CHAPTER

Object segmentation and shape models

12

Object segmentation can be carried out using raw thresholding or edge detection, but such processes are liable to stray far-off course because of random image artifacts such as noise and shadows. The idea of using shape models is to control the situation so that the segmentations obtained are more meaningful. Active contour models or "snakes" provide a means for tackling this problem: principal components analysis (PCA)-based methods provide another.

Look out for:

- the need for shape models during segmentation
- the use of active contour models (snakes) for segmenting object boundaries
- the principle of energy minimization
- problems in handling snake evolution: model constraints
- the "level set" approach to object segmentation
- the "fast-marching" and "front-propagation" methods
- the use of PCA for learning shape models
- how the list of eigenvalues is limited
- how the system is trained using landmark points
- speeding up search by implementing it at various scales to improve matching
- improving on the Mahalanobis distance model using gray-scale boundary profiles.

The snake approach is subject to difficulties because all the necessary rules have to be incorporated into the snake analytically without acknowledging exactly what type of shape it will have: PCA-based methods are more powerful as they train the system to know exactly what type of shape is being sought. However, they involve substantial amounts of complication and computation, though as a result, they can extract objects from regions of considerable noise.

12.1 INTRODUCTION

This chapter follows on naturally from Chapter 5 on edge detection. It takes as its basis the idea that edges have already been found in the image and that they need to be connected up to form objects in a systematic way. In fact, it is not

Computer Vision. DOI: http://dx.doi.org/10.1016/B978-0-12-809284-2.00012-5

absolutely necessary to find the edge points first, so long as they can be found on the fly as a shape model develops. Indeed, the snake "active contour" approach can adopt the latter strategy—in which case, the snake model is forced to grow in a systematic way until it is found to match the edges that are present in the picture. As we shall see in the following section, snakes are made to be subject to rules that force them to develop under the influence of energy minimization and other constraints. That approach and the ensuing level-set technique are guided by general segmentation principles but do not include exact models of the shapes that are to be found. On the other hand, alternative approaches such as PCA are available for generating shape models: these act more like recognition algorithms than segmentation algorithms. In addition, they have become extremely widely used and deserve to be covered in fair depth. We start with a basic description of the active contour approach.

12.2 ACTIVE CONTOURS

Active contour models (also known as "deformable contours" or "snakes") are widely used for systematically refining object contours. The basic concept is to obtain a complete and accurate outline of an object that may be ill-defined in places, whether through lack of contrast, or noise or fuzzy edges. A starting approximation is made, either by instituting a large contour that may be shrunk to size, or a small contour that may be expanded suitably, until its shape matches that of the object. In principle, the initial boundary can be rather arbitrary, whether mostly outside or within the object in question. Then, its shape is made to evolve subject to an energy minimization process: on the one hand, it is desired to minimize the *external* energy corresponding to imperfections in the degree of fit; on the other hand, it is desired to minimize the *internal* energy, so that the shape of the snake does not become unnecessarily intricate, e.g., by taking on any of the characteristics of image noise. There are also model constraints that are represented in the formulation as contributions to the external energy: typical of such constraints is that of preventing the snake from moving into prohibited regions, such as beyond the image boundary, or, for a moving vehicle, off the region of the road.

The snake's internal energy includes elastic energy which might be needed to extend or compress it, and bending energy. If no bending energy terms were included, sharp corners and spikes in the snake would be free to occur with no restriction. Similarly, if no elastic energy terms were included, the snake would be permitted to grow or shrink without penalty.

The image data is normally taken to interact with the snake via three main types of image feature—lines, edges, and terminations (the latter can be line terminations or corners). Various weights can be given to these features according to the behavior required of the snake. For example, it might be required to hug edges and go around corners, and only to follow lines in the absence of edges: so the line weights would be made much lower than the edge and corner weights.

These considerations lead to the following breakdown of the snake energy:

$$\begin{aligned} E_{\text{snake}} &= E_{\text{internal}} + E_{\text{external}} \\ &= E_{\text{internal}} + E_{\text{image}} + E_{\text{constraints}} \\ &= E_{\text{stretch}} + E_{\text{bend}} + E_{\text{line}} + E_{\text{edge}} + E_{\text{term}} + E_{\text{repel}} \end{aligned} \tag{12.1}$$

The energies are written down in terms of small changes in position $\mathbf{x}(s) = (x(s), y(s))$ of each point on the snake, the parameter s being the arc length distance along the snake boundary. Thus, we have:

$$E_{\text{stretch}} = \int \kappa(s) \left\| \mathbf{x}_s(s) \right\|^2 ds \tag{12.2}$$

and

$$E_{\text{bend}} = \int \lambda(s) \left\| \mathbf{x}_{ss}(s) \right\|^2 ds \tag{12.3}$$

where the suffices s, ss imply first and second order differentiation, respectively. Similarly, E_{edge} is calculated in terms of the intensity gradient magnitude $\left| grad\ I \right|$, leading to:

$$E_{\text{edge}} = - \int \mu(s) \left\| grad\ I \right\|^2 ds \tag{12.4}$$

where $\mu(s)$ is the edge weighting factor.

The overall snake energy is obtained by summing the energies for all positions on the snake: a set of simultaneous differential equations is then set up to minimize the total energy. Space prevents a full discussion of this process here. Suffice it to say that the equations cannot be solved analytically, and recourse has to be made to iterative numerical solution, during which the shape of the snake evolves from some high energy initialization state to the final low energy equilibrium state, defining the contour of interest in the image.

In the general case, there are several possible complications to be tackled, which are as follows:

1. several snakes may be required to locate an initially unknown number of relevant image contours;
2. different types of snake will need different initialization conditions;
3. snakes will sometimes have to split up as they approach contours that turn out to be fragmented.

There are also procedural problems. The intrinsic snake concept is that of well-behaved differentiability. However, lines, edges, and terminations are usually highly localized, so there is no means by which a snake even a few pixels away could be expected to learn about them and hence to move toward them. In these circumstances, the snake would "thrash around," and fail to systematically zone in on a contour representing a global minimum of energy. To overcome this problem, smoothing of the image is required, so that edges can communicate with the snake some distance away, and the smoothing must gradually be reduced as the

snake nears its target position. Ultimately, the problem is that the algorithm has no high-level appreciation of the overall situation but merely reacts to a conglomerate of local pieces of information in the image: this makes segmentation using snakes somewhat risky despite the intuitive attractiveness of the concept.

In spite of these potential problems, a valuable feature of the snake concept is that, if set up correctly, the snake can be rendered insensitive to minor discontinuities in a boundary: this is important, as this makes it capable of negotiating practical situations such as fuzzy or low contrast edges, or places where small artifacts get in the way (this may happen with resistor leads for example); this capability is possible because the snake energy is set up *globally*—quite unlike the situation for boundary tracking where error propagation can cause wild deviations from the desired path. The reader is referred to the abundant literature on the subject, not only to clarify the basic theory (Kass and Witkin, 1987; Kass et al., 1988) but also to find how it may be made to work well in real situations.

12.3 PRACTICAL RESULTS OBTAINED USING ACTIVE CONTOURS

In this section, we briefly explore a simple implementation of the active contour concept. Arguably, the implementation chosen is amongst the simplest that will work in practical situations while still adhering to the active contour concept. To make it work without undue complication or high levels of computation, a "greedy" algorithm is used—that is, one that makes local optimizations (energy minimizations) in the expectation that this will result in global optimization. Naturally, it could lead to solutions that do not correspond to absolute minima of the energy function, though this is by no means a problem that is caused solely by using a greedy algorithm, as almost all forms of iterative energy minimization method can fall into this trap.

The first thing to do when devising such an algorithm is to interpret the theory in practical terms. Thus, we rewrite the snake stretch function (Eq. (12.2)) in the discrete form:

$$E_{\text{stretch}} = \sum_{i=1}^{N} \kappa \left\| \mathbf{x}_i - \mathbf{x}_{i+1} \right\|^2 \tag{12.5}$$

where there are N snake points $\mathbf{x}_i$, $i = 1, \ldots, N$: note that this set must be accessed cyclically. In addition, when using a greedy algorithm and updating the position of the ith snake point, the following local form of Eq. (12.5) has to be used:

$$\varepsilon_{\text{stretch},\, i} = \kappa \left(\left\| \mathbf{x}_i - \mathbf{x}_{i-1} \right\|^2 + \left\| \mathbf{x}_i - \mathbf{x}_{i+1} \right\|^2 \right) \tag{12.6}$$

Unfortunately, although this function causes the snake to be tightened, it can also result in clustering of snake points. To avoid this, the following alternative form can be useful:

$$\varepsilon_{\text{stretch},\, i} = \kappa \left[(d - \left\| \mathbf{x}_i - \mathbf{x}_{i-1} \right\|)^2 + (d - \left\| \mathbf{x}_i - \mathbf{x}_{i+1} \right\|)^2 \right] \tag{12.7}$$

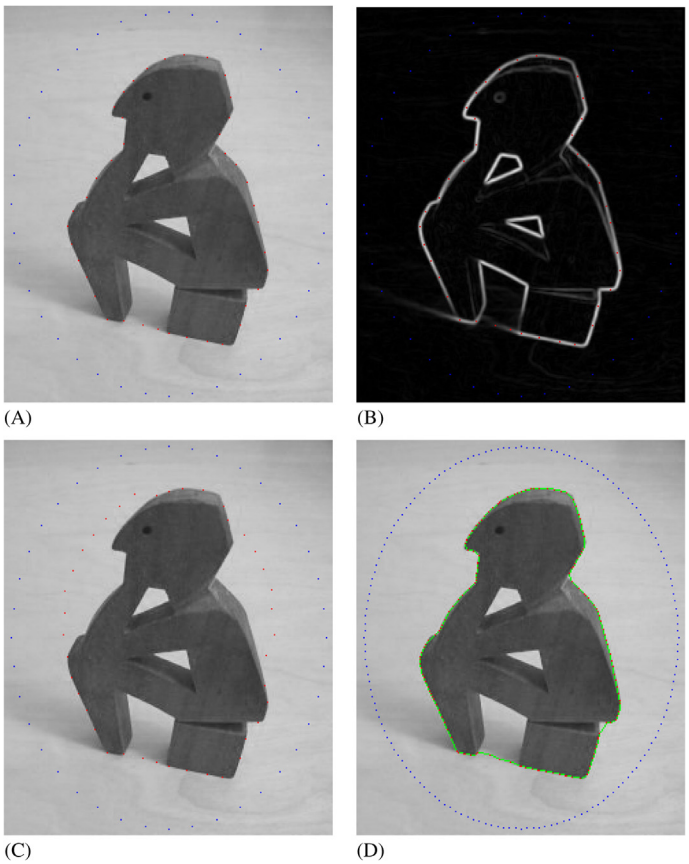

FIGURE 12.1

Generation of active contour models (snakes). (A) Original picture with snake initialization points (blue) near the image boundary; the final snake locations (red) hug the outside of the object but hardly penetrate the large concavity at the bottom: they actually lie approximately along a weak shadow edge. (B) Result of smoothing and application of Sobel operator to (A); the snake algorithm used this image as its input. The snake output is superimposed in (red) on (B), so that the high degree of colocation with the edge maxima can readily be seen. (C) Intermediate result, after half (30) the total number of iterations (60): this illustrates that, after one edge point has been captured, it becomes much easier for other such points to be captured. (D) Result of using an increased number of initialization points (blue) and joining (green) the final locations (red) to give a connected boundary: some remanent deficiencies are evident.

where d is a fixed number representing the smallest likely value of the mean distance between adjacent pairs of snake points, for the given type of target object. In the implementation used in Fig. 12.1, d had the noncritical value of 8 pixels; interestingly, this also resulted in faster convergence toward the final form of the

snake, as it was encouraged to move further to minimize the magnitudes of the terms in round brackets.

The contour shown in Fig. 12.1 fills the concavity at the top right, but hardly moves into the concavity at the bottom because of a low contrast shadow edge: note that more or less influence by weak edges can readily be obtained by adjusting the grad^2 coefficient μ in Eq. (12.4). Elsewhere the snake ends up with almost exact adherence to the object boundary. The snake shown in the figure employs $p = 40$ points, and $r = 60$ iterations are needed to bring it to its final position. In each iteration, the greedy optimization for each snake point is over an $n \times n$ pixel region with $n = 11$. Overall, the computation time is controlled by and essentially proportional to the quantity prn^2.

The final contour in Fig. 12.1D shows the result of using an increased number of initialization points and joining the final locations to give a connected boundary: some of the remaining deficiencies could be reduced by fitting with splines or other means instead of simply joining the dots.

As indicated earlier, this was a simple implementation—so much so that no attempt was made to take corners and bends into account, though in the case shown in Fig. 12.1, no disadvantages or deviations can be seen, except in (D). Clearly, a suitable redesign involving additional energy terms would have to be included to cope with more complex image data. It is interesting that so much can be achieved by just two terms, viz. the stretch and edge terms in Eq. (12.1). However, an important factor in getting the greedy algorithm to work optimally, one snake point at a time is the need to include the energies for *both* adjacent links (as in Eqs. (12.6) and (12.7)), so as to limit bias and other complications.

12.4 THE LEVEL-SET APPROACH TO OBJECT SEGMENTATION

Although the active contour approach described in the previous two sections can be effective in many situations, it nevertheless has several drawbacks (Cremers et al., 2007), which are as follows:

1. There is the possibility of snake self-intersection.
2. Topological changes like splitting or merging of the evolving contour are not allowed.
3. The algorithm is highly dependent on the initialization, and this can result in the snake being biased or getting stuck in local minima.
4. Snakes lack meaningful probabilistic interpretation, so generalizing their action to cover color, texture, or motion is not straightforward.

The level-set approach is intended to remedy these deficiencies. The basic approach is to work with whole regions rather than edges and to evolve an

"embedding function" in which contours are represented implicitly rather than directly. In fact, the embedding function is a function $\phi(\mathbf{x}, t)$, and the contour is defined as the zero level of this function:

$$C(t) = \{\mathbf{x} | \phi(\mathbf{x}, t) = 0\} \tag{12.8}$$

For a contour which evolves (by gradient descent) along each local normal $\mathbf{n}$ with a speed F, we have:

$$\phi(C(t), t) = 0 \tag{12.9}$$

which leads to

$$\frac{\mathrm{d}}{\mathrm{d}t}\phi(C(t), t) = \nabla\phi\frac{\partial C}{\partial t} + \frac{\partial \phi}{\partial t} = F\,\nabla\phi\cdot\mathbf{n} + \frac{\partial \phi}{\partial t} = 0 \tag{12.10}$$

Substituting for $\mathbf{n}$ using:

$$\mathbf{n} = \frac{\nabla\phi}{|\nabla\phi|} \tag{12.11}$$

we obtain:

$$\frac{\partial \phi}{\partial t} = -|\nabla\phi|F \tag{12.12}$$

Next, we need to substitute for F. Following Caselles et al. (1997), we have:

$$\frac{\partial \phi}{\partial t} = |\nabla\phi| div\left(g(I)\frac{\nabla\phi}{|\nabla\phi|}\right) \tag{12.13}$$

where $g(I)$ is a generalized version of $|\nabla\phi|$ in the snake potential.

Note that because the contour C is not mentioned explicitly, the updating takes place over all pixels, thereby involving a great many useless calculations: thus, the "narrow band" method was devised to overcome this problem and involves updating only in a narrow strip around the current contour. However, the need to continually update this strip means that the computational load remains considerable. An alternative approach is the "fast-marching" method, which essentially propagates a solution rapidly along an active wavefront, while leaving pixel values frozen behind it. As a result, this method involves maintaining the sign of the speed values F. The Hermes algorithm of Paragios and Deriche (2000) seeks to combine the two approaches. It aims at a *final* solution where all the necessary constraints are fulfilled, while maintaining these constraints only loosely at intermediate stages. The overall front propagation algorithm overcomes the four problems mentioned above: in particular, it is able to track nonrigid objects, copes with splitting and merging, and has low computational cost. The paper confirms these claims by showing traffic scenes in which vehicles and pedestrians are successfully tracked.

Rather than developing this method further here, we move on to describe a rather different and exceedingly widely used approach in which shape models are explicitly trained using PCA.

12.5 SHAPE MODELS

In order to model shapes, it is necessary to work in a specific representation. The representations used for boundary pattern analysis in Chapter 9, Boundary Pattern Analysis included (r, θ), (s, ψ), and (s, κ) boundary plots. Although such plots can be further modeled by Fourier methods, this can be cumbersome, and the complications conferred by translation, rotation, and shape distortions make it preferable to retreat from representation by continuous boundaries to representation by sets of discrete boundary points. This leads to advantages when moving from low to high resolution, in that, the boundary can still be represented by the same number of points in the same relative positions. Furthermore, if the representation is discrete, methods like PCA can be used to process the data: as we shall see below, this is of particular value when a training approach is to be used to learn shapes and obtain minimal descriptors of them. It is also relevant that PCA involves considerable computation, but nowadays that is a far less important consideration than it was 15−20 years ago; hence, if a method is accurate, robust, trainable, and provides valuable information as part of its normal running, then it should be considered for serious use. In fact, what we need to keep in mind here as part of the "valuable information" that is provided is the capability for eliminating the later (smaller) terms in a series—namely, the low energy eigenvalues that correspond largely to noise—and thereby reducing the redundancy and computational burden later on.

To implement shape models based on discrete boundary points, we need to start with definitive information on the models themselves. To achieve this, "landmark" points are marked around the shapes of interest, often with the aid of a mouse. Although this approach has obvious limitations in terms of human labor, it can nevertheless be highly accurate and reliable when small numbers of points are needed to define shapes. In addition, it has particular value when medical images are to be analyzed, as these are often rather fuzzy and require expert interpretation, consistency being of fundamental importance when using a computer algorithm to analyze them. In any case, once a few highly specific landmark points landmark points have been accurately identified, further points can often be derived with virtually no additional human interaction. In what follows we illustrate this using a series of hand images derived from a short motion video.

Fig. 12.2A shows a picture of a human hand, and Fig. 12.2B shows nine landmark points that have been marked on it using a mouse: these include five fingertip points ($t_1 - t_5$) and four finger-base points ($b_1 - b_4$). Fig. 12.2C shows how a further set of five mid-finger points ($m_1 - m_5$) can be derived, using the white construction lines. Numbering the three sets of points from the top, we can write down the following vector relations giving the positions of $m_1 - m_5$ as

$$\mathbf{m}_1 = \mathbf{b}_1 + c(\mathbf{b}_1 - \mathbf{b}_4) \tag{12.14}$$

$$\mathbf{m}_2 = \mathbf{b}_2 + \frac{1}{2}(\mathbf{b}_2 - \mathbf{b}_3) \tag{12.15}$$

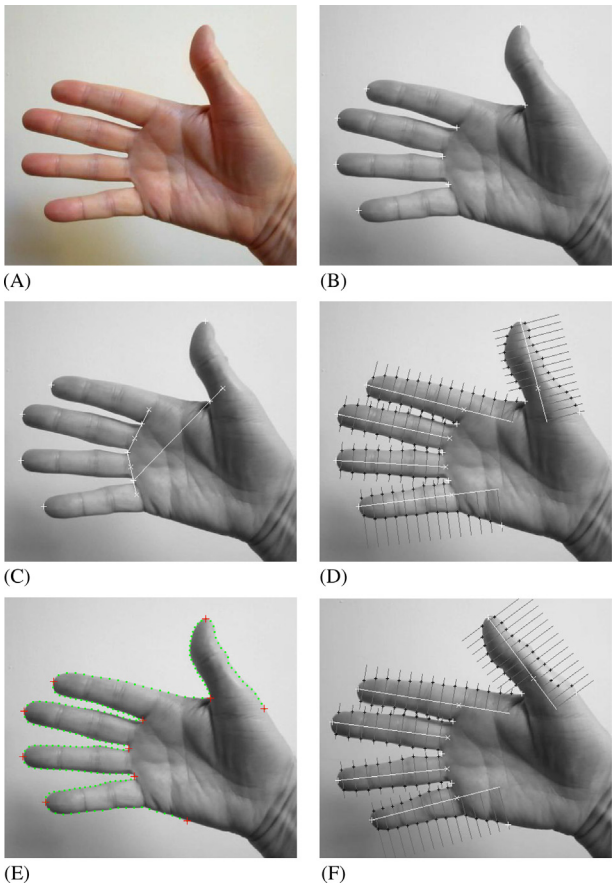

FIGURE 12.2

Producing boundary features for PCA analysis. (A) Original image of a hand. (B) Result of using a mouse to generate landmark points at the tips and bases of the fingers. (C) The geometric constructions used to generate mid-base finger locations. (D) White lines generated by joining finger tips to base points and producing; dark lines perpendicular to the white lines used to locate boundary points. (E) Boundary points interpolated and equally spaced boundary points found. (F) A shiny light patch leading to misplaced boundary points, which PCA has to be able to cope with. See text for further details of the various stages.

$$\mathbf{m}_3 = \frac{1}{2}(\mathbf{b}_2 + \mathbf{b}_3) \tag{12.16}$$

$$\mathbf{m}_4 = \frac{1}{2}(\mathbf{b}_3 + \mathbf{b}_4) \tag{12.17}$$

$$\mathbf{m}_5 = \mathbf{b}_4 + \frac{1}{2}(\mathbf{b}_4 - \mathbf{b}_3) \tag{12.18}$$

Of these, the formulae for m_3 and m_4 are obviously appropriate, and those for m_2 and m_5 are also reasonable: although in principle, the last of these could be adjusted for the smaller width of the little finger, this doesn't seem to be necessary in practice. The formula for m_1 is more problematic, but it works well if parameter c is taken to be 1/6. It should be remarked that this vector method is based on the assumption that the shape of the hand will remain approximately planar, thereby making affine stretching acceptable: on the other hand, if intricate 3-D stretching or clenching were to take place, it would not yield good results, and many more landmark points would be needed.

The next stage is to construct five mid-finger lines from t_i to m_i ($i = 1$ to 5), produced as necessary to cover the extended sides of fingers 1, 2, and 5. These lines are divided into equal sections, and perpendiculars are dropped from them far enough to enter the background regions. This permits a number of approximately equally spaced finger boundary points to be located, as shown in Fig. 12.2D. Finally, these boundary points are interpolated and smoothed by suitable algorithms, and sets of equally spaced points are found on the finger boundaries (Fig. 12.2E). The purpose of this is to ensure that the boundary points are as accurately reproducible as possible, and thus that they match corresponding points on similar hand shapes. In general, none of the points are more accurately located than the original nine landmark points, but at least the boundary points obtained (114 for each hand in the set) are located accurately without further human intervention. Notice that, in some cases, lack of boundary contrast, coupled with shiny light patches particularly within the thumb, resulted in a few undesirable deviations in the boundary points (Fig. 12.2F). However, as we shall see, PCA was able to average out effects such as these: no efforts were made in these tests to eliminate such effects before passing the boundary points to the PCA algorithm, though it would have been reasonably straightforward to do so (e.g., a median-based outlier detector together with a suitable interpolation scheme could have been used).

Before applying PCA, it is necessary to eliminate translation and rotation parameters. First, translation is eliminated by subtracting the mean location of all the boundary points. Then, the mean orientation of all the boundary points is found, and this is subtracted from all the orientations, and new coordinates for the boundary points are determined. These actions are achieved by using the arctan function (specifically the C++ or Matlab "atan2" function), followed by cosine and sine functions to reconstruct the new object coordinates. It is important to remember that averaging angles directly can lead to meaningless results, because of their periodicity (this comment also applies for median filtering in the rotationally symmetric hue domain, and in many other such cases): the most rigorous approach is to convert all points to the unit circle, then average their sines and cosines, and then deduce the mean direction. Finally, it is also useful to normalize object sizes by minimizing the sum of square distances between the model and the new image points. These normalizations and alignments are aimed at removing variations from the incoming data and making the work of the PCA algorithm

easier, and the final model independent of object position, orientation, and size. Put another way, it makes the resulting point distributions closer to Gaussian—which would be the case in an ideal PCA implementation.

Returning to our hand problem, and applying PCA to the 17 sets of 114 boundary points, the results shown in Table 12.1 were obtained. It is common to curtail the list of eigenvalues when the cumulative sum *csum* is more than η times the maximum, η typically being in the range 90%−99.5%, depending on the specific type of image data. Cootes and Taylor (1996), who originally developed the active shape model (ASM) approach, normally used a value of 98%. Applying this to the *csum* values listed in Table 12.1, we find that the first five eigenvalues should be retained. In fact, looking at the hand model modes shown in Fig. 12.3, five is a reasonable number to take, as the image variations resulting from smaller modes are almost undetectable. We can get further insight into the situation by examining Fig. 12.4. Here, the modes are not multiplied by the eigenvalue weights (which are $\pm\sqrt{\lambda}$ rather than $\pm\lambda$) but are multiplied instead by a fixed value of 100, so that their underlying nature can be seen. Certainly, the largest eigenvalues represent interesting coherent motions of the fingers, but after the

Table 12.1 Eigenvalues for 17 Hand Images

Order	λ	*csum*	$\sqrt{\lambda}$	$\sqrt{\lambda/n}$
1	36,590.6	36,590.6	191.3	17.9
2	8400.7	44,991.3	91.7	8.6
3	4572.9	49,564.2	67.6	6.3
4	868.1	50,432.3	29.5	2.8
5	687.0	51,119.3	26.2	2.5
6	388.3	51,507.6	19.7	1.8
7	156.9	51,664.5	12.5	1.2
8	103.8	51,768.3	10.2	1.0
9	81.0	51,849.3	9.0	0.8
10	44.9	51,894.2	6.7	0.6
11	32.2	51,926.4	5.7	0.5
12	23.2	51,949.6	4.8	0.5
13	16.7	51,966.2	4.1	0.4
14	10.4	51,976.6	3.2	0.3
15	7.0	51,983.6	2.6	0.2
16	5.5	51,989.1	2.4	0.2

The eigenvalues λ for the set of 17 hand images of the type shown in Fig. 12.2A are listed in decreasing order of size below, together with the cumulative sums, csum*. Also listed are the square roots of the eigenvalues, which represent standard deviations. See text for the significance of the cumulative sums. The final column gives an approximation to the mean variation for the boundary points, showing where it takes subpixel levels. In general, boundary noise would be expected to lead to s.ds. of 1 pixel or less, and strong modes (eigenvalues) to give s.ds. representing coherences over several pixels.*

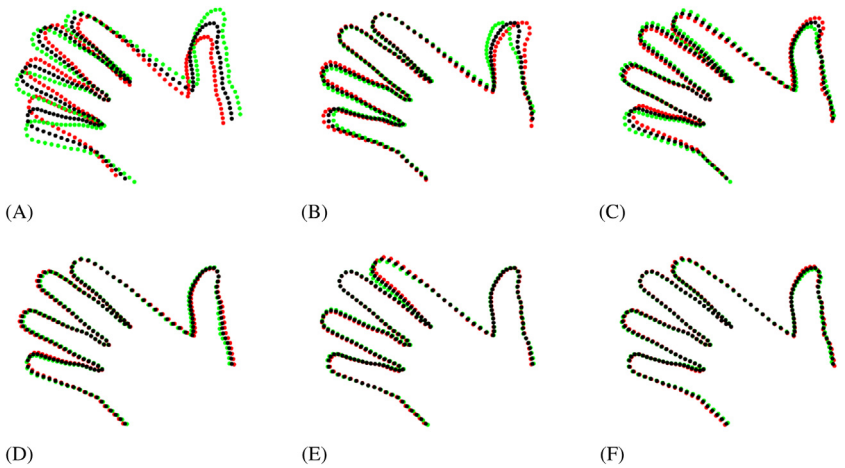

FIGURE 12.3

Effects produced by the various eigenvectors. Here, the effects for the six largest eigenvalues are shown in sequence, after training on 17 hands of the type presented in Fig. 12.2. The black dots show the mean hand position, and the red and green dots show the effects of moving either way from the mean position by the square root of the relevant eigenvalue (which represents the true strength of the eigenvector). Clearly, very little variation is produced by the sixth or higher eigenvalues.

fifth eigenvalue, few such coherences are discernible, and the remaining motions can be attributed largely to boundary noise. In some instances, these almost certainly result from uncertainties due to the shiny light patches mentioned earlier. (Interestingly, the reason for calling the eigenvectors "modes" stems from the physics of molecular vibrations, where the eigenvectors correspond to modes of vibration. This is not totally irrelevant, as we can imagine each hand mode sets the whole hand into a mode of vibration, which is all the more relevant for the video from which the 17 hand images were derived.)

The fact that the PCA eigenvalues are commonly interpreted as "energies" is not especially useful. Indeed, the boundary variations from the smaller eigenvalues appear in Fig. 12.4 as noise and their effects are better described, and measured in images, in terms of standard deviations (s.ds.). This gives us a better idea about which eigenvalues to curtail. To understand this, it is necessary to consider how the variations affect individual boundary points. Specifically, the eigenvalues given in Table 12.1 arise from the motions of $n = 114$ boundary points. We can approximate the mean boundary variations by taking the mean square variations as λ/n and the corresponding s.d. as $\sqrt{\lambda/n}$: the latter are listed in Table 12.1. In our hand problem, the boundary variations for the modes vary from around 20 pixels to subpixel values ~ 0.2 pixels, the latter figure being low because the boundary interpolation algorithm has significantly smoothed the boundary noise.

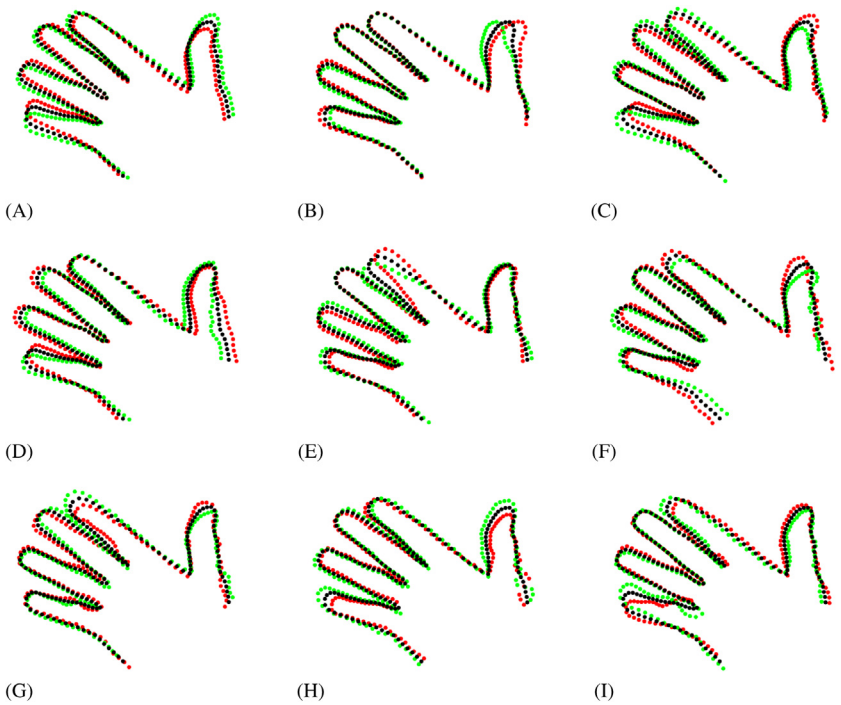

FIGURE 12.4

Clarification of the effects produced by the smaller eigenvalues. Here, the effects of the nine largest eigenvalues are shown in sequence. This figure differs from that of Fig. 12.3, in that, there is a fixed multiplier of 100 on the eigenvectors, which are now no longer weighted by the square root of the relevant eigenvalue. The resulting magnification reveals that rather than governing coherent motions of the fingers, the smaller eigenvalues tend to reflect random boundary noise. The result of the particular boundary uncertainties arising from shiny light patches (e.g., see Fig. 12.2F) are clearly visible to the right of (F)–(I).

In general, the most relevant level at which to curtail the eigenvalues ought to be where the boundary point variation s.d. ($\sqrt{\lambda/n}$) drops to a value close to the expected pixel noise level s.d.

12.5.1 LOCATING OBJECTS USING SHAPE MODELS

Before examining how the trained PCA system can be used to locate test objects in images, it will be useful to summarize the progress we have made so far. First, we have seen that it is necessary to apply similarity transformations to bring the training set examples into alignment—specifically, by normalizing the translation, rotation, and size parameters, so that the sum of squared distances between corresponding landmark points is minimized: consistency at this preliminary stage is

key to making the PCA eigenvectors more accurate. Of course, the test patterns will have to be treated in the same way as the training set patterns and subjected to similar transformations so that their translations, rotations, and sizes are also prenormalized (see below).

Having brought the training set samples into alignment, their shapes can be stacked into s long shape vectors covering the N landmark points (x_{ij}, y_{ij}), $j = 1$ to N:

$$\mathbf{x}_i = (x_{i1}, y_{i1}, x_{i2}, y_{i2}, \ldots, x_{iN} y_{iN})^T, i = 1 \text{ to } s \tag{12.19}$$

We can now define a mean shape for the training set objects

$$\bar{\mathbf{x}} = \frac{1}{s} \sum_{i=1}^{s} \mathbf{x}_i \tag{12.20}$$

(It is also sensible to find the means of the N individual landmark positions, using the similar equation $\bar{\xi} = \frac{1}{N} \sum_{j-1}^{N} \xi_j$, where $\xi_j = \left(x_{1j}, y_{1j}, x_{2j}, y_{2j}, \ldots, x_{sj}, y_{sj},\right)^T$, $j = 1$ to N. Indeed, at first sight Eq. (12.20) may confusingly appear more like a point averaging equation than one for averaging shapes.)

We can also compute the shape covariance

$$\mathbf{S} = \frac{1}{s-1} \sum_{i=1}^{s} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \tag{12.21}$$

To perform PCA fitting, we first approximate the normalized test pattern by the weighted sum of its component eigenvectors

$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{\Phi b} \tag{12.22}$$

Here, $\mathbf{b}$ is the weighting vector containing the t model parameter values corresponding to the t eigenvalues that have been chosen as the largest and most significant ones (see previous section), and $\mathbf{\Phi}$ is the stacked matrix of the corresponding eigenvectors $\mathbf{\Phi} = (\phi_1, \phi_2, \ldots, \phi_t)$. Inverting Eq. (12.22) gives the weighting vector $\mathbf{b}$

$$\mathbf{b} = \mathbf{\Phi}^T (\mathbf{x} - \bar{\mathbf{x}}) \tag{12.23}$$

(Note that $\mathbf{\Phi}$ is symmetric, so its inverse is equal to its transpose.)

The next part of the task is more difficult, because it involves matching test data represented by raw pixel values to training data expressed in terms of landmark points. Furthermore, it would be nonsense to try to generate landmark points manually for more than one or two test images. Instead, we need to find a way of matching gray-scale points directly between the test and training images. In fact, it is possible to start the process by generating a local gray-level appearance profile along the edge normal at each landmark point. (For simplicity, the local normal at each landmark point can be found by taking the perpendicular bisector of the adjacent two landmark points.) This profile can then be matched against the normalized test pattern profile along the same line, and the position of the best fit for a suitable boundary point determined. If this procedure is carried out for each landmark point, we will end up with a new shape model for the test

object. The fitting process can be carried out iteratively at several—typically two or three—scales until convergence takes place.

The local gray-level appearance model was originally taken as the normalized first derivative ($\mathbf{g}_i$) profile (Cootes and Taylor, 1996). In fact, Cootes et al. performed the match by minimizing the Mahalanobis distance M from the mean $\overline{\mathbf{g}}$ of the training set profiles ($\mathbf{g}_i$, $i = 1$ to s) to each unknown test sample $\mathbf{g}_u$, M being given by

$$M^2 = (\mathbf{g}_u - \overline{\mathbf{g}})^T \mathbf{S}_g{}^{-1} (\mathbf{g}_u - \overline{\mathbf{g}}) \tag{12.24}$$

In this equation, $\mathbf{S}_g{}^{-1}$ is the inverse of the gradient covariance matrix $\mathbf{S}_g$. Note that M reduces to the Euclidean distance when $\mathbf{S}_g$ is the identity matrix. The reason for minimizing Mahalanobis distance is to maximize the probability that $\mathbf{g}_u$ is drawn from the same multivariate Gaussian distribution as the training set data. It is clear that the inverse is appropriate because for a 1-D Gaussian distribution $\mathbf{S}_g{}^{-1}$ reduces correctly to the inverse of the variance (the factor $1/2\sigma^2$ has to appear in the Gaussian exponent).

The result of applying this procedure to the PCA model obtained from our set of 17 hand images (see Figs. 12.2–12.4) is shown in Fig. 12.5. The

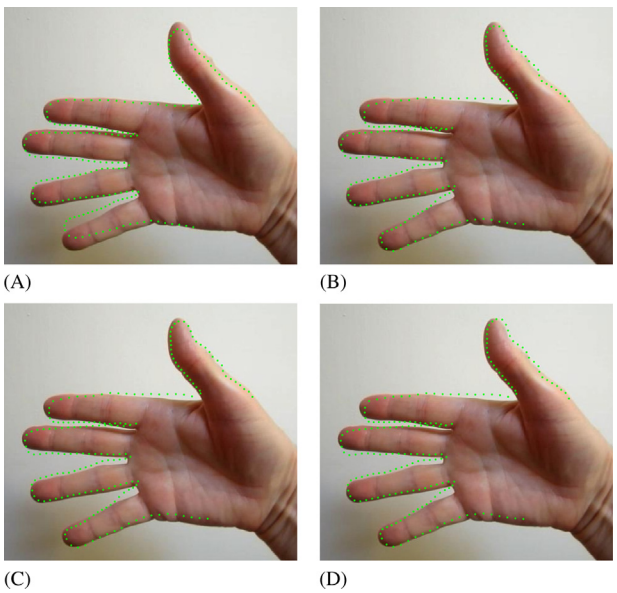

FIGURE 12.5

Result of fitting hand picture to PCA model. (A) Initial approximation. (B)–(D) Convergence of the fit with the Mahalanobis distance approach, after 40, 80, and 120 iterations. Note the restrictions produced using this approach—ultimately because of large intensity variations over the hand region.

successive images (A)–(D) show convergence after about 120 iterations, but the accuracy is seen to be limited. Ultimately, this problem is due to some relatively large intensity variations over the hand region—specifically, shiny regions such as those on the thumb noted earlier, and darker shadow regions around the lower edges of the fingers. Indeed, the individual points on the boundary where Mahalanobis matches are located (see Fig. 12.6) are frequently several pixels away from their ideal positions; and although the matches do improve slightly over the sequence, in this case, they remain biased and noisy.

Many workers have produced improved methods for carrying out gray-scale profile matching in cases where the images are nonideal and cannot be well matched by a linear Mahalanobis distance model. For example, van Ginneken et al. (2002) point to cases where the background of the object may be one of several possible types, and the best match can be identified using a *k*-NN classifier. Similarly, Kroon (2011) quotes situations where large gray-level variations including instances of shadows and/or specular highlights appear near the object boundaries. Kroon develops a rigorous approach to this problem, based on

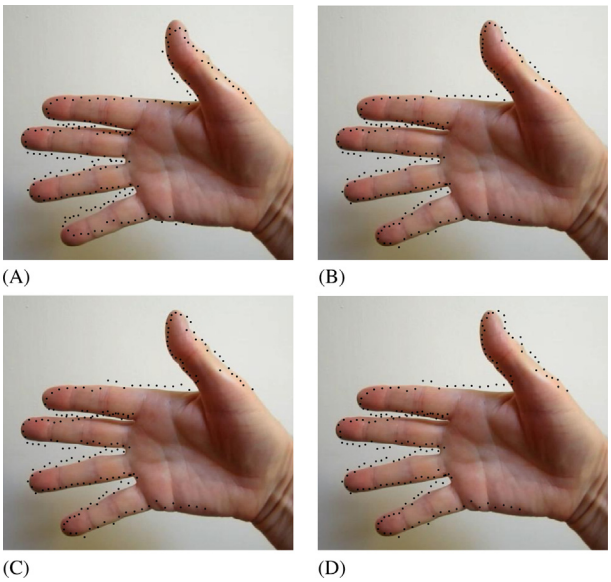

FIGURE 12.6

Local predictions produced by the Mahalanobis distance approach. (A)–(D) show the respective predictions after 1, 41, 81, and 121 iterations. Note that although these improve over the sequence, they remain biassed and noisy.

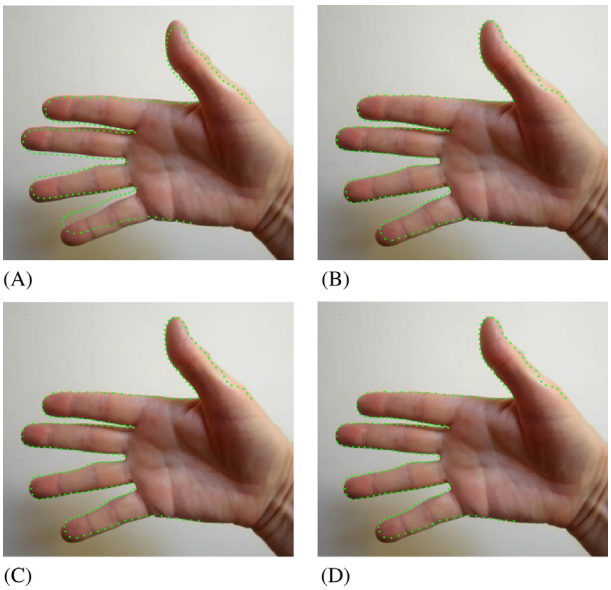

FIGURE 12.7

Result of fitting hand picture to PCA model using improved intensity profiles. (A) Initial approximation. (B)–(D) Convergence of the fit with the PCA intensity profile approach, after 8, 16, and 24 iterations. Here, the restrictions caused by intensity variations over the hand are largely eliminated, resulting in an almost perfect fit after significantly fewer iterations.

applying PCA to the local gray-level appearance profiles to identify the most common modes of variation: this application of PCA (viz., to gray-scale *intensity* profiles) is distinct from that which has already been applied to the boundary *shape* profiles. Following training and finding the eigenvectors, model parameters are determined for the unknown test profile: each parameter is divided by the corresponding s.d. obtained during training, and the matching distance is then calculated as the quadratic sum of the normalized model values: this matching distance is minimized to optimize the fit. Figs. 12.7 and 12.8 show that this approach is highly successful for this particular hand location problem. In particular, the resulting fit is accurate and appears after far fewer iterations than for the Mahalanobis distance approach; clearly, the ultimate reason for this is that the match locations feeding the overall fit are themselves much more reliable and accurate.

Finally, we give a summary of the overall algorithm that has to be used for training the system and fitting test objects (Table 12.2). This has several

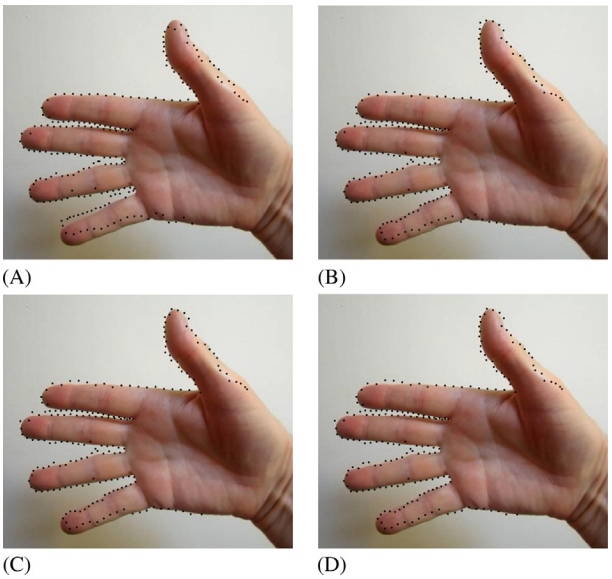

FIGURE 12.8

Local predictions produced by the PCA intensity profile approach. (A)–(D) show the respective predictions after 1, 9, 17, and 25 iterations. Note that these improve over the sequence and are significantly more controlled than for the Mahalanobis distance approach.

intricacies due, for example, to the repeated changes in the frames of reference at different stages of the calculation. There are also several further aspects of the algorithm that it would have been confusing to include in the table, but which should nevertheless be highlighted are as follows:

1. Transformation **T** involves normalizing the pose parameters (translation, rotation, and size).
2. During matching, a total of $2k + 1$ sample points centered around each landmark point are used for matching, and a range of movement of $\pm l$ is allowed, where $l \approx k$.
3. Typical values of parameters are as follows: $k = 6$, $l = 6$, $m = 3$.
4. Search may be speeded up by implementing it at several scales, from coarse to fine, e.g., in steps of 4, 2, and finally 1 pixel.
5. The algorithm is written in a general form that allows either Mahalanobis or PCA profile matching distance to be used.

Table 12.2 Summary of the PCA Object Modeling and Fitting Algorithm

	Reference Frame
Training	
Generate landmark points on training shapes	I
Normalize the landmark coordinates using a suitable transformation, and thus align training shapes to a common reference frame N. Take $\mathbf{T_0}$ as the mean of all the training set transformations	I → N
Perform PCA	N
Limit the number of eigenvalues to the *t* largest, most significant ones	N
Obtain (trained) intensity profile for each landmark point	I
Testing	
Initialize the model shape parameters to those of the mean (trained) shape (i.e., set $\mathbf{b} = 0$)	N
Place the mean object shape (including landmark points) close to the test object, either manually or using the inverted transformation $\mathbf{T}_0^{-1}$	N → I
do {	
Create appearance map of test object by sampling gray profiles along the contour normals, including *k* sample points on each side of each landmark point	I
Search along each contour normal for the section of length $2k + 1$ having the closest appearance profile to that on the model normal	I
Take each section center as the new landmark position	I
Normalize the landmark coordinates using an updated transformation $\mathbf{T}$, employing the same normalization technique as for training	I → N
Calculate the new model parameters using equation $\mathbf{b} = \mathbf{\Phi}^T(\mathbf{x} - \bar{\mathbf{x}})$	N
Limit the model parameters to the range $\pm\, m\sqrt{\lambda}$	N
Convert the model parameters back to normalized contour positions using equation $\mathbf{x} = \bar{\mathbf{x}} + \mathbf{\Phi b}$	N
Convert the contour positions back to real image locations using the inverted normalization transformation $\mathbf{T}^{-1}$	N → I
} until sufficiently converged *or* the maximum number of iterations is reached	

The column on the right indicates where the action is—either in the image frame I, or in the normalized frame N, or moving between them.

12.6 CONCLUDING REMARKS

This chapter has introduced the idea of shape models for object segmentation. Over many years, it has been found that raw thresholding and edge detection do not provide straightforward means of segmenting images, as they are liable to stray far-off course: in particular, this is because of the effects of

random image artifacts such as noise and shadows, not to mention the clutter of nontarget objects in the background. The idea of using shape models is to control the situation so that the segmentations obtained become more meaningful. Active contour models (snakes) and level-set models provide means for tackling this problem: PCA-based methods provide another. The snake and level-set approaches are subject to difficulties because all the necessary rules have to be incorporated into the segmentation algorithm analytically without acknowledging exactly what type of shape it will have: PCA-based methods are more powerful as they train the system to know exactly what type of shape is being sought. Clearly, they involve far more sophistication because of the often huge efforts needed to train the system on real data. Thus, they involve vastly more complication and computation, though on the positive side, they can extract the required (learnt) objects from regions of considerable noise.

Over many years, thresholding-based methods and region growing formed the basis of object segmentation. Subsequently, active contour models (snakes) and level-set methods were aimed at solving the same problem but using more subtle iterative, analytically based methods. However, it was only when PCA-based methods started to be applied to search for objects as instances of models that had been trained on specific target objects that reliability was able to increase dramatically—with the result that objects could be recovered from regions of considerable noise.

12.7 BIBLIOGRAPHICAL AND HISTORICAL NOTES

The basic theory of active contour models and snakes was derived many years ago by Kass, Witkin, and Terzopoulos (Kass and Witkin, 1987; Kass et al., 1988). Subsequently, many workers aimed to produce methods that had the same purpose but which could also overcome the manifest disadvantages of normal snakes. Amongst these were those who promoted the level-set approach—Cremers et al. (2007), Caselles et al. (1997), and Paragios and Deriche (2000). Cootes and Taylor (1996) were almost solely responsible for the PCA-based ASM methods and later developed the more general active appearance model method (Cootes et al., 2001).

Cosío et al. (2010) used simplex search in ASMs for improved boundary segmentation: this involves fast numerical optimization to find the most suitable values of nonlinear functions without the need to calculate function derivatives. Their approach typically employs four pose parameters and ten shape parameters for defining a shape such as the prostate. The method significantly increases the range of object poses and thus results in more accurate boundary segmentation. Chiverton et al. (2008) describe a method that is closely related to the active contour concept: it zones in on objects using parameters relating to

foreground similarity and background dissimilarity and employs a new variational logistic maximum a posteriori contextual modeling schema. In this case, the (achieved) aim is to permit tracking of moving objects by iterative adaptive matching. Mishra et al. (2011) identify five basic limitations of preexisting active contour methods. Their solution is to decouple the internal and external active contour energies and to perform updating for each of them separately. The method is shown to be faster and to have at least comparable segmentation accuracy to five earlier methods.