm-difference (6.10) can be approximated as a sum:

$$\|\mathbf{r} - \mathbf{r}_f\|_{\overline{\mathbf{n}}}{}^2 \approx \frac{1}{N} \sum_{i=1}^{N} \left[(\mathbf{r}_f(s_i) - \mathbf{r}(s_i)) \cdot \overline{\mathbf{n}}(s_i) \right]^2 . \tag{6.12}$$

Recall that, in shape-space, $\mathbf{r}(s_i)$ can be expressed explicitly in terms of the shape-space vector $\mathbf{X}$:

$$\mathbf{r}(s_i) = U(s_i)(W\mathbf{X} + \mathbf{Q}_0)$$

so the norm-difference can be expressed explicitly in terms of the shape-space vector $\mathbf{X}$ as

$$\|\mathbf{r} - \mathbf{r}_f\|_{\overline{\mathbf{n}}}{}^2 \approx \frac{1}{N} \sum_{i=1}^{N} \left(\nu_i - \mathbf{h}(s_i)^T [\mathbf{X} - \overline{\mathbf{X}}] \right)^2 \tag{6.13}$$

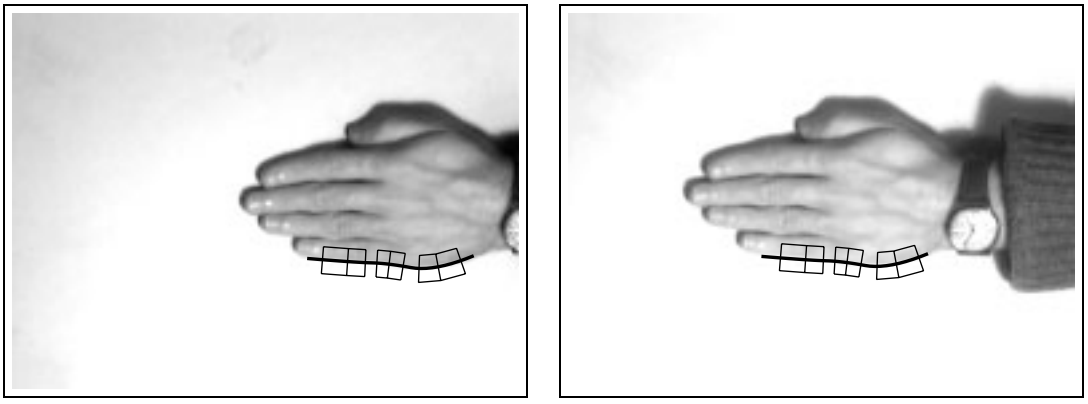
where

$$\nu_i = (\mathbf{r}_f(s_i) - \overline{\mathbf{r}}(s_i)) \cdot \overline{\mathbf{n}}(s_i), \tag{6.14}$$

is the "innovation" — the displacement measured relative to the mean shape and resolved along the normal, and

$$\mathbf{h}(s)^T = \overline{\mathbf{n}}(s_i)^T U(s_i) W. \tag{6.15}$$

Details of an algorithm to solve the fitting problem, using the invariant norm, are developed next. In the meantime, note (figure 6.6) that the new algorithm solves the reluctance problem demonstrated earlier arising from the use of norm-difference with the inherited parameterisation (figure 6.4).



(a) (b)

Figure 6.6: Normal displacement solves the reluctance problem. *Curve fitting using normal displacement is applied repeatedly to a sequence of images starting with (a) and ending with (b). The fitted curve follows the moving image feature, without the reluctance associated with norm-difference and inherited parameterisation (figure 6.4).*

Weighted norm

A more general sampled form for the norm, in place of (6.13), incorporates weights w_i:

$$\|\mathbf{r} - \mathbf{r}_f\|_{\overline{\mathbf{n}}}{}^2 = \left(\sum_{i=1}^{N} w_i\right)^{-1} \sum_{i=1}^{N} w_i \delta_i^2 \tag{6.16}$$

where

$$\delta_i = \nu_i - \mathbf{h}(s_i)^T[\mathbf{X} - \overline{\mathbf{X}}].$$

For instance, setting

$$w_1 = w_N = \frac{1}{2} \quad \text{and} \quad w_i = 1,\ 1 < i < N$$

implements the trapezium rule which correctly takes account of the ends of an open curve. For closed curves, in which $i = 0$ and $i = N$ represent the same physical point which must be counted exactly once, appropriate weights are

$$w_i = 1 \quad \text{for} \quad 1 \leq i < N \quad \text{and} \quad w_N = 0.$$

Weights can also be used to implement modified norms which accentuate areas of the image-feature curve. One good reason for doing this is to allow increased influence around areas of fine detail, such as the ends of the fingers in figure 5.1.

A mechanism closely related to variable weighting is non-uniform sampling in which intervals between successive s_i are allowed to vary. For instance, denser sampling may be desirable in areas of fine detail. Alternatively, it may be desirable to sample the feature curve at equal intervals of arclength, rather than of the spline parameter s which is not generally arclength. Lastly, an effect similar to sampling uniformly in arclength can be achieved by appropriate weighting that compensates for non-uniform sampling:

$$w_i = |\overline{\mathbf{r}}'(s_i)| \quad \text{where} \quad \overline{\mathbf{r}}'(s) \equiv \frac{d\overline{\mathbf{r}}}{ds}.$$

6.3 Recursive solution of curve-fitting problems

We already saw one solution to the regularised curve-fitting problem (6.3), using projection followed by weighted summation. The new fitting problem based on the normal displacement measure is best solved in a rather different style. The new algorithm is recursive, working by traversing the data-curve once, updating the estimated shape $\hat{\mathbf{X}}$ as it does so. (In fact the original problem (6.3) could also be solved concisely in the recursive manner.) The complete fitting algorithm is given in figure 6.7. For the regularised fitting problem above, we set the "measurement error" constant $\sigma_i = \sigma = \sqrt{N}$, but in subsequent chapters, other values of σ_i will be used. The first two steps of the algorithm establish feature points $\mathbf{r}_f(s_i)$ which serve as the data for curve fitting. Step 3 initialises the "information matrix" S_i which is a measure of the "strength" of each intermediate estimate $\hat{\mathbf{X}}_i$, taking account of the first i data points. Step 3 also initialises the "information weighted sum" $\mathbf{Z}_i$ which accumulates the influence of the mean shape and the individual measurements, each with its proper weight. In step 4, the measurements $\mathbf{r}_f(s_i)$ are assimilated in turn. Note that, as expected, it is only the normal component ν_i of each measurement that is used. [Note the effect of the weighted norm (6.16) is to choose a variable "measurement error" $\sigma_i = \sqrt{w_i^{-1} \sum_j w_j}$.] In step 5, the "aggregated" observation vector $\mathbf{Z}$ is defined, together with S, its statistical information as an estimator of $\mathbf{X}$. In fact $\mathbf{Z}$ is an unbiased estimate not of $\mathbf{X}$ directly, but of $S\mathbf{X}$. (This is much safer than trying to deal with an estimate of $\mathbf{X}$ itself, given that the inverse of S need not exist.) In step 6, the aggregated measurement $\mathbf{Z}$ that incorporates the influence of all data points, is finally combined in what is, in fact, an information weighted sum (see derivation below) to give the estimated shape-space vector $\hat{\mathbf{X}}$.

Curve-fitting problem

Given an initial shape estimate $\overline{\mathbf{r}}(s)$ (or $\overline{\mathbf{X}}$ in shape-space) with normals $\overline{\mathbf{n}}(s)$, and a regularisation weight matrix $\overline{S}$, solve:

$$\min_{\mathbf{X}} T \quad \text{where} \quad T = (\mathbf{X} - \overline{\mathbf{X}})^T \overline{S} (\mathbf{X} - \overline{\mathbf{X}}) + \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left(\nu_i - \mathbf{h}(s_i)^T [\mathbf{X} - \overline{\mathbf{X}}] \right)^2 .$$

Algorithm

1. Choose samples $s_i, i = 1, \ldots, N$, s.t. $s_1 = 0$, $s_{i+1} = s_i + h$, $s_N = L$.
2. For each i, apply some image-processing filter along a suitable line (e.g. curve normal) passing through $\overline{\mathbf{r}}(s_i)$, to establish the position of $\mathbf{r}_f(s_i)$.
3. Initialise
$$\mathbf{Z}_0 = 0, \quad S_0 = 0.$$
4. Iterate, for $i = 1, \ldots, N$:
$$\begin{aligned} \nu_i &= (\mathbf{r}_f(s_i) - \overline{\mathbf{r}}(s_i)) \cdot \overline{\mathbf{n}}(s_i); \\ \mathbf{h}(s_i)^T &= \overline{\mathbf{n}}(s_i)^T U(s_i) W; \\ S_i &= S_{i-1} + \frac{1}{\sigma_i^2} \mathbf{h}(s_i) \mathbf{h}(s_i)^T; \\ \mathbf{Z}_i &= \mathbf{Z}_{i-1} + \frac{1}{\sigma_i^2} \mathbf{h}(s_i) \nu_i. \end{aligned}$$
5. The aggregated observation vector is
$$\mathbf{Z} = \mathbf{Z}_N \quad \text{with associated statistical information} \quad S = S_N.$$
6. Finally, the best fitting curve is given in shape-space by:
$$\hat{\mathbf{X}} = \overline{\mathbf{X}} + (\overline{S} + S)^{-1} \mathbf{Z}.$$

Figure 6.7: Recursive algorithm for curve fitting.

Example 1

A very simple fitting problem for tutorial purposes is illustrated in figure 6.8. It

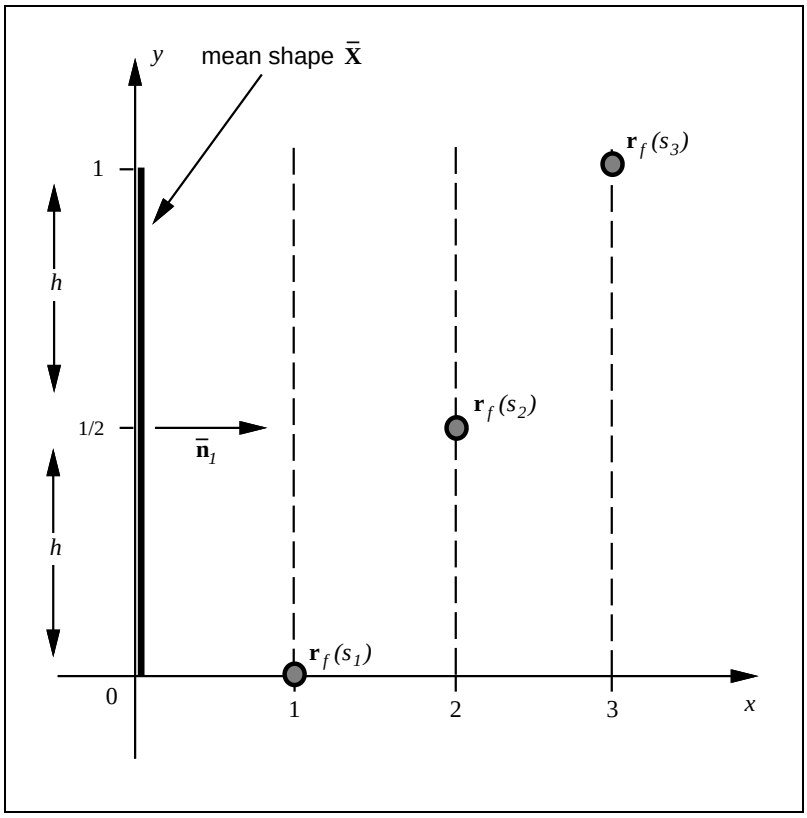


Figure 6.8: *Example fitting problem for the recursive algorithm — see text.*

involves a family of vertical line segments

$$\mathbf{r}(s) = (x, s)^T \quad \text{for} \quad 0 \leq s \leq L$$

(a single-span linear spline!) with length $L = 1$. Take the template $\mathbf{Q}_0$ to correspond to a vertical line at the origin:

$$\mathbf{r}_0(s) = (0, s)^T \quad \text{for} \quad 0 \leq s \leq L$$

so that shape-space is then parameterised by the single-component shape-vector $\mathbf{X} = x$, and

$$U(s)W = (1,\, 0)^T,$$

so that $\mathbf{X}$ is transformed to spline space as

$$U(s)W\mathbf{X} + \mathbf{r}_0 = (x,\, s)^T,$$

as required. The mean shape is also taken to be the line through the origin, so $\overline{\mathbf{X}} = 0$. Measurements will be made at

$$s_1 = 0,\ s_2 = \frac{1}{2},\ s_3 = 1$$

so that $h = 1/2$, as illustrated in figure 6.8,

$$\sigma^2 = N = \frac{L}{h} + 1 = 3, \quad \text{and} \quad \overline{\mathbf{n}}(s_i) = (1,0)^T \quad \text{for} \quad i = 1,2,3.$$

Now we can also calculate

$$\mathbf{h}(s_i)^T = \overline{\mathbf{n}}(s_i)^T U(s_i)W = 1 \quad \text{for} \quad i = 1,2,3.$$

The data points shown in the figure are

$$\mathbf{r}_f(s_1) = (1,0)^T,\ \mathbf{r}_f(s_2) = (2,1/2)^T,\ \mathbf{r}_f(s_3) = (3,1)^T.$$

It is not difficult to show that the metric matrix for this shape-space is $\mathcal{H} = 1$.

First, take the case of fitting without regularisation, so $\alpha = 0$. Following the steps of the algorithm gives

i	ν_i	S_i	$\mathbf{Z}_i$
0		0	0
1	1	$\frac{1}{3}$	$\frac{1}{3}$
2	2	$\frac{2}{3}$	1
3	3	1	2

The aggregated measurement is then $\mathbf{Z} = 2$ with information $S = 1$, and since $\overline{S} = 0$ (because $\alpha = 0$), the estimate is $\hat{\mathbf{X}} = S^{-1}\mathbf{Z} = 2$, simply the average value of the x-coordinates of the three data points, as might be expected.

Example 2

If regularisation is added with $\alpha = 1$, the main part of the algorithm proceeds as above, but since now $\overline{S} = 1$, the final step is

$$\hat{\mathbf{X}} = \overline{\mathbf{X}} + \left(\overline{S} + S\right)^{-1} \mathbf{Z} = 0 + \frac{1}{2} \cdot 2 = 1,$$

which has been pulled down towards $\overline{\mathbf{X}} = 0$ by regularisation, as expected.

Proof of correctness: the algorithm is of a standard type for recursive solution of least-squares problems. Defining the partial sum

$$T_i = \frac{1}{\sigma^2} \sum_{j=1}^{i} \left(\nu_j - \mathbf{h}(s_j)^T [\mathbf{X} - \overline{\mathbf{X}}]\right)^2 .$$

it is straightforward to prove by induction on i that

$$T_i = (\mathbf{X} - \overline{\mathbf{X}})^T S_i (\mathbf{X} - \overline{\mathbf{X}}) - (\mathbf{X} - \overline{\mathbf{X}})^T \mathbf{Z}_i - \mathbf{Z}_i^T (\mathbf{X} - \overline{\mathbf{X}}) + c_i \quad \text{for} \quad i = 1, \ldots, N,$$

where c_i is a constant, independent of $\mathbf{X}$. When $i = N$, this gives

$$T_N = (\mathbf{X} - \overline{\mathbf{X}})^T S (\mathbf{X} - \overline{\mathbf{X}}) - (\mathbf{X} - \overline{\mathbf{X}})^T \mathbf{Z} - \mathbf{Z}^T (\mathbf{X} - \overline{\mathbf{X}}) + c_N,$$

where $S = S_N$ and $\mathbf{Z} = \mathbf{Z}_N$, as in the algorithm. Now

$$T = (\mathbf{X} - \overline{\mathbf{X}})^T \overline{S} (\mathbf{X} - \overline{\mathbf{X}}) + T_N$$

and completing the square gives

$$T = (\mathbf{X} - \hat{\mathbf{X}})^T (\overline{S} + S)(\mathbf{X} - \hat{\mathbf{X}}) + c,$$

where c is independent of $\mathbf{X}$, and $\hat{\mathbf{X}}$ is as defined in the algorithm, so $\mathbf{X} = \hat{\mathbf{X}}$ optimises T, as required.

Validation gate

The innovation ν_i in (6.14) represents the difference between the actual measurement and the measurement that would be predicted at $s = s_i$ based on the mean shape. It is potentially useful for detecting and deleting rogue data or "outliers". In the context of tracking, outliers arise when the object being tracked is partially obscured,

and the edge of some background object is then detected and masquerades as $\mathbf{r}_f(s_i)$, for some i. The resulting error in position may be considerable, well outside the normal range attributable to random factors such as electrical and optical noise. Such an "outlier" should be signaled by an unusually large magnitude $|\nu_i|$ of innovation. If this occurs, the ith measurement can be discarded, and the ith iteration in the algorithm altogether omitted. Otherwise, the ith data point is said to be validated and the ith iteration proceeds as normal.

It remains to choose a threshold — at what level is it considered that $|\nu_i|$ is sufficiently large to signal an outlier? A principled answer to this question emerges from statistical models described in chapter 8, leading to the "validation-gate" for outlier removal. The width of the validation gate effectively fixes the width of the search region for image processing along normals (figure 5.1 on page 98). In fact the threshold on $|\nu_i|$ determines the length of the search segment, the interval on the normal that lies within the search region.

Fitting corners

In some applications it may be desired to fit to corner features rather than edge-features, or to a mixture of corners and edges. The recursive fitting algorithm extends in a straightforward manner. Suppose the measured feature $\mathbf{r}_f(s_i)$ is a corner or other distinguished point, so that its full displacement $\boldsymbol{\nu}_i^r = \mathbf{r}_f(s_i) - \mathbf{r}(s_i)$ must be taken into account, rather than just the normal component. Then step 4 of the recursive algorithm in figure 6.7 must be modified to take account of this, as in figure 6.9.

Alternative recursive solution to the fitting problem

For completeness, an alternative to the fitting algorithm described above in figure 6.7 is given here. It solves exactly the same minimisation problem, but using an alternative set of variables. Where the algorithm above was based on "information" S, the algorithm here is based on covariance P. This algorithm will be readily recognised as a special case of a "Kalman filter" by those who are already familiar with them. This alternative curve-fitting algorithm can also be explained intuitively. In place of the information matrix S_i, this algorithm is expressed in terms of its inverse $P_i = S_i^{-1}$ (to be interpreted in later chapters as a statistical covariance). The weighted sum variable $\mathbf{Z}_i$ is no longer needed; instead successive estimates $\hat{\mathbf{X}}_i$, based on the first i data points, are generated directly. The variable ν_i' is a "successive" form of innovation, the difference between the actual value of the ith measurement and its expected value based on extrapolation from the previous $i-1$ measurements, and it is only this difference

Modified iterative step:

4 Iterate, for $i = 1, \ldots, N$:

$$\begin{aligned} \boldsymbol{\nu}_i^r &= \mathbf{r}_f(s_i) - \overline{\mathbf{r}}(s_i); \\ h^r(s_i)^T &= U(s_i)W; \\ S_i &= S_{i-1} + \frac{1}{\sigma_i^2} h^r(s_i) h^r(s_i)^T; \\ \mathbf{Z}_i &= \mathbf{Z}_{i-1} + \frac{1}{\sigma_i^2} h^r(s_i) \boldsymbol{\nu}_i^r. \end{aligned}$$

Figure 6.9: Recursive fitting algorithm: modification for corner features.

that generates any change in successive estimates $\hat{\mathbf{X}}_i$. The proof of equivalence of the two algorithms is omitted here, but could be found in a standard textbook on statistical filtering.

Computational cost

Both algorithms have computational complexity $O(NN_X^2)$, where N_X is the dimension of shape-space, as usual. This is based on the assumption that, for a given shape-space, $\mathcal{H}^{-1}$ and the products $U(s_i)W$ can be calculated off-line and need not be counted towards the total computational cost. It is further assumed that $N \geq N_X$ which is normally the case as this is the minimum value for which S can have full rank. The original algorithm has only one $O(NN_X^2)$ operation, as opposed to two in the alternative algorithm. However, the original algorithm involves additional matrix inversions which have complexity $O(N_X^3)$. This means that for small $N_X \ll N$ the original algorithm is more efficient, but as $N_X \to N$ the alternative algorithm becomes the more efficient.

Example 3

Example 2 is repeated here using the alternative algorithm; of course it should achieve the same result. The estimate is $\hat{\mathbf{X}} = \hat{\mathbf{X}}_3 = 1$, in agreement with the original algorithm.

Algorithm:

1. Obtain measurements $\mathbf{r}_f(s_i)$ as before.

2. Initialise
$$\begin{aligned} P_0 &= \overline{S}^{-1} \\ \hat{\mathbf{X}}_0 &= \overline{\mathbf{X}} \end{aligned}$$

3. Iterate, for $i = 1, \ldots, N$:
$$\begin{aligned} \nu_i &= (\mathbf{r}_f(s_i) - \overline{\mathbf{r}}(s_i)) \cdot \overline{\mathbf{n}}(s_i); \\ \mathbf{h}(s_i)^T &= \overline{\mathbf{n}}(s_i)^T U(s_i) W; \\ \nu_i' &= \nu_i - \mathbf{h}(s_i)^T (\hat{\mathbf{X}}_{i-1} - \overline{\mathbf{X}}); \\ K_i &= P_{i-1}\mathbf{h}(s_i)(\mathbf{h}(s_i)^T P_{i-1}\mathbf{h}(s_i) + \sigma_i^2)^{-1} \\ \hat{\mathbf{X}}_i &= \hat{\mathbf{X}}_{i-1} + K_i \nu_i'; \\ P_i &= (I - K_i \mathbf{h}(s_i)^T) P_{i-1}; \end{aligned}$$

4. The best fitting curve is given in shape-space by:
$$\hat{\mathbf{X}} = \mathbf{X}_N.$$

Note that intermediate estimates $\hat{\mathbf{X}}_i$ are automatically generated by this algorithm.

Figure 6.10: Alternative recursive fitting algorithm.

i	ν_i	ν_i'	K_i	$\hat{\mathbf{X}}_i$	P_i
0				0	1
1	1	1	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{3}{4}$
2	2	$\frac{7}{4}$	$\frac{1}{5}$	$\frac{3}{5}$	$\frac{3}{5}$
3	3	$\frac{12}{5}$	$\frac{1}{6}$	1	$\frac{1}{2}$

Note also that, as expected, the final value of covariance P_3, satisfies $P_3 = S_3^{-1}$ where S_3 is the final information from example 2.

6.4 Examples

B-spline snake

An example of fitting a B-spline snake is given in figure 6.11. Working in a spline

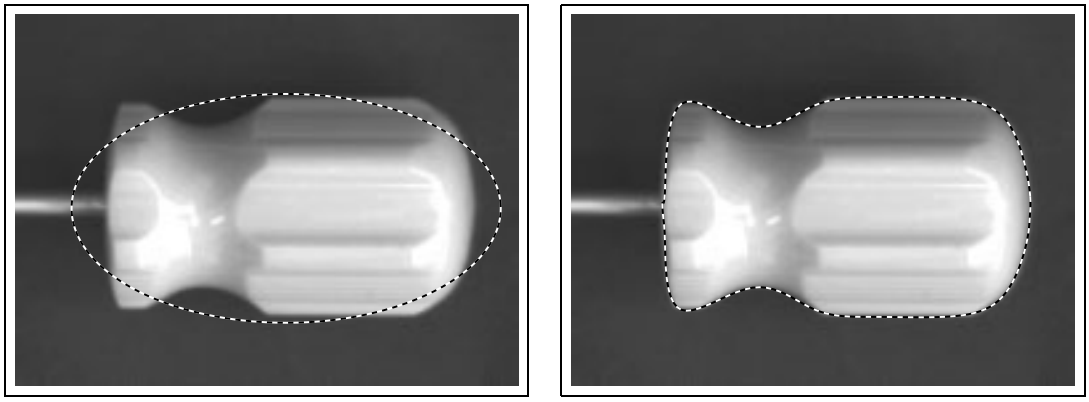


Figure 6.11: Using a snake to capture an outline. *For automated planning of robot grasps (see chapter 1) it is necessary to capture the outline of the part to be grasped. This can be done even without much prior knowledge of part shape, using a B-spline snake initialised as an ellipse (left), achieving a good fit (right). (Figure courtesy of Colin Davidson.)*

space is appropriate when no strong assumption can be made about the expected shape. The snake is initialised using moments (as explained in figure 4.6 on page 82) to obtain a coarse elliptical approximation to the outline. Weak regularisation is used, just sufficient to stabilise the computation but without unduly biasing the fit. The low level of regularisation suffices in this case because the background is clean and featureless.

More stable behaviour may be needed when the background is more cluttered or the foreground partly obscured or simply outside the search region. Stability can be achieved by increasing the strength of regularisation as in figure 6.12. In that example, missing measurements around the fingertips mean that the shape of the fitted curve is underconstrained. (In fact a minimal regulariser with $\alpha = 0.001$ is applied simply to ensure numerical stability). When a significant degree of regularisation ($\alpha = 0.1$) is applied, the missing data is satisfactorily interpolated by defaulting to the template (initial) shape. Regularisation is set to be invariant to Euclidean similarities, to allow the snake to rotate and translate freely, while maintaining shape constraints. Even

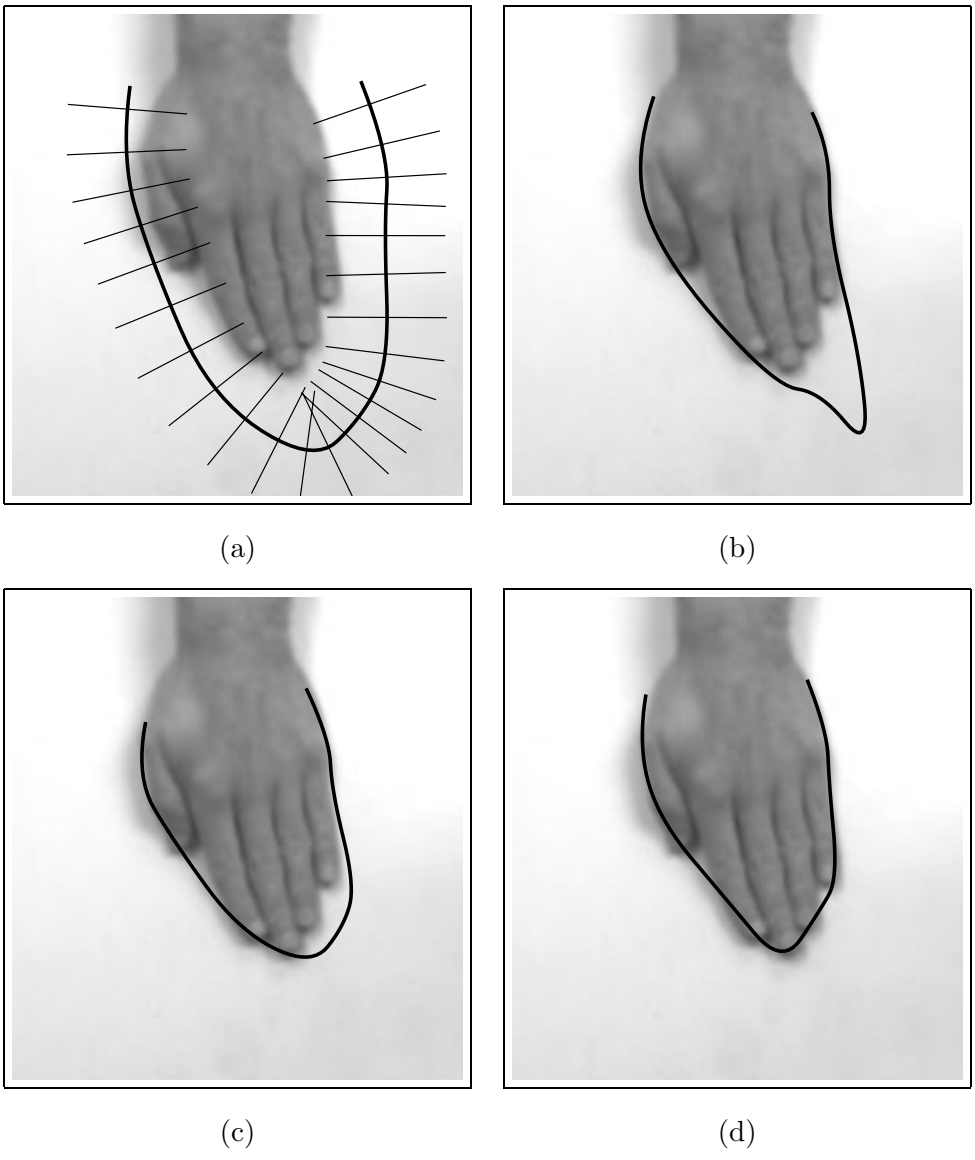


Figure 6.12: Regularisation stabilises snake fitting. *Given an initial snake curve (a) in which some features (fingertips) fall outside the search region, unregularised fitting gives poor results (b). Introducing regularisation (with invariance to Euclidean similarities) produces a good fit (c). This is refined further after iterating the fitting cycle (d).*

more accurate fitting is achieved by iterating the fitting process to convergence. At each successive iteration the fitted shape from the previous iteration is used as an estimate from which image measurements are made. However the stabilising template remains constant throughout.

An alternative method of stabilisation is to restrict the displacement of points $\mathbf{r}(s)$ to run along a family of straight lines such as normals, or parallel lines in a chosen direction. It is simple and effective, but usable only for applications where motion can be restricted to a fixed family of lines. Surveillance of railed vehicles or traffic confined to lanes is one example.

Deformable template

At the expense of needing more specific prior knowledge about shape, deformable templates running in a suitably constrained shape-space generally behave more stably than snakes. An example of fitting a deformable template in affine shape-space is shown in figure 6.13. The initial estimated contour was obtained using moments. Then recursive fitting in affine spaces considerably refines contour shape, as shown.

Rudimentary tracking

Much of the second part of the book concerns tracking shapes in motion and it is interesting to try applying the curve-fitting algorithm to that problem. Now a curve must be fitted to each image in a sequence, and an estimated curve is therefore required for each image. The obvious strategy is to use the fitted curve from one image as the estimate for the next. This is quite effective, as figure 6.14 shows, provided the normal search segments are sufficiently long to encompass the lag of the estimated curves. The faster the motion, the longer search segments need to be. However, longer search segments have the drawback that tracking becomes more prone to distraction by background clutter, as the figure shows. There is therefore a trade-off between agility of motion and density of clutter. The capacity to deal with this trade-off is greatly expanded by dynamical modelling. If each estimated curve, rather than being a mere copy of the previous fitted curve, is actually an extrapolation of the motion to date, tracking performance can be greatly improved, and this is the subject of chapters 9 and 10.

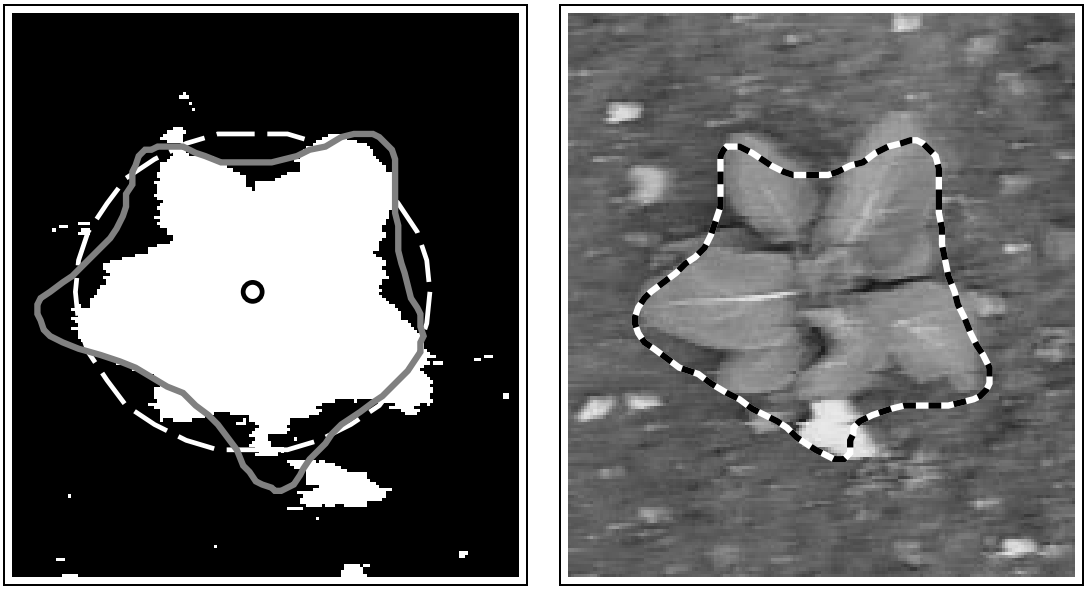


Figure 6.13: Deformable template in affine space. *From an initial estimate (grey curve, left) obtained using moments, a cabbage template is fitted to image data over an affine space to obtain the fit shown (right).*

Bibliographic notes

Regularisation, used in this chapter to bias fitted curves towards a default shape, is a standard algorithmic device for stabilising otherwise unstable or underconstrained systems of equations (Press et al., 1988). Regularisation has been used a good deal in Computer Vision (Horn and Schunk, 1981; Grimson, 1981; Ikeuchi and Horn, 1981; Poggio et al., 1985; Terzopoulos, 1986) and in image processing for "constrained restoration" of degraded images (Gonzales and Wintz, 1987).

An important aspect of the curve fitter described in the chapter is that it avoids the very considerable computational expense of applying filters to entire images. Feature detection is restricted to be one-dimensional, along normal curves (Harris and Stennett, 1990; Lowe, 1991), and within a region of interest (Inoue and Mizoguchi, 1985) or search region.

The need to use normal displacement, factoring out the spurious tangential displacement is known in vision as the "aperture problem," is the basis of the visual

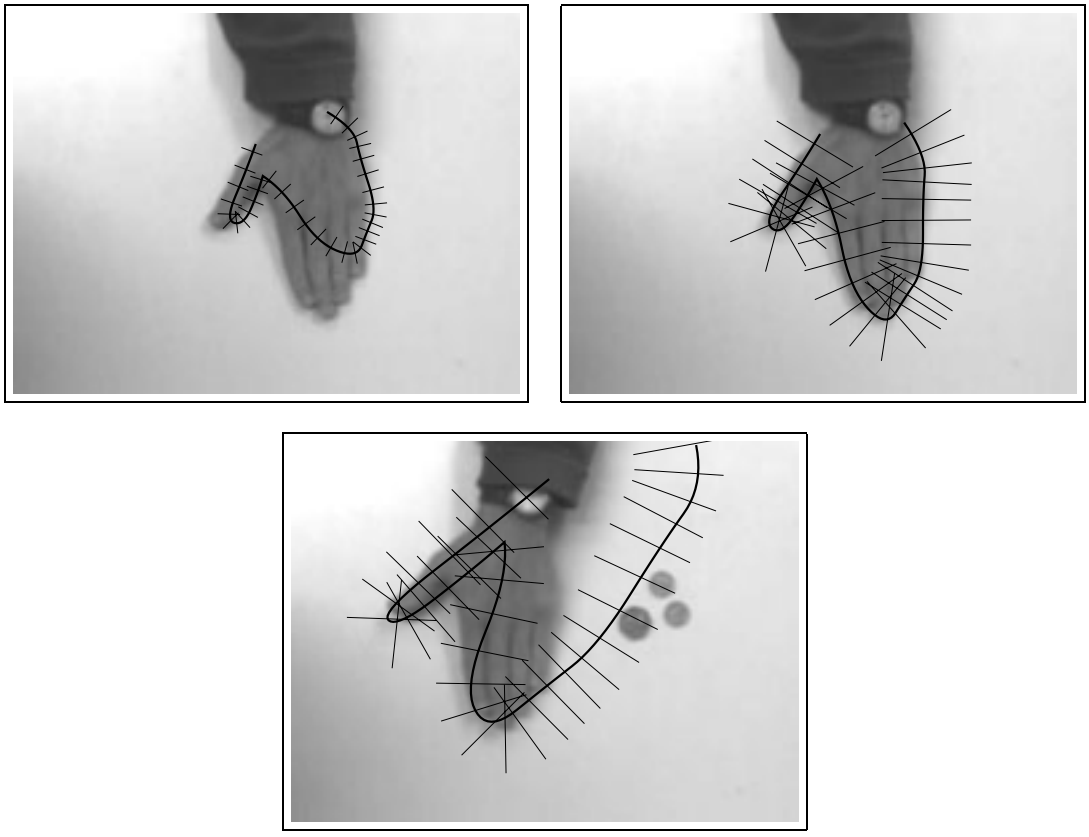


Figure 6.14: From fitting to tracking. *Recursive fitting is applied to successive images in a sequence in which a hand moves from right to left. The fitted curve from one image is used as the initial estimate in the next. With short search segments (left), even slow motion causes estimated shape to lag sufficiently that the hand falls outside the search region. Longer search segments (right) may cure this problem, but make tracking more prone to distraction by clutter (bottom).*

"barber's pole" illusion and has received much attention in the design of algorithms for analysing visual motion (Horn and Schunk, 1981; Hildreth, 1983; Horn, 1986; Murray and Buxton, 1990). However, normal displacement alone underconstrains the motion of a contour — the "aperture problem in the large" (Waxman and Wohn, 1985), though this defect is somewhat mitigated by regularisation.

Curve fitting is done efficiently here by a recursive algorithm, and this is based on

recursive least-squares estimation (Bar-Shalom and Fortmann, 1988) which in turn is an application of the "dynamic programming" principle (Bellman and Dreyfus, 1962) for optimisation, applied to quadratic, multi-variate minimisation (Jacobs, 1993). Dynamic programming was first used with snakes by (Amini et al., 1988), in a discrete form as opposed to the continuous form described in this chapter. The original snake paper (Kass et al., 1987) used sparse matrix methods to solve least-squares curve fitting and that is closely related to the recursive algorithm in the case that shape-space is the spline space $\mathcal{S}_Q$. The validation of data by checking the absolute value of innovations is used in a simple form of robust estimator known as an m-estimator (Hampel et al., 1995). It is a special case from the family of "robust" estimators known as M-estimators which allow some flexibility in the way outliers are treated. General M-estimators can be expensive to compute, requiring repeatedly refined regressions to form the final robust estimate. The simple validation mechanism described in the chapter, the validation gate (Bar-Shalom and Fortmann, 1988), has the great virtue of low computational cost and this will be essential for real-time processing of image sequences in later chapters.

Chapter 7

Pose recovery

In certain three-dimensional applications (chapter 1), such as the 3D mouse in figure 1.16 on page 20, a shape-vector $\mathbf{X}$ is used to compute pose, in that case the position and attitude of the hand. Similarly, in facial animation, it is desirable to compute the attitude of the head, independently of expression if possible. The problem is to convert a shape-vector $\mathbf{X}$, from a planar or three-dimensional shape-space respectively, into three-dimensional translation $\mathbf{R}_c$ and rotation R.

7.1 Calculating the pose of a planar object

In the planar affine space, pose is recovered from a shape-vector $\mathbf{X}$ by first obtaining affine parameters $\mathbf{u}, M$ via (4.5) on page 78. Then $\mathbf{u}, M$ are used to obtain pose parameters. This is fairly straightforward under orthographic projection. For large fields of view however it is necessary to use weak perspective and then the pose-recovery computation must be elaborated to compensate for the obliqueness of projection in the periphery of the field of view.

Orthographic projection

First, a solution is presented for the pose of an object under orthographic projection, suitable for use when the field of view is significantly smaller than 1 radian. Given a planar affine vector $\mathbf{X}$, the algorithm computes object pose as the position vector $\mathbf{R}_c = (X_c, Y_c, Z_c)^T$ of the object's centre and the orientation as a rotation R relative to the reference pose. First the algorithm is given, then a short proof (optional).

1. Compute $\mathbf{u}$ and M from
$$\mathbf{X} = (u_1, u_2, M_{11} - 1, M_{22} - 1, M_{21}, M_{12})^T.$$

2. Compute the eigenvectors $\mathbf{v}_1, \mathbf{v}_2$ and eigenvalues λ_1, λ_2 (both must be positive and are ordered so that $\lambda_1 \geq \lambda_2$) of the matrix MM^T.

3.
$$Z_c = f/\sqrt{\lambda_1}$$

4.
$$\cos\theta = \left(\frac{Z_c}{f}\right)^2 \det M$$

5.
$$(\cos\phi, \sin\phi)^T = \mathbf{v}_1$$

6.
$$R(\psi) = S_y(1/\cos\theta)R(-\phi)\ M\ \frac{Z_c}{f}$$
$$\text{where}\quad R(\psi) \equiv \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix} \quad \text{and} \quad S_y(\mu) \equiv \begin{pmatrix} 1 & 0 \\ 0 & \mu \end{pmatrix}.$$

7.
$$\text{Combined rotation: } R = R_z(\phi)R_x(\theta)R_z(\psi).$$

8.
$$\text{Translation: } \mathbf{R}_c = (X_c, Y_c, Z_c)^T \quad \text{where} \quad X_c = u_1 Z_c/f, \quad Y_c = u_2 Z_c/f.$$

Figure 7.1: Algorithm for recovery of the pose of a planar object.

The algorithm is summarised in figure 7.1 and a commentary follows. Given $\mathbf{u}, M$ from step 1 of the algorithm, we first work on M to recover the object's attitude as represented by the rotation matrix R. It is well known that a three-dimensional rota-

tion can be decomposed in terms of three *Euler angles* θ, ϕ, ψ appearing in a sequence of three standard rotations. It is particularly convenient here to use a decomposition that is aligned with the image plane:

$$R(\phi, \theta, \psi) = R_z(\phi) R_x(\theta) R_z(\psi). \tag{7.1}$$

This should be read from right to left as a z-rotation through an angle ψ followed by a rotation through angle θ about the x-axis and finally a z-rotation through angle ϕ, as illustrated in figure 7.2. Our algorithm recovers θ, ϕ and ψ so that the rotation matrix can be calculated as in (7.1). First Z_c, ϕ, θ and ψ are computed in steps 3–6. Note that a redundancy in ϕ arises from the indeterminacy of the sign of eigenvalue $\mathbf{v}_1$: for a given rotation R, ϕ can equally well be replaced by $\phi + 180^o$ (the substitution $\phi \to 180^o + \phi$, $\theta \to -\theta$, $\psi \to 180^o + \phi$ leaves R unchanged). Finally, the focal length f of the camera is known, so the image-plane components of translation $\mathbf{R}_c$ can be recovered directly in step 8. The result of applying this algorithm to a sequence of images of a moving hand are shown in figure 7.3.

Ambiguities

There are two kinds of ambiguity in the recovered parameter. The first is a straightforward indeterminacy in linear scale. If (X_c, Y_c, Z_c) was doubled and accompanied by a doubling in size of the object, there would be no visible effect on the image contour. The scale factor is fixed here by our convention in orthographic perspective that $Z_c = f$ in the standard view.

The second and more complex ambiguity is apparent in step (4) above: since $\cos\theta = \cos(-\theta)$, there must be two possible solutions $\pm\theta$. Unlike the indeterminacy of ϕ which represents simply an alternative representation of a given physical transformation, this is a genuine ambiguity, a reversal as illustrated in figure 7.4. In the 3D mouse application, for example, this could result in a "flip" of the object controlled by the mouse. One practical solution is to arrange for the camera to be oblique to the table so that the hand avoids the "singular" orientation $\theta = 0$, when image and object planes are parallel. Then at each time-step, the value $\pm\theta$ is chosen which give θ, ϕ and ψ closest to their values at the previous time-step.

Derivation of pose-recovery algorithm. The key point is that the effect of a rotation $R_x(\theta)$, on the image of a planar object lying on an xy plane, is to compress the image along

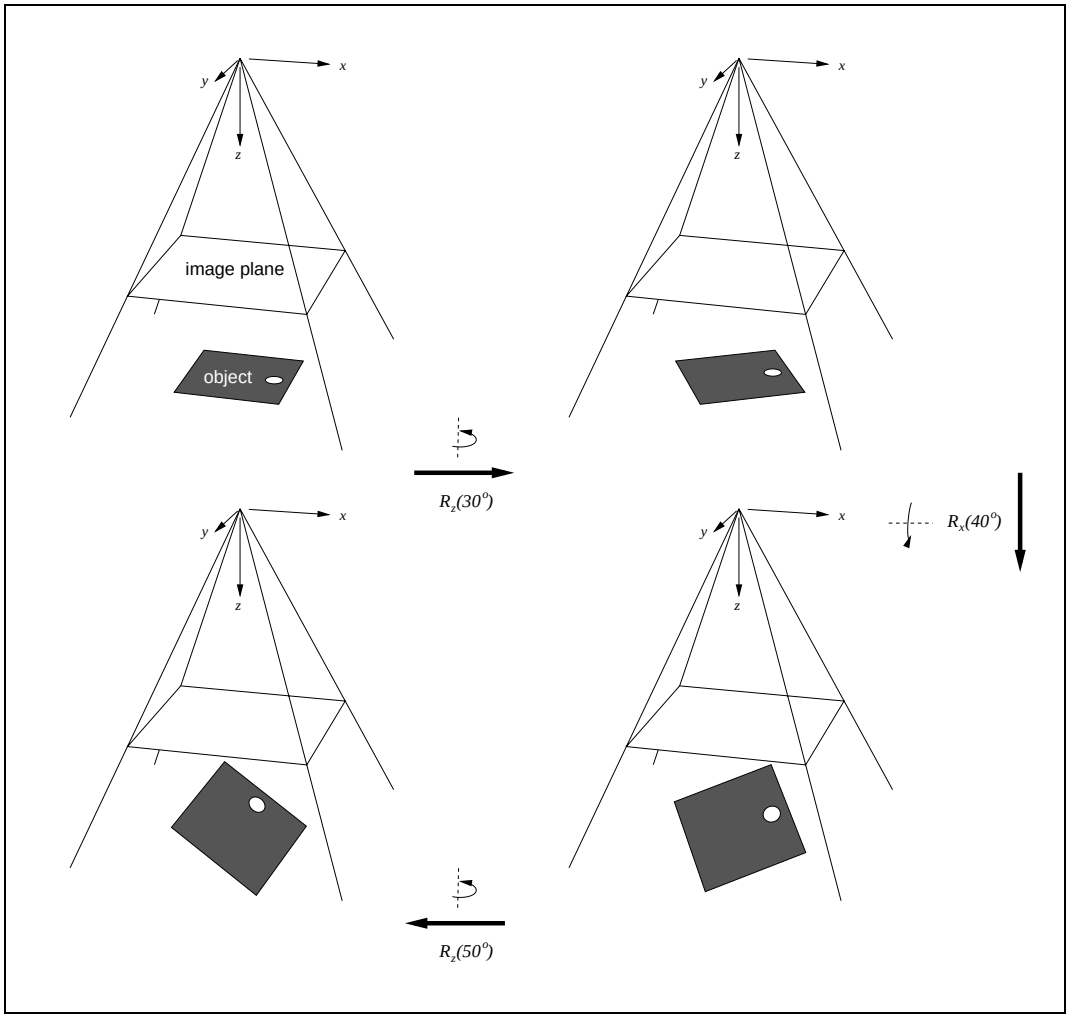


Figure 7.2: Euler angles. *The figure shows Euler angles to describe rotations in the camera coordinate frame. A general rotation R is expressed as a sequence (7.1) and the case $\phi = 50^o$, $\theta = 40^o$, $\psi = 30^o$ is illustrated.*

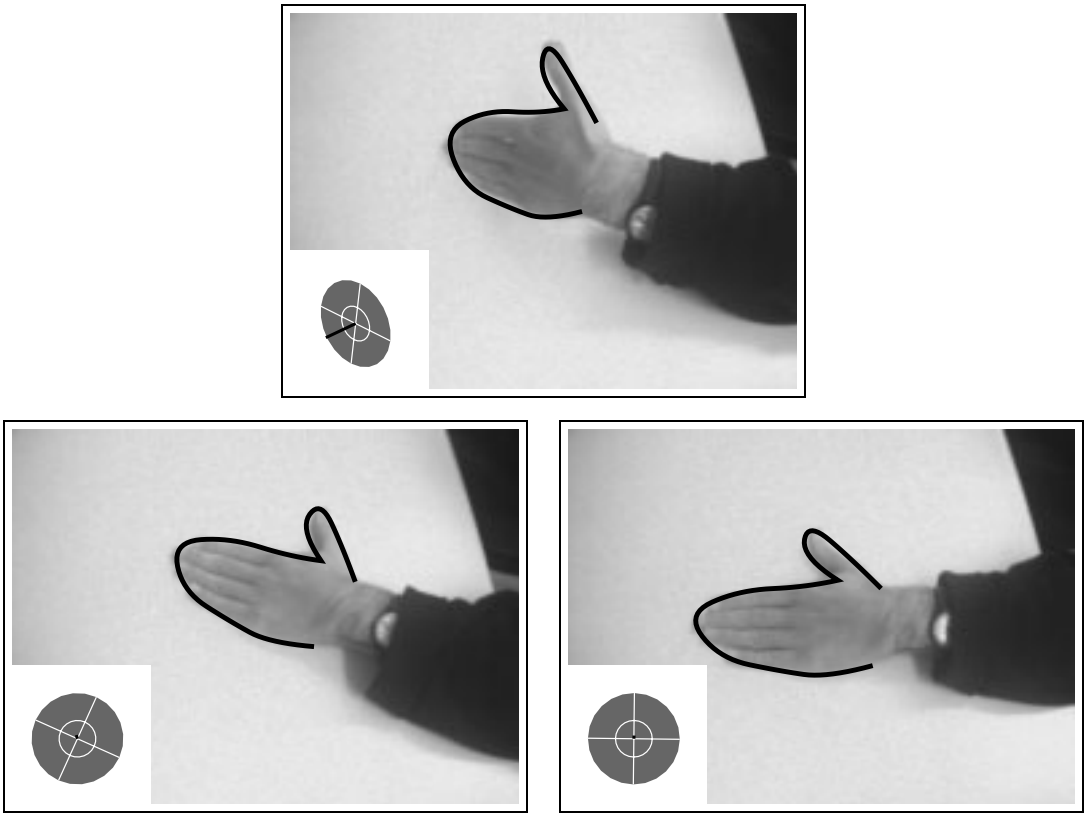


Figure 7.3: Computing pose. *The outline of a hand is tracked in a planar affine shape-space and the pose-recovery algorithm described above is applied. Computed pose is displayed using the "target" icon in the corner of each image.*

the y-axis, that is to apply a scaling transformation $S_y(\cos\theta)$. In that case, the affine image transformation due to the Euler angle transformations (7.1) and to distance scaling is:

$$M = \frac{f}{Z_c} R_z(\phi) S_y(\cos\theta) R_z(\psi).$$

The algorithm given above is simply a decomposition of this sequence, using the fact that

$$MM^T = \left(\frac{f}{Z_c}\right)^2 R_z(\phi) S_y(\cos^2\theta) R_z(-\phi)$$

which is in diagonal form.

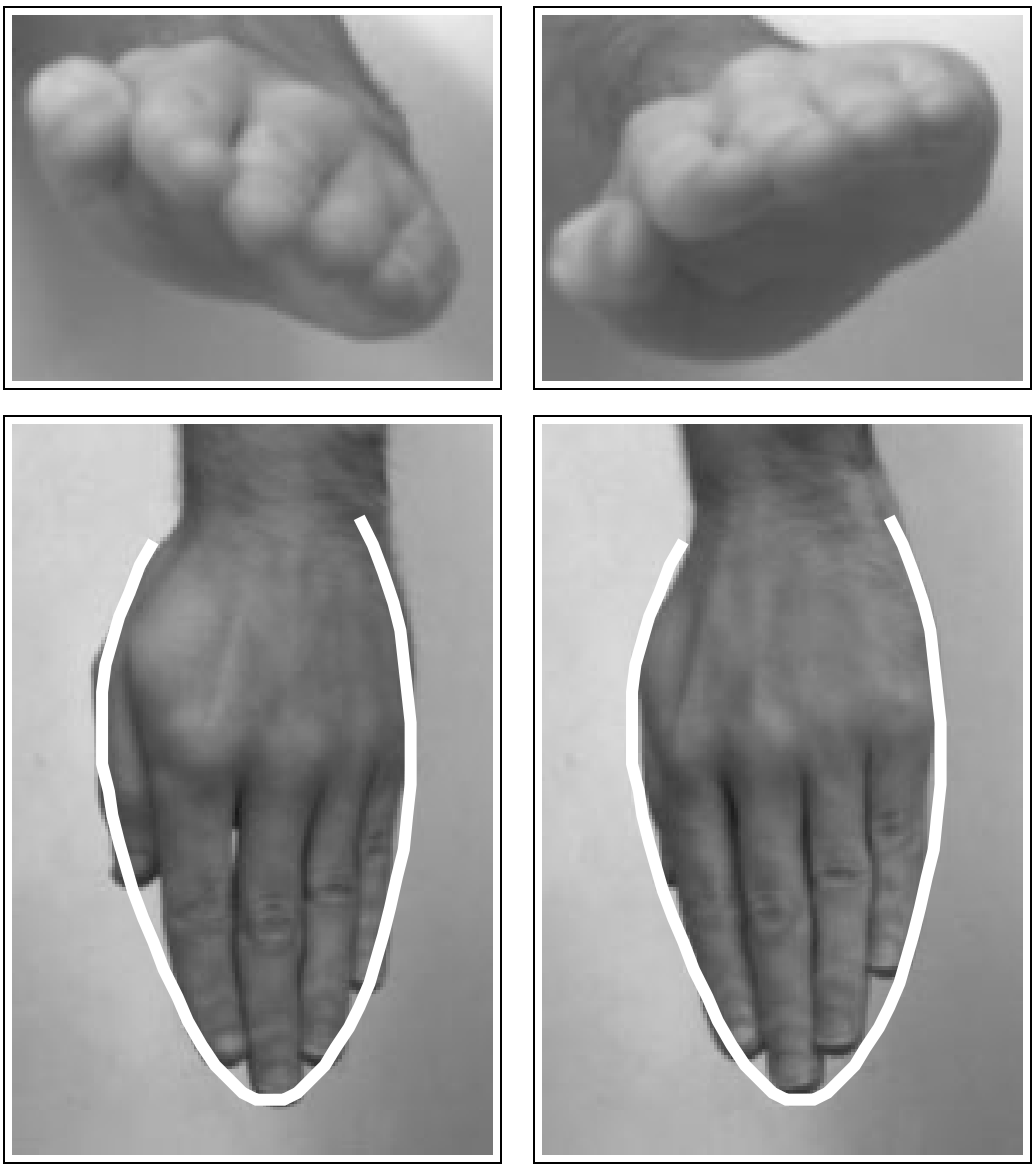


Figure 7.4: Ambiguity of pose recovery. *A hand in two alternative positions with equal and opposite slant (top row) gives rise to almost identical outline contours when viewed from above (bottom row).*

Weak Perspective projection

The pose-recovery method for orthographic projection can be simply extended for the case of weak perspective projection, when the field of view may be large but the object is still small. The most straightforward approach is to introduce a virtual orthographic image as in figure 7.5. The pose-recovery algorithm extended for weak perspective is

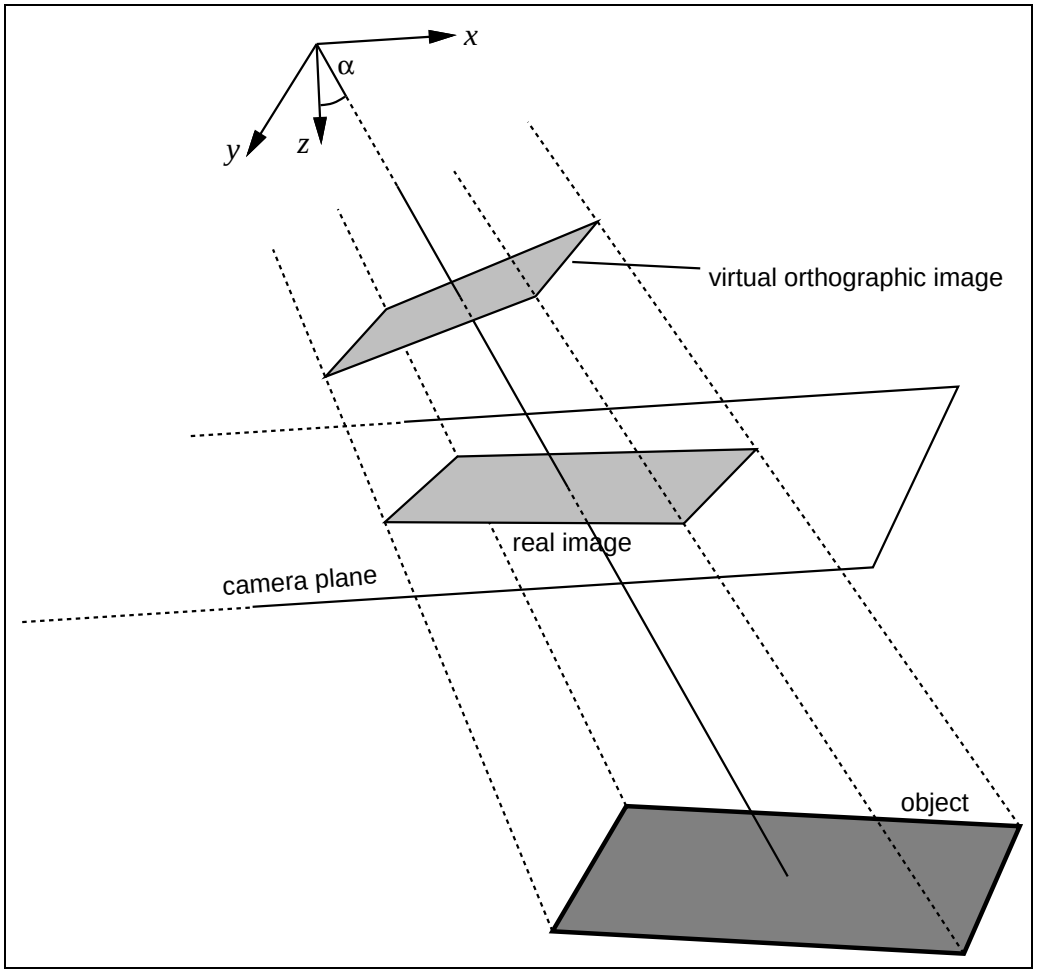


Figure 7.5: Pose correction for weak perspective. *Under weak perspective, pose recovery proceeds as in the orthographic case, on a virtual orthographic image as shown.*

1. Compute $\mathbf{u}$ and M from
$$\mathbf{X} = (u_1, u_2, M_{11} - 1, M_{22} - 1, M_{21}, M_{12})^T.$$

2. Compute polar angles α, β:
$$\alpha = \arctan(|\mathbf{u}|/f), \quad \beta = \arctan(\mathbf{u}).$$

3.
$$M' = S_x(\sec\alpha)R(-\beta)M \quad \text{where} \quad S_x(\mu) \equiv \begin{pmatrix} \mu & 0 \\ 0 & 1 \end{pmatrix}$$
($R(-\beta)$ is represented here as a 2×2 matrix.)

4. Apply steps 2–7 of the orthographic algorithm of figure 7.1 to M' (in place of M), giving R', Z_c' (in place of R, Z_c).

5.
$$R = R_z(\beta)R_y(\alpha)\ R'.$$
(These rotation matrices are now all 3×3)

6. Translation:
$\mathbf{R}_c = (X_c, Y_c, Z_c)^T$ where $Z_c = Z_c' \cos\alpha, \quad X_c = u_1 Z_c/f,\ Y_c = u_2 Z_c/f.$

Figure 7.6: Recovery of pose of a plane under weak perspective.

outlined in figure 7.6. First, express the image-translation vector $\mathbf{u}$ in terms of polar angles

$$\alpha, \ -\frac{\pi}{2} \le \alpha \le \frac{\pi}{2} \ \text{ and } \ \beta, \ -\pi \le \beta \le \pi,$$

that is, so that

$$\mathbf{u} = f \tan\alpha \begin{pmatrix} \cos\beta \\ \sin\beta \end{pmatrix}. \tag{7.2}$$

The next step 3 is to replace the affine matrix M by the affine matrix M' on the virtual orthographic plane. Then the orthographic pose-recovery algorithm is applied to M' in place of M, to give a three-dimensional rotation matrix R' (subject, of course, to reversal ambiguity) and translation in depth Z'_c, both in a virtual orthographic frame. Finally, these parameters are transformed back into the camera coordinate frame in steps 5 and 6.

7.2 Pose recovery for three-dimensional objects

The planar affine space survives intact as a subspace in the three-dimensional affine case (see (4.17) on page 89), so the pose-recovery method for the planar case continues to be usable, in principle, for the three-dimensional problem. Coefficients of the first 6 elements of the configuration vector $\mathbf{X}$ could be used to obtain $\mathbf{u}, M$ as before and the planar pose-recovery algorithm would compute pose $\mathbf{R}_c, R$ correctly, subject to the usual ambiguities. However, the planar algorithm does not make optimal use of the information now available. The additional pair of shape parameters contain valuable *parallax* information which is complementary to the information available in the planar case. The planar algorithm is at its least accurate close to the standard pose. The extreme case is the singularity at the standard pose itself, which gives rise to the ambiguity in orientation discussed earlier. In contrast, the additional parallax information is at its most useful precisely at the singularity, improving accuracy of recovered pose, and removing ambiguity, as figure 7.7 shows.

Once $\mathbf{Q}_0^z$ has been estimated for a given object (see below), the recovery of pose from some outline $\mathbf{X}$ in an image of the object is relatively straightforward — the algorithm is given in figure 7.8. Step 1 computes planar affine parameters $\mathbf{u}, M$ and parallax parameter $\mathbf{v}$ from the shape-vector. Recovery of R, Z_c in steps 2 and 3 is based on solving (4.16) on page 89, expressing R in terms of its rows R_1, R_2, R_3, which must be unit vectors, and mutually orthogonal. The matrix $\hat{R}$ resulting from step 3 need not be precisely orthogonal so step 4 finds the closest orthogonal matrix by "singular value decomposition" (appendix A.1) of $\hat{R}$. Finally, translation is calculated as in the planar affine case.

Figure 7.9 illustrates that the pose of a three-dimensional object can be recovered effectively over a wide range of views, using parallax as in the algorithm above.

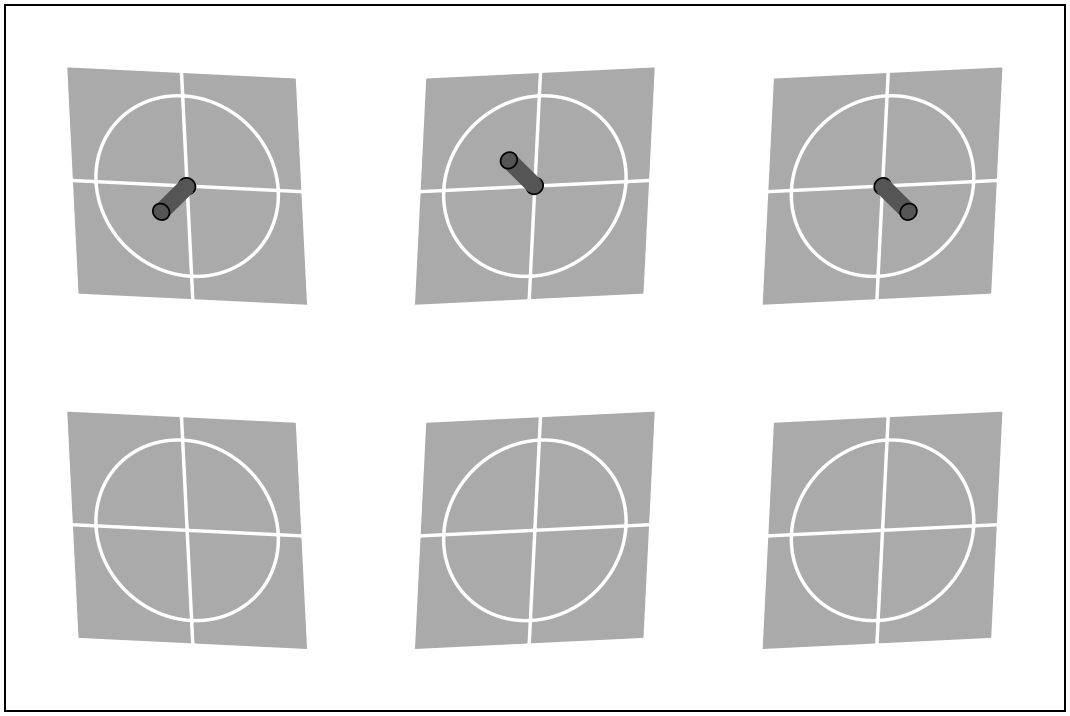


Figure 7.7: Parallax in pose recovery. *The poses of the top row of wheels, complete with axles, are easily discriminable. When parallax information is omitted by removing the axles, as on the bottom row, the orientations of the first two wheels are discriminable with difficulty and the last two actually appear identical because of the planar orientation ambiguity phenomenon.*

Estimating $\mathbf{Q}_0^z$

In principle, it would be possible to rotate the object, from its standard view, through 90^o about the y-axis, and the new view would be

$$\begin{pmatrix} \mathbf{Q}_0^x \\ \\ \mathbf{Q}_0^z \end{pmatrix}$$

from which $\mathbf{Q}_0^z$ could be obtained. This would be possible with a bent wire object, but with a solid object such as the leaves above only a modest angle of rotation is possible without alterations in hidden line structure. Therefore, practically, a method

1. Compute $\mathbf{u}$, M and $\mathbf{v}$ from the shape-vector $\mathbf{X}$, using:
$$\mathbf{X} = (u_1, u_2, M_{11} - 1, M_{22} - 1, M_{21}, M_{12}, v_1, v_2).$$

2. $$Z_c = 2f\left[|(M_{11}, M_{12}, v_1)| + |(M_{21}, M_{22}, v_2)|\right]^{-1}$$

3. $$\hat{R}_3 = \hat{R}_1 \times \hat{R}_2 \quad \text{where} \quad \begin{pmatrix} \hat{R}_1 \\ \hat{R}_2 \end{pmatrix} = \frac{Z_c}{f}\left(M \mid \mathbf{v}\right).$$

4. Enforce orthogonality:
$$R = UV \quad \text{where} \quad \hat{R} = UDV$$
(in which D should be approximately the identity I_3).

5. Translation: $\mathbf{R}_c = (X_c, Y_c, Z_c)^T$ where $X_c = u_1 Z_c/f$, $Y_c = u_2 Z_c/f$.

Figure 7.8: Recovery of pose using parallax.

is required for estimating $\mathbf{Q}_0^z$ from a rotation R through a modest angle θ about an axis lying approximately parallel to the image plane. It is assumed that the object does not translate significantly in depth (Z) during the rotation.

The rotated view $\mathbf{r}_1(s) = U(s)\mathbf{Q}_1$ is a transformed version of standard views:

$$\mathbf{r}_1(s) = \frac{f}{Z_c}\left[\mathbf{u}_1 + R_{2\times 3}\begin{pmatrix} X_0(s) \\ Y_0(s) \\ Z_0(s) \end{pmatrix}\right],$$

in which $Z_0(s)$ is as yet unknown. The translation $f\mathbf{u}_1/Z_c$ can be calculated simply

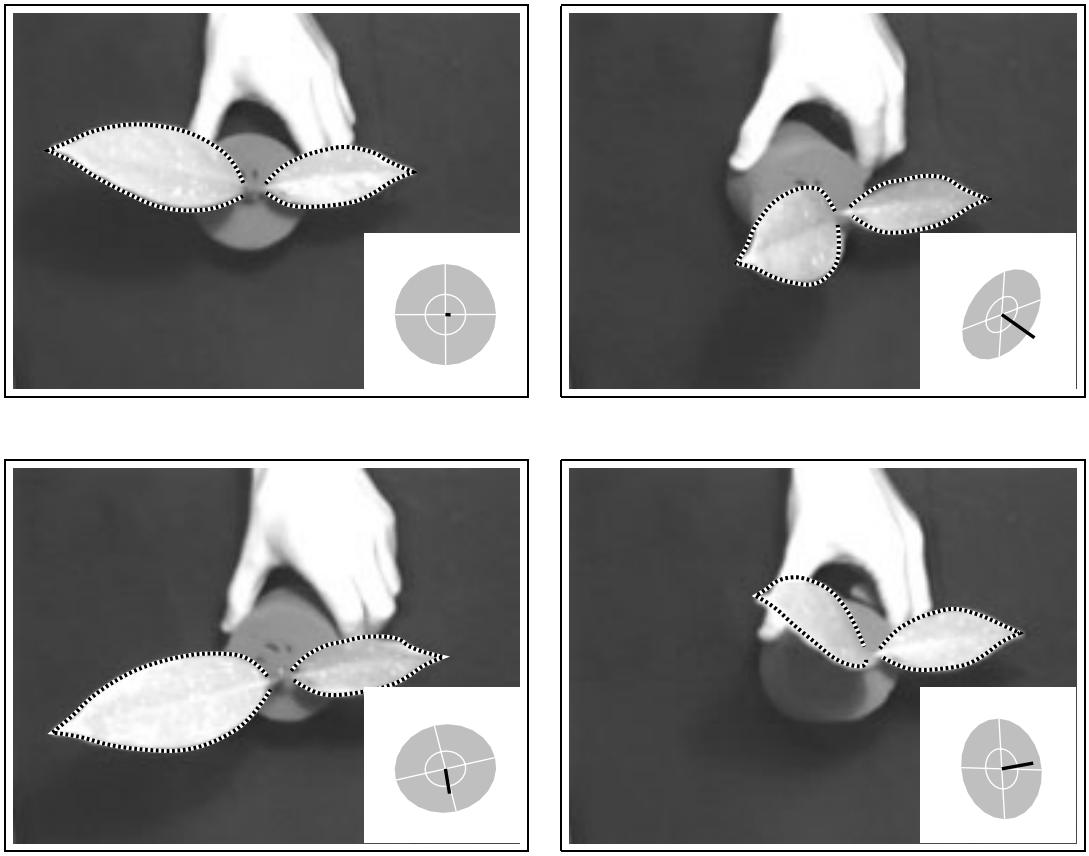


Figure 7.9: Pose recovery using the three-dimensional affine algorithm. *The pose can be accurately recovered over a wide range of transformations, including those near the planar affine degeneracy at the standard pose.*

as the centroid

$$\frac{f}{Z_c}\mathbf{u}_1 = \overline{\mathbf{r}}_1 = \begin{pmatrix} \mathbf{Q}_1^x \cdot \mathbf{1} \\ \\ \mathbf{Q}_1^y \cdot \mathbf{1} \end{pmatrix}.$$

Then $Z_0(s)$ can be estimated from

$$(\mathbf{a}^T\mathbf{a})Z_0(s) = \mathbf{a}^T \left[\left(\frac{Z_c}{f}\mathbf{r}_1(s) - \mathbf{u}_1 \right) - R_{2\times 2}\mathbf{r}_0(s) \right]$$

in which $\mathbf{a} = (R_{13}, R_{23})^T$. The resulting $Z_0(s)$ is a spline that can be expressed in terms of its control points as

$$\mathbf{Q}_0^z = \frac{1}{\mathbf{a}^T\mathbf{a}}(\mathbf{a}^T \otimes I)\left[\left(\frac{Z_c}{f}\mathbf{Q}_1 - \mathbf{u}_1 \otimes \mathbf{1}\right) - (R_{2\times 2} \otimes I)\,\mathbf{Q}_0\right] \tag{7.3}$$

where I is the $N_B \times N_B$ identity matrix and $\otimes$ is the "Kronecker product" operation (see appendix A.1).

A more accurate method

If the rotation matrix R in (7.3) is not accurately known, $\mathbf{Q}_0^z$ will be imperfectly estimated. The quality of the estimate can be improved by using more views and a more sophisticated algorithm. Suppose the views are $(\mathbf{Q}_n^x, \mathbf{Q}_n^y),\ n = 1, \ldots, N$, then the following simultaneous equations hold:

$$\begin{pmatrix} \mathbf{Q}_n^x \\ \mathbf{Q}_n^y \end{pmatrix} - \frac{f}{z_n}\left\{\begin{pmatrix} x_n\mathbf{1} \\ y_n\mathbf{1} \end{pmatrix} + T_n \begin{pmatrix} \mathbf{Q}_0^x \\ \mathbf{Q}_0^y \\ \mathbf{Q}_0^z \end{pmatrix}\right\} = 0 \tag{7.4}$$

where $(x_n, y_n, z_n, T_n),\ n = 1, \ldots, N$ and $\mathbf{Q}_0^z$ are unknown. Here (x_n, y_n, z_n) is the object translation in view n and T_n is the 2×3 rotation matrix. If f is only known approximately, it can also be treated as an unknown in the system. The unknowns can be estimated using any non-linear minimisation method, for example conjugate gradient descent. Convergence is quick, and likely to find the correct local minimum, if the unknowns are initialised approximately. An alternative linear algorithm which does not rely on careful initialisation, is the "image-stream factorisation" algorithm of Tomasi and Kanade. This works by approximating the image stream, laid out as a matrix

$$\begin{pmatrix} \mathbf{Q}_1^x & \mathbf{Q}_1^y & \mathbf{Q}_2^x & \mathbf{Q}_2^y & \ldots \end{pmatrix}$$

to have its theoretical rank (4 in this case), using singular value decomposition. Once this is done, $\mathbf{Q}_0^z$ could be calculated from the approximated image stream, subject to a certain fundamental affine indeterminacy.

7.3 Separation of rigid and non-rigid motion

In chapter 4 shape-spaces for combined rigid and non-rigid motion were constructed using key-frames. Such spaces are often too big for efficient or robust shape-fitting;

they prove still to be useful for interpretation of shape, decomposing displacements into rigid and non-rigid components. This is done by projecting a fitted shape $\mathbf{Q}$ onto the shape-space and applying "Singular Value Decomposition" (SVD).

The decomposition algorithm is given in figure 7.10 (and justified below). The

Algorithm:

Given a fitted spline curve $\mathbf{Q}$:

1. Express $\mathbf{Q}$ in the key-frame basis:
$$\left(u_1, u_2, \{\hat{Y}_i^j,\ i = 0, 1, \ldots,\ j = 1, 2, \ldots\}\right) = W^+\mathbf{Q},$$
where W is as defined in (4.21) on page 92.

2. Treating $\hat{Y}_i^j$ as components of a rectangular matrix $\hat{Y}$, apply SVD:
$$\hat{Y} = UDV.$$

3. Approximate Y as a rank 1 matrix
$$Y = UD^*V$$
where D^* is D with all but the largest singular value set to 0.

4. Recovered pose is given by the shape-vector
$$\mathbf{X} = (u_1, u_2, {Y_0}^1, {Y_0}^2, \ldots)^T.$$

5. Recovered non-rigid deformation is expressed as a contour in standard pose:
$$\mathbf{Q}^d = \mathbf{Q}_0 + \sum_i \lambda_i \mathbf{Q}_i \quad \text{where} \quad \lambda_i = \frac{{Y_i}^1}{{Y_0}^1},\ i = 1, 2, \ldots$$
and $\mathbf{Q}_i$ are key-frames as before.

Figure 7.10: Algorithm for decomposition of rigid and non-rigid displacement.

algorithm is illustrated in figure 7.11 for the problem of separating facial expression from head pose. Rigid transformations are modelled here as 3D affine, including

Figure 7.11: Pose-invariant transmission of facial expression. *Separation of non-rigid from rigid motion by SVD is used here to extract the facial expression of an actor. The extracted expression is displayed on this cat caricature in a fixed pose, and can be seen to be independent of the pose of the actor's head. (Figure courtesy of Benedicte Bascle.)*

parallax to accommodate the modest departure from co-planarity of tracked eyebrows, mouth and facial creases. Expression is represented in three-dimensional coordinates whose axes are defined by key-frames for smile, surprise and disgust, relative to a neutral expression.

Derivation of decomposition algorithm. Shape parameters ${Y_i}^j$, $i = 0, \ldots, N_k$, $j = 1, \ldots, N_r$ are highly interdependent, in principle, because, given the structure of shape-space in (4.21) on page 92, Y is a product of two vectors:

$$ {Y_i}^j = \lambda_i X_{j+2}, $$

where $\lambda_0 = 1$ is the weight of the template $\mathbf{Q}_0$ and λ_i is the weight of key-frame $\mathbf{Q}_i$ for $i = 1, \ldots, N_k$. (The index $_{j+2}$ is an offset to allow for two image-translation parameters.) Thus Y should have rank 1, and this is enforced by approximating $\hat{Y}$ to rank 1 in the SVD step. Note that one should expect D to contain one dominant singular value, indicating that Y is a good approximation to Y^*.

Bibliographic notes

Recovery of the pose of an object from its shape-space vector is based on an eigenvalue method for recovery of the pose of a textured surface in (Blake and Marinos, 1990). The "image-stream factorisation" algorithm (Tomasi and Kanade, 1991) uses singular value decomposition (SVD) (Barnett, 1990) of the image stream for motion analysis. Here, an adaptation of the algorithm was suggested for calibration of pose recovery using parallax. The algorithm for decomposition of rigid and non-rigid displacement is yet another application of SVD to bilinear decomposition problems in vision, others being structure and motion (Tomasi and Kanade, 1991), and shape and shading (Freeman and Tenenbaum, 1997).