Chapter 5

Segmentation

R. Szeliski, *Computer Vision: Algorithms and Applications*, Texts in Computer Science,
DOI 10.1007/978-1-84882-935-0_5, © Springer-Verlag London Limited 2011

Figure 5.1 Some popular image segmentation techniques: (a) active contours (Isard and Blake 1998) © 1998 Springer; (b) level sets (Cremers, Rousson, and Deriche 2007) © 2007 Springer; (c) graph-based merging (Felzenszwalb and Huttenlocher 2004b) © 2004 Springer; (d) mean shift (Comaniciu and Meer 2002) © 2002 IEEE; (e) texture and intervening contour-based normalized cuts (Malik, Belongie, Leung *et al.* 2001) © 2001 Springer; (f) binary MRF solved using graph cuts (Boykov and Funka-Lea 2006) © 2006 Springer.

Image segmentation is the task of finding groups of pixels that "go together". In statistics, this problem is known as *cluster analysis* and is a widely studied area with hundreds of different algorithms (Jain and Dubes 1988; Kaufman and Rousseeuw 1990; Jain, Duin, and Mao 2000; Jain, Topchy, Law *et al.* 2004).

In computer vision, image segmentation is one of the oldest and most widely studied problems (Brice and Fennema 1970; Pavlidis 1977; Riseman and Arbib 1977; Ohlander, Price, and Reddy 1978; Rosenfeld and Davis 1979; Haralick and Shapiro 1985). Early techniques tend to use region splitting or merging (Brice and Fennema 1970; Horowitz and Pavlidis 1976; Ohlander, Price, and Reddy 1978; Pavlidis and Liow 1990), which correspond to *divisive* and *agglomerative* algorithms in the clustering literature (Jain, Topchy, Law *et al.* 2004). More recent algorithms often optimize some global criterion, such as intra-region consistency and inter-region boundary lengths or dissimilarity (Leclerc 1989; Mumford and Shah 1989; Shi and Malik 2000; Comaniciu and Meer 2002; Felzenszwalb and Huttenlocher 2004b; Cremers, Rousson, and Deriche 2007).

We have already seen examples of image segmentation in Sections 3.3.2 and 3.7.2. In this chapter, we review some additional techniques that have been developed for image segmentation. These include algorithms based on active contours (Section 5.1) and level sets (Section 5.1.4), region splitting and merging (Section 5.2), *mean shift* (mode finding) (Section 5.3), *normalized cuts* (splitting based on pixel similarity metrics) (Section 5.4), and binary Markov random fields solved using graph cuts (Section 5.5). Figure 5.1 shows some examples of these techniques applied to different images.

Since the literature on image segmentation is so vast, a good way to get a handle on some of the better performing algorithms is to look at experimental comparisons on human-labeled databases (Arbeláez, Maire, Fowlkes *et al.* 2010). The best known of these is the Berkeley Segmentation Dataset and Benchmark[1] (Martin, Fowlkes, Tal *et al.* 2001), which consists of 1000 images from a Corel image dataset that were hand-labeled by 30 human subjects. Many of the more recent image segmentation algorithms report comparative results on this database. For example, Unnikrishnan, Pantofaru, and Hebert (2007) propose new metrics for comparing such algorithms. Estrada and Jepson (2009) compare four well-known segmentation algorithms on the Berkeley data set and conclude that while their own SE-MinCut algorithm (Estrada, Jepson, and Chennubhotla 2004) algorithm outperforms the others by a small margin, there still exists a wide gap between automated and human segmentation performance.[2] A new database of foreground and background segmentations, used by Alpert, Galun, Basri *et al.* (2007), is also available.[3]

5.1 Active contours

While lines, vanishing points, and rectangles are commonplace in the man-made world, curves corresponding to object boundaries are even more common, especially in the natural environment. In this section, we describe three related approaches to locating such boundary curves in images.

[1] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

[2] An interesting observation about their ROC plots is that automated techniques cluster tightly along similar curves, but human performance is all over the map.

[3] http://www.wisdom.weizmann.ac.il/~vision/Seg_Evaluation_DB/index.html

The first, originally called *snakes* by its inventors (Kass, Witkin, and Terzopoulos 1988) (Section 5.1.1), is an energy-minimizing, two-dimensional spline curve that evolves (moves) towards image features such as strong edges. The second, *intelligent scissors* (Mortensen and Barrett 1995) (Section 5.1.3), allow the user to sketch in real time a curve that clings to object boundaries. Finally, *level set* techniques (Section 5.1.4) evolve the curve as the zero-set of a *characteristic function*, which allows them to easily change topology and incorporate region-based statistics.

All three of these are examples of *active contours* (Blake and Isard 1998; Mortensen 1999), since these boundary detectors iteratively move towards their final solution under the combination of image and optional user-guidance forces.

5.1.1 Snakes

Snakes are a two-dimensional generalization of the 1D energy-minimizing splines first introduced in Section 3.7.1,

$$\mathcal{E}_{\text{int}} = \int \alpha(s)\|\boldsymbol{f}_s(s)\|^2 + \beta(s)\|\boldsymbol{f}_{ss}(s)\|^2\, ds, \tag{5.1}$$

where s is the arc-length along the curve $\boldsymbol{f}(s) = (x(s), y(s))$ and $\alpha(s)$ and $\beta(s)$ are first- and second-order continuity weighting functions analogous to the $s(x, y)$ and $c(x, y)$ terms introduced in (3.100–3.101). We can discretize this energy by sampling the initial curve position evenly along its length (Figure 4.35) to obtain

$$\begin{aligned} E_{\text{int}} &= \sum_i \alpha(i)\|f(i+1) - f(i)\|^2/h^2 \\ &\quad + \beta(i)\|f(i+1) - 2f(i) + f(i-1)\|^2/h^4, \end{aligned} \tag{5.2}$$

where h is the step size, which can be neglected if we resample the curve along its arc-length after each iteration.

In addition to this *internal* spline energy, a snake simultaneously minimizes external image-based and constraint-based potentials. The image-based potentials are the sum of several terms

$$\mathcal{E}_{\text{image}} = w_{\text{line}}\mathcal{E}_{\text{line}} + w_{\text{edge}}\mathcal{E}_{\text{edge}} + w_{\text{term}}\mathcal{E}_{\text{term}}, \tag{5.3}$$

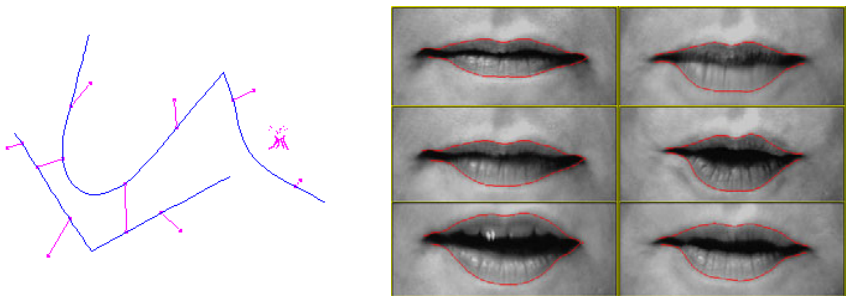
where the *line* term attracts the snake to dark ridges, the *edge* term attracts it to strong gradients (edges), and the *term* term attracts it to line terminations. In practice, most systems only use the edge term, which can either be directly proportional to the image gradients,

$$E_{\text{edge}} = \sum_i -\|\nabla I(\boldsymbol{f}(i))\|^2, \tag{5.4}$$

or to a smoothed version of the image Laplacian,

$$E_{\text{edge}} = \sum_i -|(G_\sigma * \nabla^2 I)(\boldsymbol{f}(i))|^2. \tag{5.5}$$

People also sometimes extract edges and then use a distance map to the edges as an alternative to these two originally proposed potentials.



(a) (b)

Figure 5.2 Snakes (Kass, Witkin, and Terzopoulos 1988) © 1988 Springer: (a) the "snake pit" for interactively controlling shape; (b) lip tracking.

In interactive applications, a variety of user-placed constraints can also be added, e.g., attractive (spring) forces towards anchor points $\boldsymbol{d}(i)$,

$$E_{\text{spring}} = k_i \|\boldsymbol{f}(i) - \boldsymbol{d}(i)\|^2, \tag{5.6}$$

as well as repulsive $1/r$ ("volcano") forces (Figure 5.2a). As the snakes evolve by minimizing their energy, they often "wiggle" and "slither", which accounts for their popular name. Figure 5.2b shows snakes being used to track a person's lips.

Because regular snakes have a tendency to shrink (Exercise 5.1), it is usually better to initialize them by drawing the snake outside the object of interest to be tracked. Alternatively, an expansion *ballooning* force can be added to the dynamics (Cohen and Cohen 1993), essentially moving each point outwards along its normal.

To efficiently solve the sparse linear system arising from snake energy minimization, a sparse direct solver (Appendix A.4) can be used, since the linear system is essentially pentadiagonal.[4] Snake evolution is usually implemented as an alternation between this linear system solution and the linearization of non-linear constraints such as edge energy. A more direct way to find a global energy minimum is to use dynamic programming (Amini, Weymouth, and Jain 1990; Williams and Shah 1992), but this is not often used in practice, since it has been superseded by even more efficient or interactive algorithms such as intelligent scissors (Section 5.1.3) and GrabCut (Section 5.5).

Elastic nets and slippery springs

An interesting variant on snakes, first proposed by Durbin and Willshaw (1987) and later re-formulated in an energy-minimizing framework by Durbin, Szeliski, and Yuille (1989), is the *elastic net* formulation of the Traveling Salesman Problem (TSP). Recall that in a TSP, the salesman must visit each city once while minimizing the total distance traversed. A snake that is constrained to pass through each city could solve this problem (without any optimality guarantees) but it is impossible to tell ahead of time which snake control point should be associated with each city.

[4] A closed snake has a Toeplitz matrix form, which can still be factored and solved in $O(N)$ time.

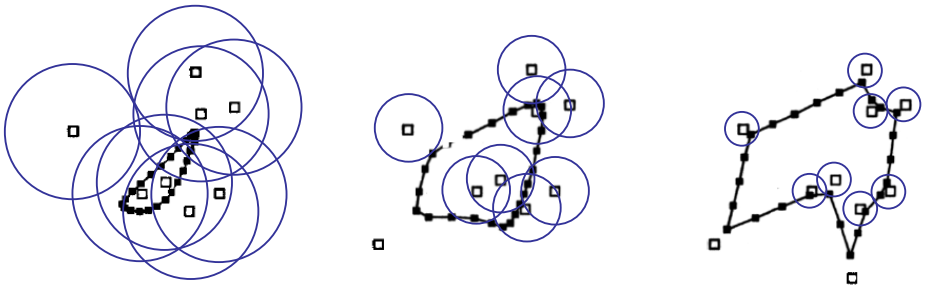


Figure 5.3 Elastic net: The open squares indicate the cities and the closed squares linked by straight line segments are the tour points. The blue circles indicate the approximate extent of the attraction force of each city, which is reduced over time. Under the Bayesian interpretation of the elastic net, the blue circles correspond to one standard deviation of the circular Gaussian that generates each city from some unknown tour point.

Instead of having a fixed constraint between snake nodes and cities, as in (5.6), a city is assumed to pass near *some* point along the tour (Figure 5.3). In a probabilistic interpretation, each city is generated as a *mixture* of Gaussians centered at each tour point,

$$p(\boldsymbol{d}(j)) = \sum_i p_{ij} \quad \text{with} \quad p_{ij} = e^{-d_{ij}^2/(2\sigma^2)} \tag{5.7}$$

where σ is the standard deviation of the Gaussian and

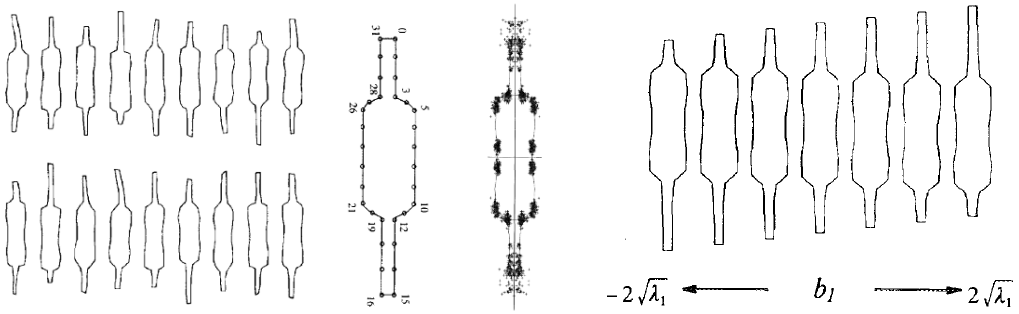
$$d_{ij} = \|\boldsymbol{f}(i) - \boldsymbol{d}(j)\| \tag{5.8}$$

is the Euclidean distance between a tour point $\boldsymbol{f}(i)$ and a city location $\boldsymbol{d}(j)$. The corresponding data fitting energy (negative log likelihood) is

$$E_{\text{slippery}} = -\sum_j \log p(\boldsymbol{d}(j)) = -\sum_j \log \left[\sum e^{-\|\boldsymbol{f}(i)-\boldsymbol{d}(j)\|^2/2\sigma^2}\right]. \tag{5.9}$$

This energy derives its name from the fact that, unlike a regular spring, which couples a given snake point to a given constraint (5.6), this alternative energy defines a *slippery spring* that allows the association between constraints (cities) and curve (tour) points to evolve over time (Szeliski 1989). Note that this is a soft variant of the popular *iterated closest point* data constraint that is often used in fitting or aligning surfaces to data points or to each other (Section 12.2.1) (Besl and McKay 1992; Zhang 1994).

To compute a good solution to the TSP, the slippery spring data association energy is combined with a regular first-order internal smoothness energy (5.3) to define the cost of a tour. The tour $\boldsymbol{f}(s)$ is initialized as a small circle around the mean of the city points and σ is progressively lowered (Figure 5.3). For large σ values, the tour tries to stay near the centroid of the points but as σ decreases each city pulls more and more strongly on its closest tour points (Durbin, Szeliski, and Yuille 1989). In the limit as $\sigma \to 0$, each city is guaranteed to capture at least one tour point and the tours between subsequent cites become straight lines.



(a) (b) (c) (d)

Figure 5.4 Point distribution model for a set of resistors (Cootes, Cooper, Taylor *et al.* 1995) © 1995 Elsevier: (a) set of input resistor shapes; (b) assignment of control points to the boundary; (c) distribution (scatter plot) of point locations; (d) first (largest) mode of variation in the ensemble shapes.

Splines and shape priors

While snakes can be very good at capturing the fine and irregular detail in many real-world contours, they sometimes exhibit too many degrees of freedom, making it more likely that they can get trapped in local minima during their evolution.

One solution to this problem is to control the snake with fewer degrees of freedom through the use of B-spline approximations (Menet, Saint-Marc, and Medioni 1990b,a; Cipolla and Blake 1990). The resulting *B-snake* can be written as

$$\boldsymbol{f}(s) = \sum_k B_k(s)\boldsymbol{x}_k \tag{5.10}$$

or in discrete form as

$$\boldsymbol{F} = \boldsymbol{B}\boldsymbol{X} \tag{5.11}$$

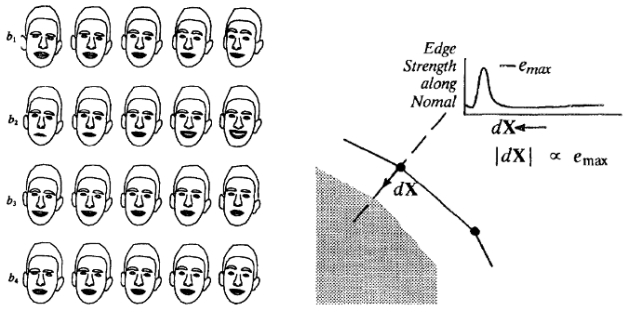
with

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{f}^T(0) \\ \vdots \\ \boldsymbol{f}^T(N) \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} B_0(s_0) & \dots & B_K(s_0) \\ \vdots & \ddots & \vdots \\ B_0(s_N) & \dots & B_K(s_N) \end{bmatrix}, \quad \text{and} \quad \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^T(0) \\ \vdots \\ \boldsymbol{x}^T(K) \end{bmatrix}. \tag{5.12}$$

If the object being tracked or recognized has large variations in location, scale, or orientation, these can be modeled as an additional transformation on the control points, e.g., $\boldsymbol{x}'_k = s\boldsymbol{R}\boldsymbol{x}_k + \boldsymbol{t}$ (2.18), which can be estimated at the same time as the values of the control points. Alternatively, separate *detection* and *alignment* stages can be run to first localize and orient the objects of interest (Cootes, Cooper, Taylor *et al.* 1995).

In a B-snake, because the snake is controlled by fewer degrees of freedom, there is less need for the internal smoothness forces used with the original snakes, although these can still be derived and implemented using finite element analysis, i.e., taking derivatives and integrals of the B-spline basis functions (Terzopoulos 1983; Bathe 2007).

In practice, it is more common to estimate a set of *shape priors* on the typical distribution of the control points $\{\boldsymbol{x}_k\}$ (Cootes, Cooper, Taylor *et al.* 1995). Consider the set of resistor



(a) (b)

Figure 5.5 Active Shape Model (ASM): (a) the effect of varying the first four shape parameters for a set of faces (Cootes, Taylor, Lanitis *et al.* 1993) © 1993 IEEE; (b) searching for the strongest gradient along the normal to each control point (Cootes, Cooper, Taylor *et al.* 1995) © 1995 Elsevier.

shapes shown in Figure 5.4a. If we describe each contour with the set of control points shown in Figure 5.4b, we can plot the distribution of each point in a scatter plot, as shown in Figure 5.4c.

One potential way of describing this distribution would be by the location $\bar{\boldsymbol{x}}_k$ and 2D covariance $\boldsymbol{C}_k$ of each individual point $\boldsymbol{x}_k$. These could then be turned into a quadratic penalty (prior energy) on the point location,

$$E_{\text{loc}}(\boldsymbol{x}_k) = \frac{1}{2}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)^T \boldsymbol{C}_k^{-1}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k). \tag{5.13}$$

In practice, however, the variation in point locations is usually highly correlated.

A preferable approach is to estimate the joint covariance of all the points simultaneously. First, concatenate all of the point locations $\{\boldsymbol{x}_k\}$ into a single vector $\boldsymbol{x}$, e.g., by interleaving the x and y locations of each point. The distribution of these vectors across all training examples (Figure 5.4a) can be described with a mean $\bar{\boldsymbol{x}}$ and a covariance

$$\boldsymbol{C} = \frac{1}{P}\sum_p (\boldsymbol{x}_p - \bar{\boldsymbol{x}})(\boldsymbol{x}_p - \bar{\boldsymbol{x}})^T, \tag{5.14}$$

where $\boldsymbol{x}_p$ are the P training examples. Using *eigenvalue analysis* (Appendix A.1.2), which is also known as *Principal Component Analysis* (PCA) (Appendix B.1.1), the covariance matrix can be written as,

$$\boldsymbol{C} = \boldsymbol{\Phi}\ \text{diag}(\lambda_0 \ldots \lambda_{K-1})\ \boldsymbol{\Phi}^T. \tag{5.15}$$

In most cases, the likely appearance of the points can be modeled using only a few eigenvectors with the largest eigenvalues. The resulting *point distribution model* (Cootes, Taylor, Lanitis *et al.* 1993; Cootes, Cooper, Taylor *et al.* 1995) can be written as

$$\boldsymbol{x} = \bar{\boldsymbol{x}} + \hat{\boldsymbol{\Phi}}\,\boldsymbol{b}, \tag{5.16}$$

where $\boldsymbol{b}$ is an $M \ll K$ element *shape parameter* vector and $\hat{\boldsymbol{\Phi}}$ are the first m columns of $\boldsymbol{\Phi}$. To constrain the shape parameters to reasonable values, we can use a quadratic penalty of the

form

$$E_{\text{shape}} = \frac{1}{2}\boldsymbol{b}^T \operatorname{diag}(\lambda_0 \ldots \lambda_{M-1})\, \boldsymbol{b} = \sum_m b_m^2 / 2\lambda_m. \tag{5.17}$$

Alternatively, the range of allowable b_m values can be limited to some range, e.g., $|b_m| \leq 3\sqrt{\lambda_m}$ (Cootes, Cooper, Taylor *et al.* 1995). Alternative approaches for deriving a set of shape vectors are reviewed by Isard and Blake (1998).

Varying the individual shape parameters b_m over the range $-2\sqrt{\lambda_m} \leq 2\sqrt{\lambda_m}$ can give a good indication of the expected variation in appearance, as shown in Figure 5.4d. Another example, this time related to face contours, is shown in Figure 5.5a.

In order to align a point distribution model with an image, each control point searches in a direction normal to the contour to find the most likely corresponding image edge point (Figure 5.5b). These individual measurements can be combined with priors on the shape parameters (and, if desired, position, scale, and orientation parameters) to estimate a new set of parameters. The resulting *Active Shape Model* (ASM) can be iteratively minimized to fit images to non-rigidly deforming objects such as medical images or body parts such as hands (Cootes, Cooper, Taylor *et al.* 1995). The ASM can also be combined with a PCA analysis of the underlying gray-level distribution to create an *Active Appearance Model* (AAM) (Cootes, Edwards, and Taylor 2001), which we discuss in more detail in Section 14.2.2.

5.1.2 Dynamic snakes and CONDENSATION

In many applications of active contours, the object of interest is being tracked from frame to frame as it deforms and evolves. In this case, it makes sense to use estimates from the previous frame to predict and constrain the new estimates.

One way to do this is to use Kalman filtering, which results in a formulation called *Kalman snakes* (Terzopoulos and Szeliski 1992; Blake, Curwen, and Zisserman 1993). The Kalman filter is based on a linear dynamic model of shape parameter evolution,

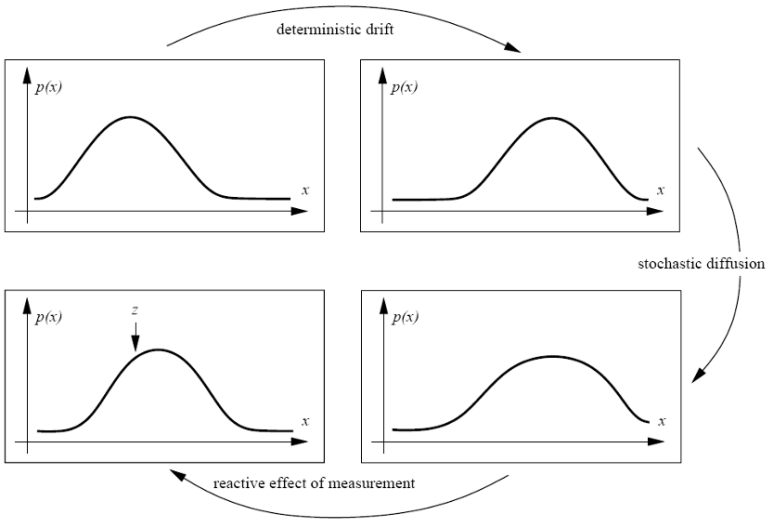
$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \boldsymbol{w}_t, \tag{5.18}$$

where $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t-1}$ are the current and previous state variables, $\boldsymbol{A}$ is the linear *transition matrix*, and $\boldsymbol{w}$ is a noise (perturbation) vector, which is often modeled as a Gaussian (Gelb 1974). The matrices $\boldsymbol{A}$ and the noise covariance can be learned ahead of time by observing typical sequences of the object being tracked (Blake and Isard 1998).

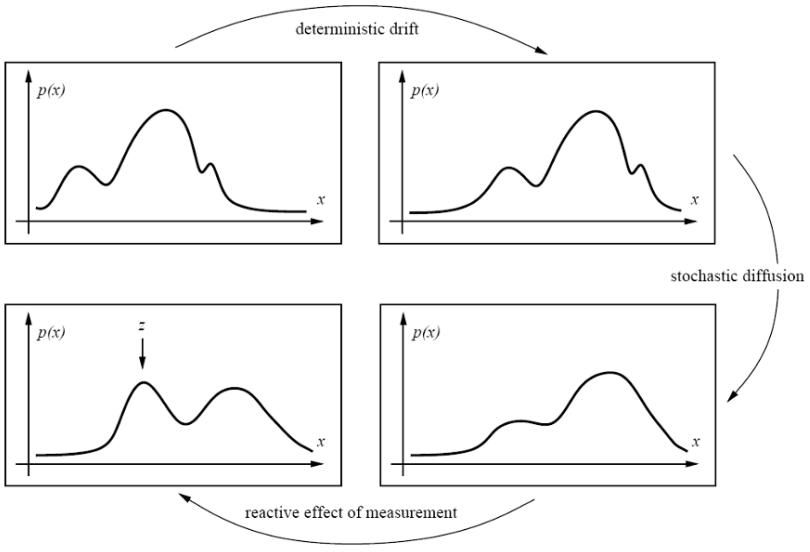
The qualitative behavior of the Kalman filter can be seen in Figure 5.6a. The linear dynamic model causes a deterministic change (drift) in the previous estimate, while the process noise (perturbation) causes a stochastic diffusion that increases the system entropy (lack of certainty). New measurements from the current frame restore some of the certainty (peakedness) in the updated estimate.

In many situations, however, such as when tracking in clutter, a better estimate for the contour can be obtained if we remove the assumptions that the distribution are Gaussian, which is what the Kalman filter requires. In this case, a general multi-modal distribution is propagated, as shown in Figure 5.6b. In order to model such multi-modal distributions, Isard and Blake (1998) introduced the use of *particle filtering* to the computer vision community.[5]

[5] Alternatives to modeling multi-modal distributions include *mixtures of Gaussians* (Bishop 2006) and *multiple hypothesis tracking* (Bar-Shalom and Fortmann 1988; Cham and Rehg 1999).



(a)



(b)

Figure 5.6 Probability density propagation (Isard and Blake 1998) © 1998 Springer. At the beginning of each estimation step, the probability density is updated according to the linear dynamic model (deterministic drift) and its certainty is reduced due to process noise (stochastic diffusion). New measurements introduce additional information that helps refine the current estimate. (a) The Kalman filter models the distributions as uni-modal, i.e., using a mean and covariance. (b) Some applications require more general multi-modal distributions.

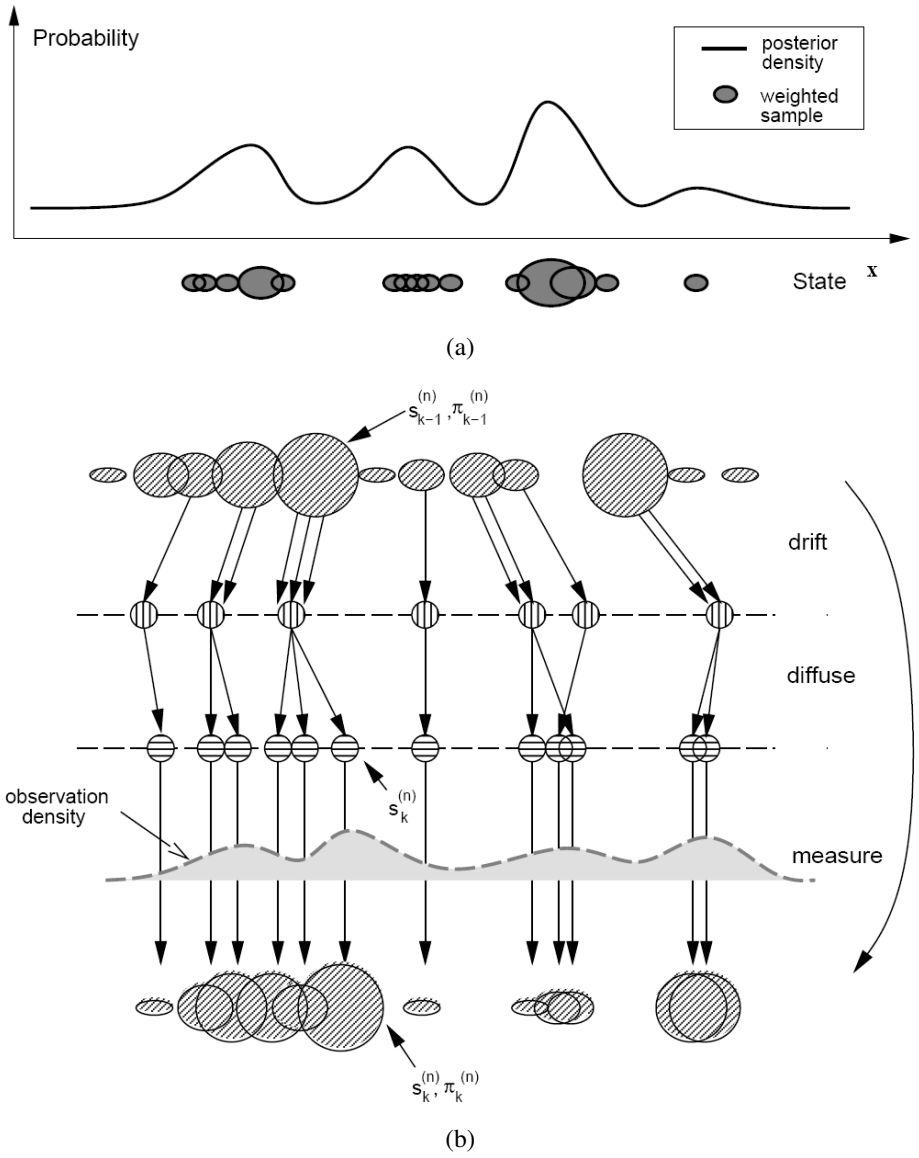
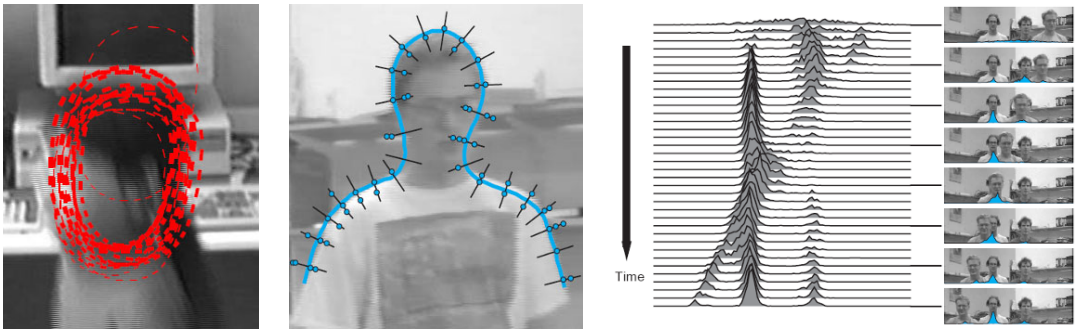


Figure 5.7 Factored sampling using particle filter in the CONDENSATION algorithm (Isard and Blake 1998) © 1998 Springer: (a) each density distribution is represented using a superposition of weighted *particles*; (b) the drift-diffusion-measurement cycle implemented using random sampling, perturbation, and re-weighting stages.



(a) (b) (c)

Figure 5.8 Head tracking using CONDENSATION (Isard and Blake 1998) © 1998 Springer: (a) sample set representation of head estimate distribution; (b) multiple measurements at each control vertex location; (c) multi-hypothesis tracking over time.

Particle filtering techniques represent a probability distribution using a collection of weighted point samples (Figure 5.7a) (Andrieu, de Freitas, Doucet *et al.* 2003; Bishop 2006; Koller and Friedman 2009). To update the locations of the samples according to the linear dynamics (deterministic drift), the centers of the samples are updated according to (5.18) and multiple samples are generated for each point (Figure 5.7b). These are then perturbed to account for the stochastic diffusion, i.e., their locations are moved by random vectors taken from the distribution of $\boldsymbol{w}$.[6] Finally, the weights of these samples are multiplied by the measurement probability density, i.e., we take each sample and measure its likelihood given the current (new) measurements. Because the point samples represent and propagate conditional estimates of the multi-modal density, Isard and Blake (1998) dubbed their algorithm CONditional DENSity propagATION or CONDENSATION.

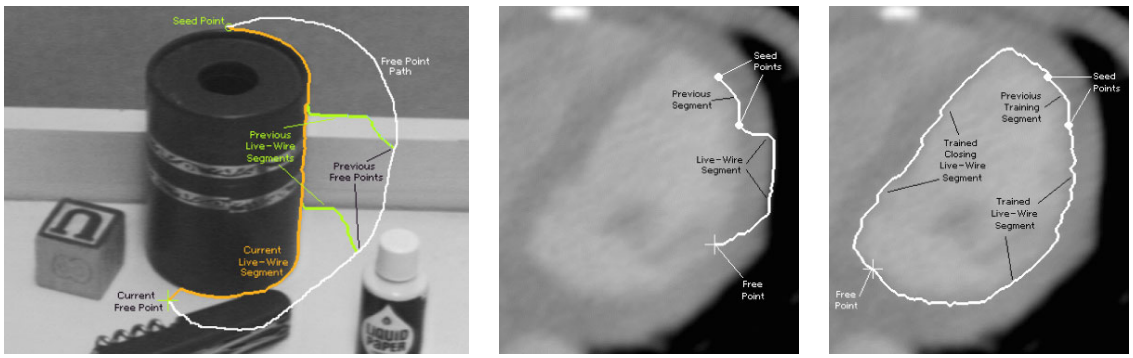
Figure 5.8a shows what a factored sample of a head tracker might look like, drawing a red B-spline contour for each of (a subset of) the particles being tracked. Figure 5.8b shows why the measurement density itself is often multi-modal: the locations of the edges perpendicular to the spline curve can have multiple local maxima due to background clutter. Finally, Figure 5.8c shows the temporal evolution of the conditional density (x coordinate of the head and shoulder tracker centroid) as it tracks several people over time.

5.1.3 Scissors

Active contours allow a user to roughly specify a boundary of interest and have the system evolve the contour towards a more accurate location as well as track it over time. The results of this curve evolution, however, may be unpredictable and may require additional user-based hints to achieve the desired result.

An alternative approach is to have the system optimize the contour in real time as the user is drawing (Mortensen 1999). The *intelligent scissors* system developed by Mortensen and Barrett (1995) does just that. As the user draws a rough outline (the white curve in Figure 5.9a), the system computes and draws a better curve that clings to high-contrast edges

[6] Note that because of the structure of these steps, non-linear dynamics and non-Gaussian noise can be used.



(a) (b) (c)

Figure 5.9 Intelligent scissors: (a) as the mouse traces the white path, the scissors follow the orange path along the object boundary (the green curves show intermediate positions) (Mortensen and Barrett 1995) © 1995 ACM; (b) regular scissors can sometimes jump to a stronger (incorrect) boundary; (c) after training to the previous segment, similar edge profiles are preferred (Mortensen and Barrett 1998) © 1995 Elsevier.

(the orange curve).

To compute the optimal curve path (*live-wire*), the image is first pre-processed to associate low costs with edges (links between neighboring horizontal, vertical, and diagonal, i.e., $\mathcal{N}_8$ neighbors) that are likely to be boundary elements. Their system uses a combination of zero-crossing, gradient magnitudes, and gradient orientations to compute these costs.

Next, as the user traces a rough curve, the system continuously recomputes the lowest-cost path between the starting *seed point* and the current mouse location using Dijkstra's algorithm, a breadth-first dynamic programming algorithm that terminates at the current target location.

In order to keep the system from jumping around unpredictably, the system will "freeze" the curve to date (reset the seed point) after a period of inactivity. To prevent the live wire from jumping onto adjacent higher-contrast contours, the system also "learns" the intensity profile under the current optimized curve, and uses this to preferentially keep the wire moving along the same (or a similar looking) boundary (Figure 5.9b–c).

Several extensions have been proposed to the basic algorithm, which works remarkably well even in its original form. Mortensen and Barrett (1999) use *tobogganing*, which is a simple form of watershed region segmentation, to pre-segment the image into regions whose boundaries become candidates for optimized curve paths. The resulting region boundaries are turned into a much smaller graph, where nodes are located wherever three or four regions meet. The Dijkstra algorithm is then run on this reduced graph, resulting in much faster (and often more stable) performance. Another extension to intelligent scissors is to use a probabilistic framework that takes into account the current trajectory of the boundary, resulting in a system called JetStream (Pérez, Blake, and Gangnet 2001).

Instead of re-computing an optimal curve at each time instant, a simpler system can be developed by simply "snapping" the current mouse position to the nearest likely boundary point (Gleicher 1995). Applications of these boundary extraction techniques to image cutting and pasting are presented in Section 10.4.

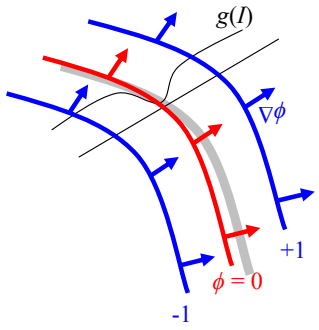


Figure 5.10 Level set evolution for a geodesic active contour. The embedding function ϕ is updated based on the curvature of the underlying surface modulated by the edge/speed function $g(I)$, as well as the gradient of $g(I)$, thereby attracting it to strong edges.

5.1.4 Level Sets

A limitation of active contours based on parametric curves of the form $\boldsymbol{f}(s)$, e.g., snakes, B-snakes, and CONDENSATION, is that it is challenging to change the topology of the curve as it evolves. (McInerney and Terzopoulos (1999, 2000) describe one approach to doing this.) Furthermore, if the shape changes dramatically, curve reparameterization may also be required.

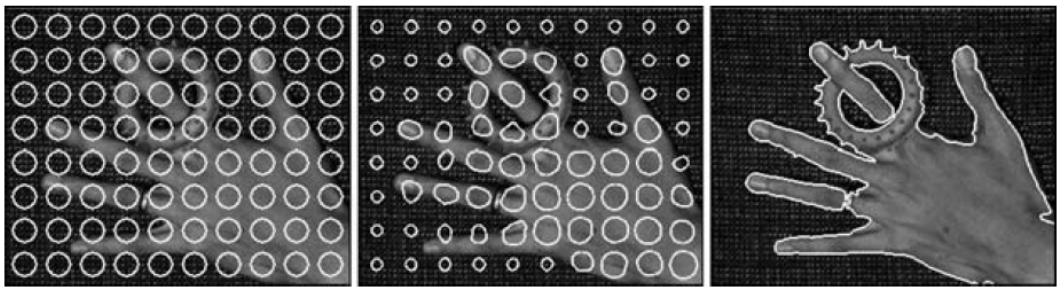
An alternative representation for such closed contours is to use a *level set*, where the *zero-crossing(s)* of a *characteristic* (or signed distance (Section 3.3.3)) function define the curve. Level sets evolve to fit and track objects of interest by modifying the underlying *embedding function* (another name for this 2D function) $\phi(x, y)$ instead of the curve $\boldsymbol{f}(s)$ (Malladi, Sethian, and Vemuri 1995; Sethian 1999; Sapiro 2001; Osher and Paragios 2003). To reduce the amount of computation required, only a small strip (frontier) around the locations of the current zero-crossing needs to updated at each step, which results in what are called *fast marching methods* (Sethian 1999).

An example of an evolution equation is the *geodesic active contour* proposed by Caselles, Kimmel, and Sapiro (1997) and Yezzi, Kichenassamy, Kumar *et al.* (1997),

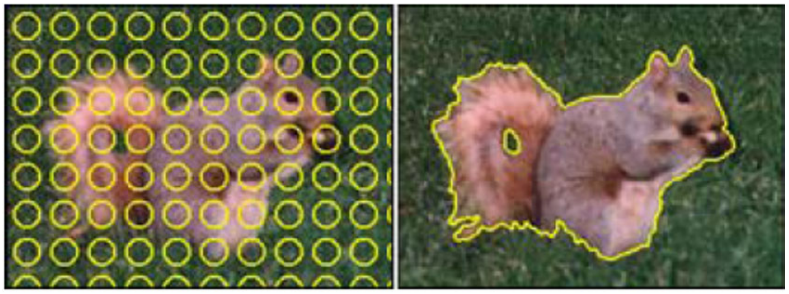
$$\begin{aligned} \frac{d\phi}{dt} &= |\nabla\phi| \text{div}\left(g(I)\frac{\nabla\phi}{|\nabla\phi|}\right) \\ &= g(I)|\nabla\phi| \text{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) + \nabla g(I) \cdot \nabla\phi, \end{aligned} \tag{5.19}$$

where $g(I)$ is a generalized version of the snake edge potential (5.5). To get an intuitive sense of the curve's behavior, assume that the embedding function ϕ is a signed distance function away from the curve (Figure 5.10), in which case $|\phi| = 1$. The first term in Equation (5.19) moves the curve in the direction of its curvature, i.e., it acts to straighten the curve, under the influence of the modulation function $g(I)$. The second term moves the curve down the gradient of $g(I)$, encouraging the curve to migrate towards minima of $g(I)$.

While this level-set formulation can readily change topology, it is still susceptible to local minima, since it is based on local measurements such as image gradients. An alternative



(a)



(b)

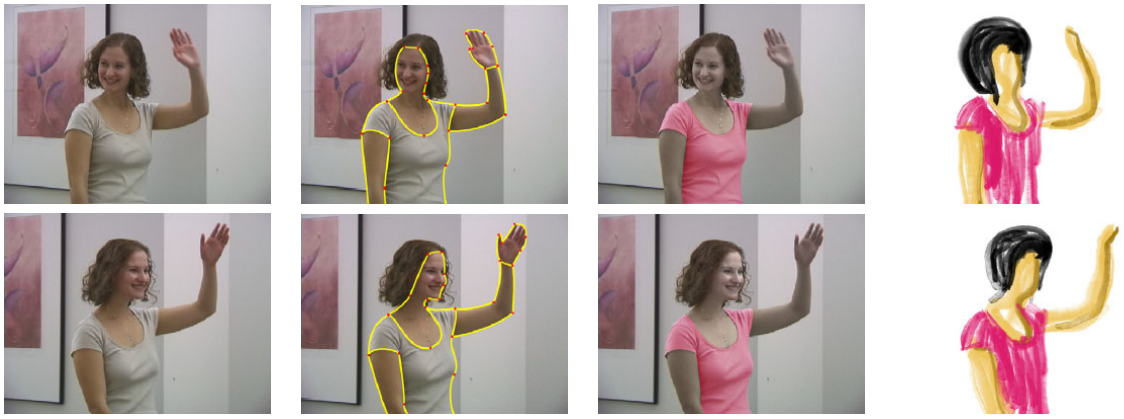
Figure 5.11 Level set segmentation (Cremers, Rousson, and Deriche 2007) © 2007 Springer: (a) grayscale image segmentation and (b) color image segmentation. Uni-variate and multi-variate Gaussians are used to model the foreground and background pixel distributions. The initial circles evolve towards an accurate segmentation of foreground and background, adapting their topology as they evolve.

approach is to re-cast the problem in a segmentation framework, where the energy measures the consistency of the image statistics (e.g., color, texture, motion) inside and outside the segmented regions (Cremers, Rousson, and Deriche 2007; Rousson and Paragios 2008; Houhou, Thiran, and Bresson 2008). These approaches build on earlier energy-based segmentation frameworks introduced by Leclerc (1989), Mumford and Shah (1989), and Chan and Vese (1992), which are discussed in more detail in Section 5.5. Examples of such level-set segmentations are shown in Figure 5.11, which shows the evolution of the level sets from a series of distributed circles towards the final binary segmentation.

For more information on level sets and their applications, please see the collection of papers edited by Osher and Paragios (2003) as well as the series of Workshops on Variational and Level Set Methods in Computer Vision (Paragios, Faugeras, Chan *et al.* 2005) and Special Issues on Scale Space and Variational Methods in Computer Vision (Paragios and Sgallari 2009).

5.1.5 *Application*: Contour tracking and rotoscoping

Active contours can be used in a wide variety of object-tracking applications (Blake and Isard 1998; Yilmaz, Javed, and Shah 2006). For example, they can be used to track facial features for performance-driven animation (Terzopoulos and Waters 1990; Lee, Terzopoulos, and Wa-



(a) (b) (c) (d)

Figure 5.12 Keyframe-based rotoscoping (Agarwala, Hertzmann, Seitz *et al.* 2004) © 2004 ACM: (a) original frames; (b) rotoscoped contours; (c) re-colored blouse; (d) rotoscoped hand-drawn animation.

ters 1995; Parke and Waters 1996; Bregler, Covell, and Slaney 1997) (Figure 5.2b). They can also be used to track heads and people, as shown in Figure 5.8, as well as moving vehicles (Paragios and Deriche 2000). Additional applications include medical image segmentation, where contours can be tracked from slice to slice in computerized tomography (3D medical imagery) (Cootes and Taylor 2001) or over time, as in ultrasound scans.

An interesting application that is closer to computer animation and visual effects is *rotoscoping*, which uses the tracked contours to deform a set of hand-drawn animations (or to modify or replace the original video frames).[7] Agarwala, Hertzmann, Seitz *et al.* (2004) present a system based on tracking hand-drawn B-spline contours drawn at selected keyframes, using a combination of geometric and appearance-based criteria (Figure 5.12). They also provide an excellent review of previous rotoscoping and image-based, contour-tracking systems.

Additional applications of rotoscoping (object contour detection and segmentation), such as cutting and pasting objects from one photograph into another, are presented in Section 10.4.

5.2 Split and merge

As mentioned in the introduction to this chapter, the simplest possible technique for segmenting a grayscale image is to select a threshold and then compute connected components (Section 3.3.2). Unfortunately, a single threshold is rarely sufficient for the whole image because of lighting and intra-object statistical variations.

In this section, we describe a number of algorithms that proceed either by recursively splitting the whole image into pieces based on region statistics or, conversely, merging pixels and regions together in a hierarchical fashion. It is also possible to combine both splitting and merging by starting with a medium-grain segmentation (in a quadtree representation) and

[7] The term comes from a device (a rotoscope) that projected frames of a live-action film underneath an acetate so that artists could draw animations directly over the actors' shapes.