Chapter 5

Segmentation

5.1 Active contours . . . 237
5.1.1 Snakes . . . 238
5.1.2 Dynamic snakes and CONDENSATION . . . 243
5.1.3 Scissors . . . 246
5.1.4 Level Sets . . . 248
5.1.5 *Application*: Contour tracking and rotoscoping . . . 249
5.2 Split and merge . . . 250
5.2.1 Watershed . . . 251
5.2.2 Region splitting (divisive clustering) . . . 251
5.2.3 Region merging (agglomerative clustering) . . . 251
5.2.4 Graph-based segmentation . . . 252
5.2.5 Probabilistic aggregation . . . 253
5.3 Mean shift and mode finding . . . 254
5.3.1 K-means and mixtures of Gaussians . . . 256
5.3.2 Mean shift . . . 257
5.4 Normalized cuts . . . 260
5.5 Graph cuts and energy-based methods . . . 264
5.5.1 *Application*: Medical image segmentation . . . 268
5.6 Additional reading . . . 268
5.7 Exercises . . . 270

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

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

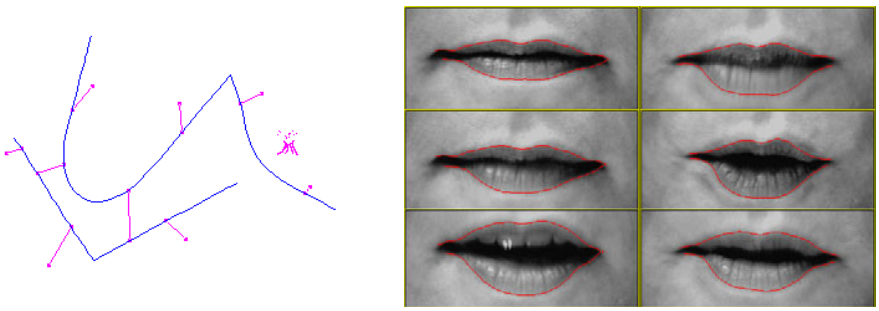
where the *line* term attracts the snake to dark ridges, the *edge* term attracts it to strong gradients (edges), and the *term* term attracts it to line terminations. In practice, most systems only use the edge term, which can either be directly proportional to the image gradients,

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

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.

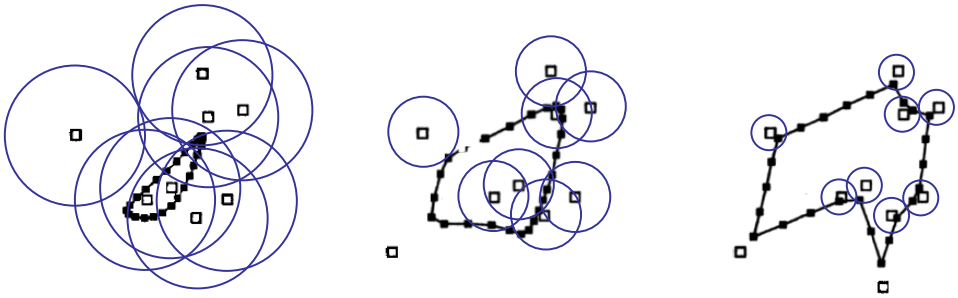


Figure 5.3 Elastic net: The open squares indicate the cities and the closed squares linked by straight line segments are the tour points. The blue circles indicate the approximate extent of the attraction force of each city, which is reduced over time. Under the Bayesian interpretation of the elastic 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,

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

where σ is the standard deviation of the Gaussian and

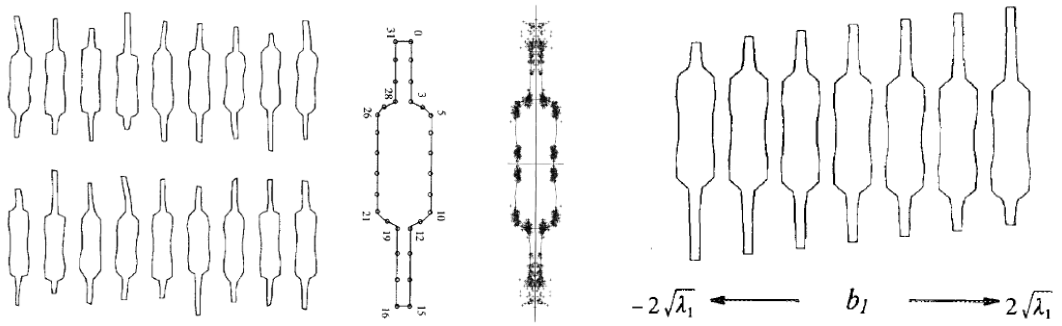
$$d_{ij} = \|\boldsymbol{f}(i) - \boldsymbol{d}(j)\| \tag{5.8}$$

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

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

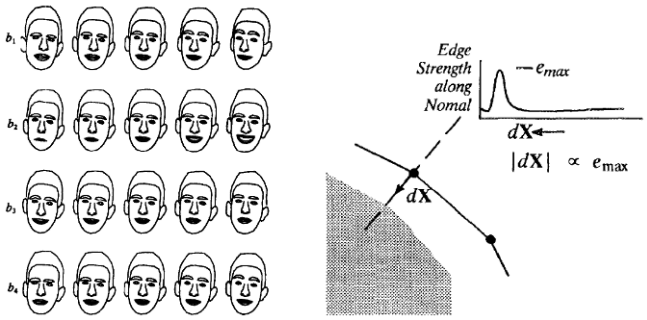
with

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

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

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

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,

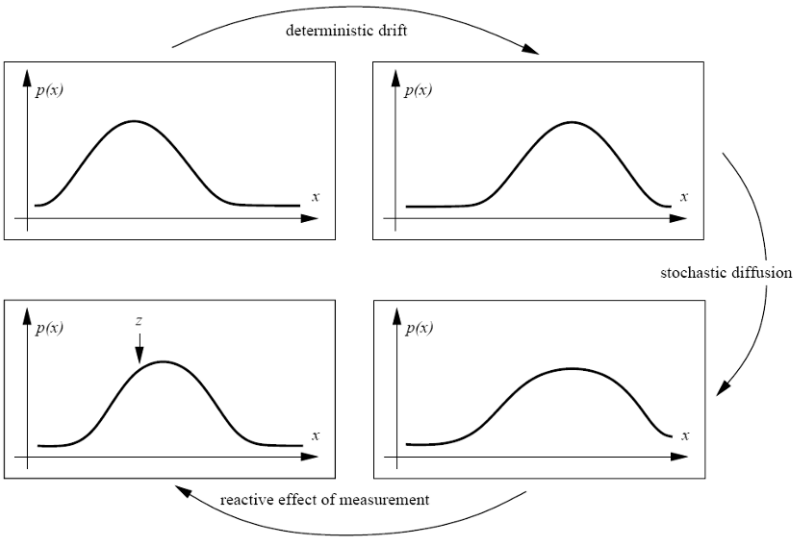
$$\boldsymbol{x}_t = \boldsymbol{A}\boldsymbol{x}_{t-1} + \boldsymbol{w}_t, \tag{5.18}$$

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).

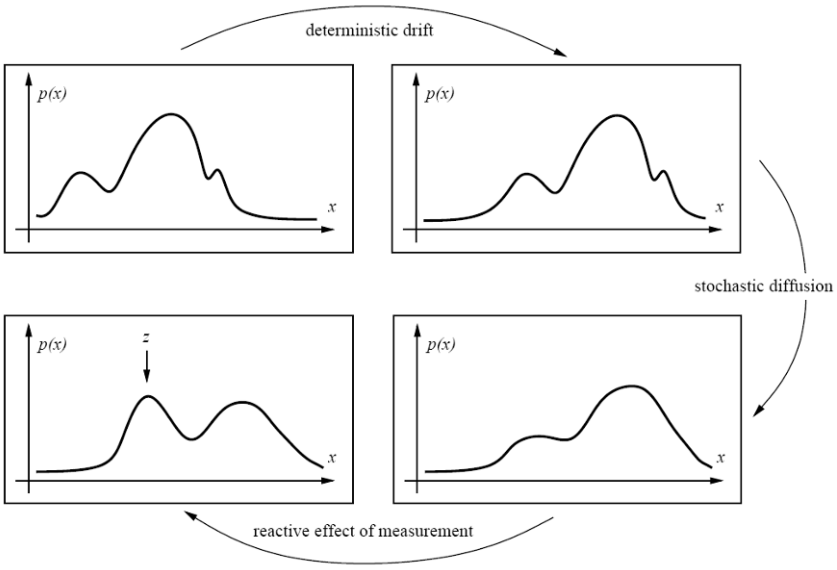
The qualitative behavior of the Kalman filter can be seen in Figure 5.6a. The linear dynamic model causes a deterministic change (drift) in the previous estimate, while the process noise (perturbation) causes a stochastic diffusion that increases the system entropy (lack of certainty). New measurements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

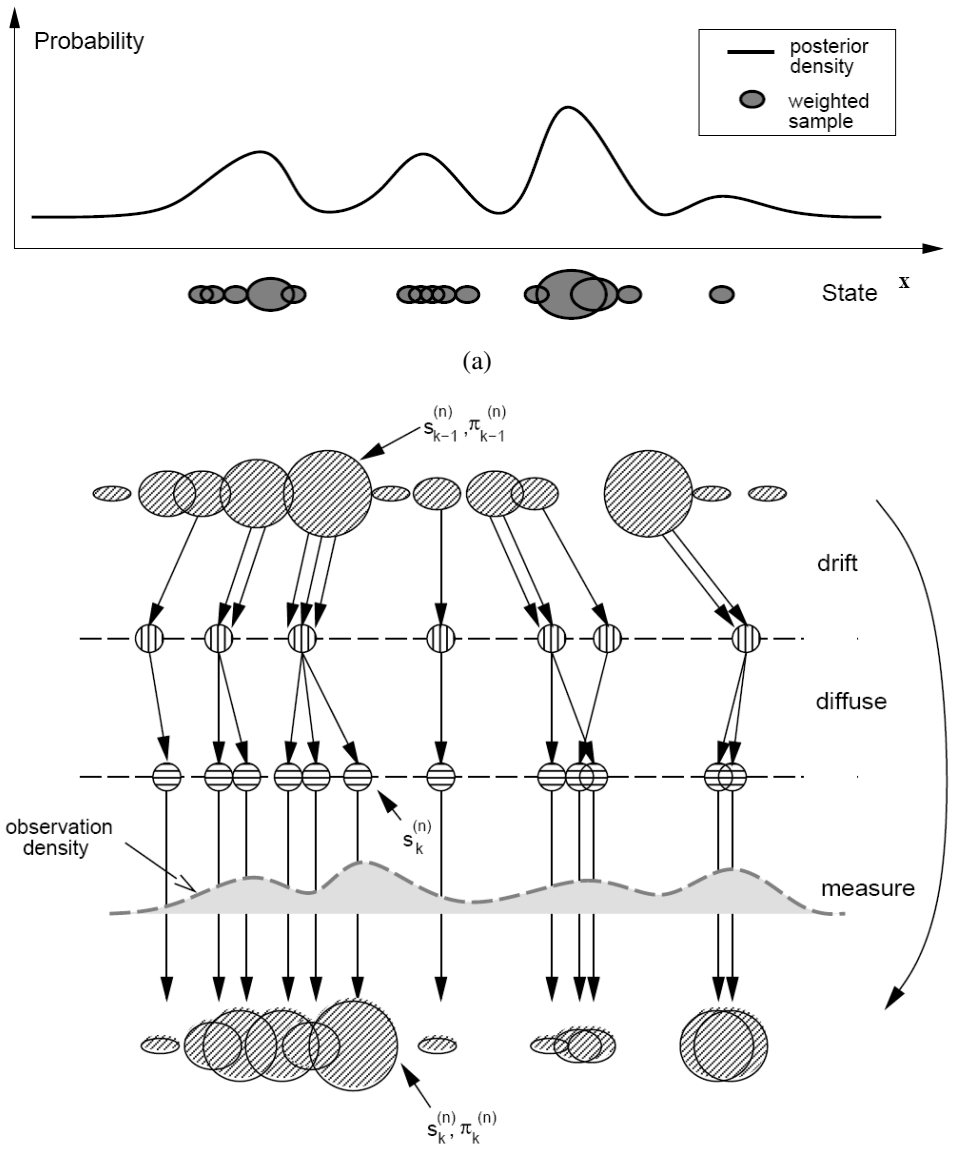
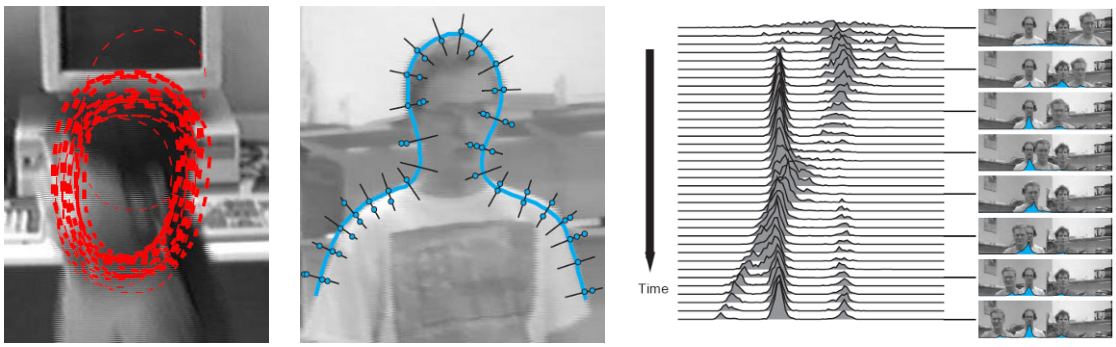


Figure 5.7 Factored sampling using particle filter in the CONDENSATION algorithm (Isard and Blake 1998) © 1998 Springer: (a) each density distribution is represented using a superposition of weighted *particles*; (b) the drift-diffusion-measurement cycle implemented using random sampling, perturbation,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

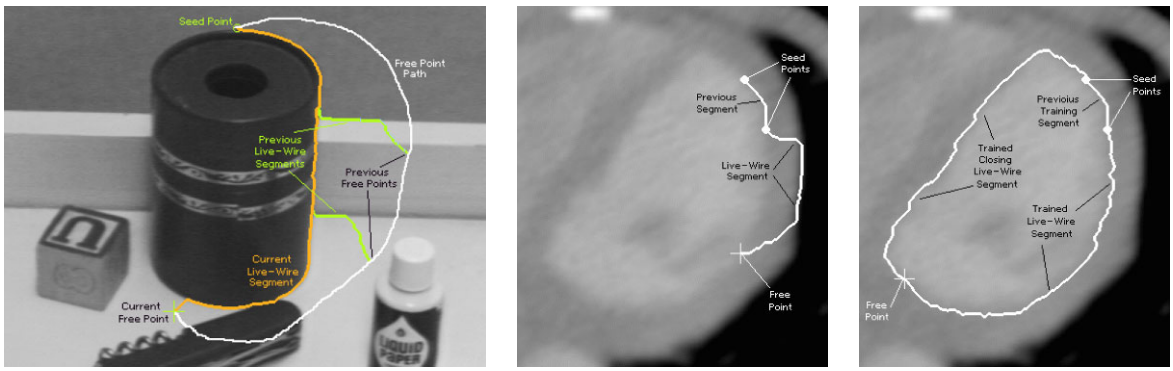
Figure 5.8a shows what a factored sample of a head tracker might look like, drawing a red B-spline contour for each of (a subset of) the particles being tracked. Figure 5.8b shows why the measurement density itself is often multi-modal: the locations of the edges perpendicular to the spline curve can 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.

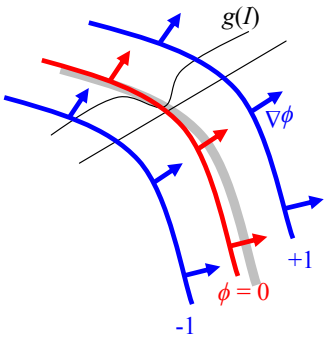


Figure 5.10 Level set evolution for a geodesic active contour. The embedding function ϕ is updated based on the curvature of the underlying surface modulated by the edge/speed function $g(I)$, as well as the gradient of $g(I)$, thereby attracting it to strong edges.

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.

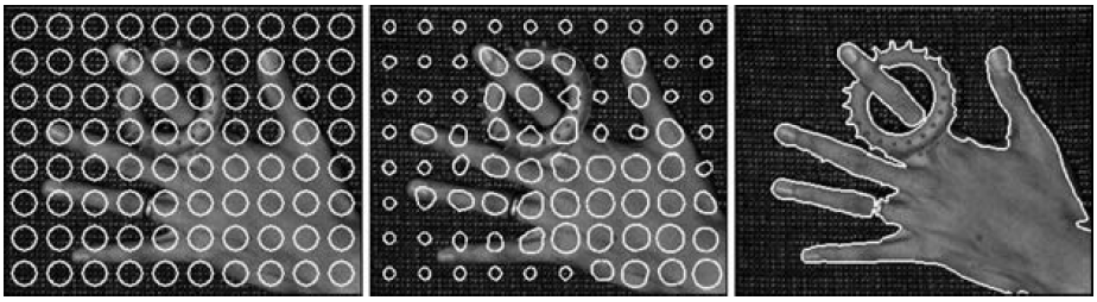
An alternative representation for such closed contours is to use a *level set*, where the *zero-crossing(s)* of a *characteristic* (or signed distance (Section 3.3.3)) function define the curve. Level sets evolve to fit and track objects of interest by modifying the underlying *embedding function* (another 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),

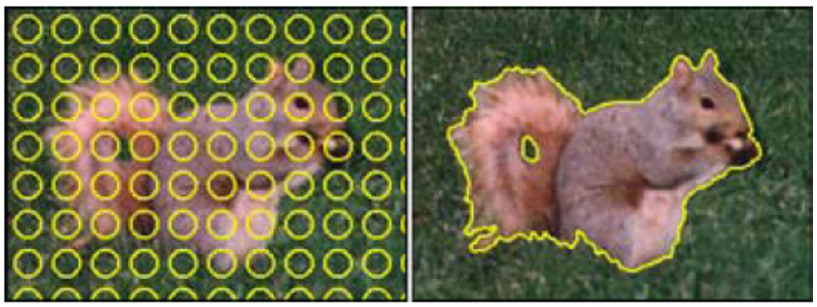
$$\begin{aligned} \frac{d\phi}{dt} &= |\nabla\phi| \text{div}\left(g(I)\frac{\nabla\phi}{|\nabla\phi|}\right) \\ &= g(I)|\nabla\phi| \text{div}\left(\frac{\nabla\phi}{|\nabla\phi|}\right) + \nabla g(I) \cdot \nabla\phi, \end{aligned} \tag{5.19}$$

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)

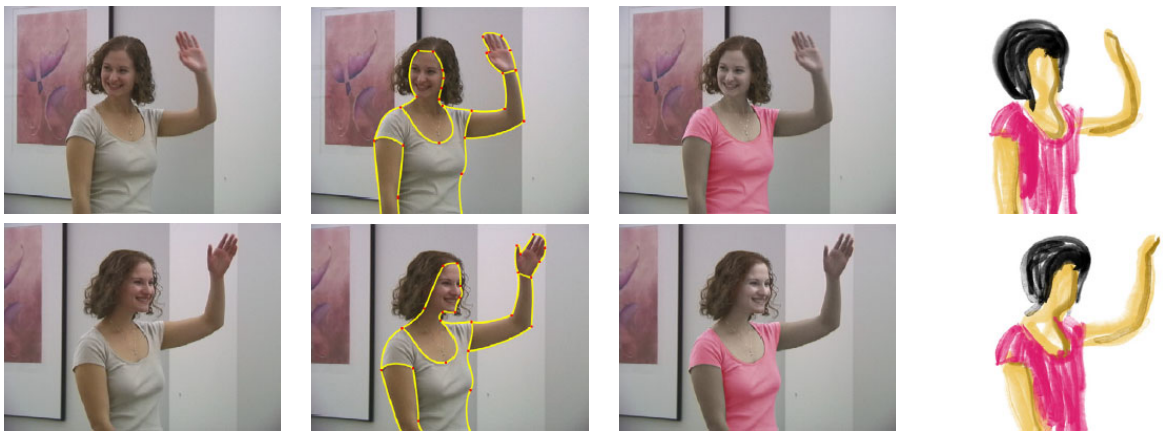
Figure 5.11 Level set segmentation (Cremers, Rousson, and Deriche 2007) © 2007 Springer: (a) grayscale image segmentation and (b) color image segmentation. Uni-variate and multi-variate Gaussians are used to model the foreground and background pixel distributions. The initial circles evolve towards an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

then allowing both merging and splitting operations (Horowitz and Pavlidis 1976; Pavlidis and Liow 1990).

5.2.1 Watershed

A technique related to thresholding, since it operates on a grayscale image, is *watershed* computation (Vincent and Soille 1991). This technique segments an image into several *catchment basins*, which are the regions of an image (interpreted as a height field or landscape) where rain would flow into the same lake. An efficient way to compute such regions is to start flooding the landscape at all of the local minima and to label ridges wherever differently evolving components meet. The whole algorithm can be implemented using a priority queue of pixels and breadth-first search (Vincent and Soille 1991).[8]

Since images rarely have dark regions separated by lighter ridges, watershed segmentation is usually applied to a smoothed version of the gradient magnitude image, which also makes it usable with color images. As an alternative, the maximum oriented energy in a steerable filter (3.28–3.29) (Freeman and Adelson 1991) can be used as the basis of the *oriented watershed transform* developed by Arbeláez, Maire, Fowlkes *et al.* (2010). Such techniques end up finding smooth regions separated by visible (higher gradient) boundaries. Since such boundaries are what active contours usually follow, active contour algorithms (Mortensen and Barrett 1999; Li, Sun, Tang *et al.* 2004) often precompute such a segmentation using either the watershed or the related *tobogganing* technique (Section 5.1.3).

Unfortunately, watershed segmentation associates a unique region with each local minimum, which can lead to over-segmentation. Watershed segmentation is therefore often used as part of an interactive system, where the user first marks seed locations (with a click or a short stroke) that correspond to the centers of different desired components. Figure 5.13 shows the results of running the watershed algorithm with some manually placed markers on a confocal microscopy image. It also shows the result for an improved version of watershed that uses local morphology to smooth out and optimize the boundaries separating the regions (Beare 2006).

5.2.2 Region splitting (divisive clustering)

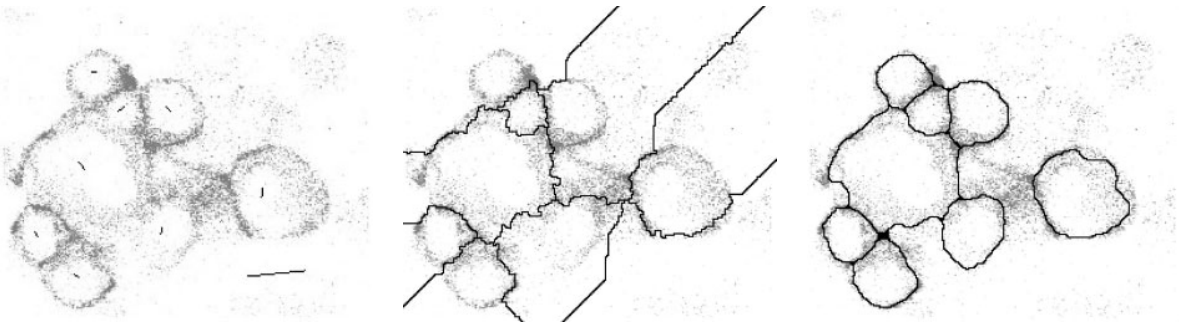
Splitting the image into successively finer regions is one of the oldest techniques in computer vision. Ohlander, Price, and Reddy (1978) present such a technique, which first computes a histogram for the whole image and then finds a threshold that best separates the large peaks in the histogram. This process is repeated until regions are either fairly uniform or below a certain size.

More recent splitting algorithms often optimize some metric of intra-region similarity and inter-region dissimilarity. These are covered in Sections 5.4 and 5.5.

5.2.3 Region merging (agglomerative clustering)

Region merging techniques also date back to the beginnings of computer vision. Brice and Fennema (1970) use a dual grid for representing boundaries between pixels and merge re-

[8] A related algorithm can be used to compute maximally stable extremal regions (MSERs) efficiently (Section 4.1.1) (Nistér and Stewénius 2008).



(a) (b) (c)

Figure 5.13 Locally constrained watershed segmentation (Beare 2006) © 2006 IEEE: (a) original confocal microscopy image with marked seeds (line segments); (b) standard watershed segmentation; (c) locally constrained watershed segmentation.

gions based on their relative boundary lengths and the strength of the visible edges at these boundaries.

In data clustering, algorithms can link clusters together based on the distance between their closest points (single-link clustering), their farthest points (complete-link clustering), or something in between (Jain, Topchy, Law *et al.* 2004). Kamvar, Klein, and Manning (2002) provide a probabilistic interpretation of these algorithms and show how additional models can be incorporated within this framework.

A very simple version of pixel-based merging combines adjacent regions whose average color difference is below a threshold or whose regions are too small. Segmenting the image into such *superpixels* (Mori, Ren, Efros *et al.* 2004), which are not semantically meaningful, can be a useful pre-processing stage to make higher-level algorithms such as stereo matching (Zitnick, Kang, Uyttendaele *et al.* 2004; Taguchi, Wilburn, and Zitnick 2008), optic flow (Zitnick, Jojic, and Kang 2005; Brox, Bregler, and Malik 2009), and recognition (Mori, Ren, Efros *et al.* 2004; Mori 2005; Gu, Lim, Arbelaez *et al.* 2009; Lim, Arbeláez, Gu *et al.* 2009) both faster and more robust.

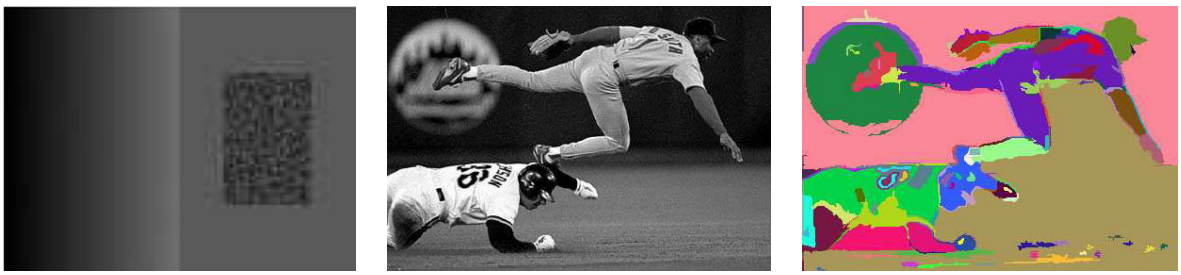
5.2.4 Graph-based segmentation

While many merging algorithms simply apply a fixed rule that groups pixels and regions together, Felzenszwalb and Huttenlocher (2004b) present a merging algorithm that uses *relative dissimilarities* between regions to determine which ones should be merged; it produces an algorithm that provably optimizes a global grouping metric. They start with a pixel-to-pixel dissimilarity measure $w(e)$ that measures, for example, intensity differences between $\mathcal{N}_8$ neighbors. (Alternatively, they can use the *joint feature space* distances (5.42) introduced by Comaniciu and Meer (2002), which we discuss in Section 5.3.2.)

For any region R, its *internal difference* is defined as the largest edge weight in the region's minimum spanning tree,

$$Int(R) = \min_{e \in MST(R)} w(e). \tag{5.20}$$

For any two adjacent regions with at least one edge connecting their vertices, the difference



(a) (b) (c)

Figure 5.14 Graph-based merging segmentation (Felzenszwalb and Huttenlocher 2004b) © 2004 Springer: (a) input grayscale image that is successfully segmented into three regions even though the variation inside the smaller rectangle is larger than the variation across the middle edge; (b) input grayscale image; (c) resulting segmentation using an $\mathcal{N}_8$ pixel neighborhood.

between these regions is defined as the minimum weight edge connecting the two regions,

$$Dif(R_1, R_2) = \min_{e=(v_1,v_2)|v_1 \in R_1, v_2 \in R_2} w(e). \tag{5.21}$$

Their algorithm merges any two adjacent regions whose difference is smaller than the minimum internal difference of these two regions,

$$MInt(R_1, R_2) = \min(Int(R_1) + \tau(R_1), Int(R_2) + \tau(R_2)), \tag{5.22}$$

where $\tau(R)$ is a heuristic region penalty that Felzenszwalb and Huttenlocher (2004b) set to $k/|R|$, but which can be set to any application-specific measure of region goodness.

By merging regions in decreasing order of the edges separating them (which can be efficiently evaluated using a variant of Kruskal's minimum spanning tree algorithm), they provably produce segmentations that are neither too fine (there exist regions that could have been merged) nor too coarse (there are regions that could be split without being mergeable). For fixed-size pixel neighborhoods, the running time for this algorithm is $O(N \log N)$, where N is the number of image pixels, which makes it one of the fastest segmentation algorithms (Paris and Durand 2007). Figure 5.14 shows two examples of images segmented using their technique.

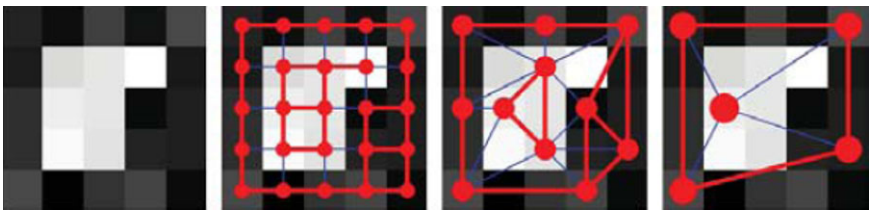
5.2.5 Probabilistic aggregation

Alpert, Galun, Basri *et al.* (2007) develop a probabilistic merging algorithm based on two cues, namely gray-level similarity and texture similarity. The gray-level similarity between regions R_i and R_j is based on the *minimal external difference* from other neighboring regions,

$$\sigma^+_{local} = \min(\Delta^+_i, \Delta^+_j), \tag{5.23}$$

where $\Delta^+_i = \min_k |\Delta_{ik}|$ and Δ_{ik} is the difference in average intensities between regions R_i and R_k. This is compared to the *average intensity difference*,

$$\sigma^-_{local} = \frac{\Delta^-_i + \Delta^-_j}{2}, \tag{5.24}$$



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

Figure 5.15 Coarse to fine node aggregation in segmentation by weighted aggregation (SWA) (Sharon, Galun, Sharon *et al.* 2006) © 2006 Macmillan Publishers Ltd [Nature]: (a) original gray-level pixel grid; (b) inter-pixel couplings, where thicker lines indicate stronger couplings; (c) after one level of coarsening, where each original pixel is strongly coupled to one of the coarse-level nodes; (d) after two levels of coarsening.

where $\Delta_i^- = \sum_k (\tau_{ik}\Delta_{ik}) / \sum_k (\tau_{ik})$ and τ_{ik} is the boundary length between regions R_i and R_k. The texture similarity is defined using relative differences between histogram bins of simple oriented Sobel filter responses. The pairwise statistics σ^+_{local} and σ^-_{local} are used to compute the likelihoods p_{ij} that two regions should be merged. (See the paper by Alpert, Galun, Basri *et al.* (2007) for more details.)

Merging proceeds in a hierarchical fashion inspired by algebraic multigrid techniques (Brandt 1986; Briggs, Henson, and McCormick 2000) and previously used by Alpert, Galun, Basri *et al.* (2007) in their segmentation by weighted aggregation (SWA) algorithm (Sharon, Galun, Sharon *et al.* 2006), which we discuss in Section 5.4. A subset of the nodes $C \subset V$ that are (collectively) *strongly coupled* to all of the original nodes (regions) are used to define the problem at a coarser scale (Figure 5.15), where strong coupling is defined as

$$\frac{\sum_{j \in C} p_{ij}}{\sum_{j \in V} p_{ij}} > \phi, \tag{5.25}$$

with ϕ usually set to 0.2. The intensity and texture similarity statistics for the coarser nodes are recursively computed using weighted averaging, where the relative strengths (couplings) between coarse- and fine-level nodes are based on their merge probabilities p_{ij}. This allows the algorithm to run in essentially $O(N)$ time, using the same kind of hierarchical aggregation operations that are used in pyramid-based filtering or preconditioning algorithms. After a segmentation has been identified at a coarser level, the exact memberships of each pixel are computed by propagating coarse-level assignments to their finer-level "children" (Sharon, Galun, Sharon *et al.* 2006; Alpert, Galun, Basri *et al.* 2007). Figure 5.22 shows the segmentations produced by this algorithm compared to other popular segmentation algorithms.

5.3 Mean shift and mode finding

Mean-shift and mode finding techniques, such as k-means and mixtures of Gaussians, model the feature vectors associated with each pixel (e.g., color and position) as samples from an unknown probability density function and then try to find clusters (modes) in this distribution.

Consider the color image shown in Figure 5.16a. How would you segment this image based on color alone? Figure 5.16b shows the distribution of pixels in L*u*v* space, which is equivalent to what a vision algorithm that ignores spatial location would see. To make the

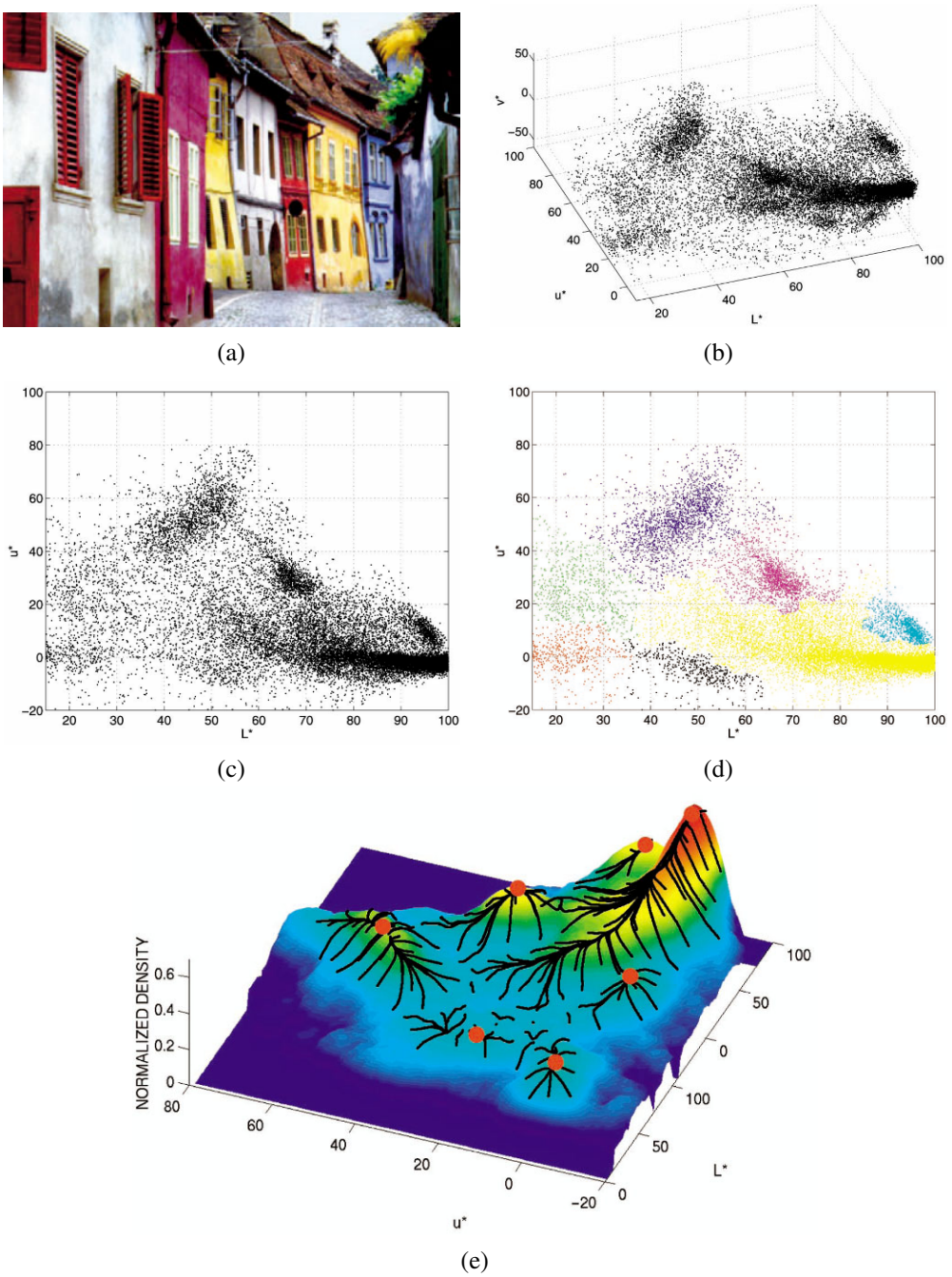


Figure 5.16 Mean-shift image segmentation (Comaniciu and Meer 2002) © 2002 IEEE: (a) input color image; (b) pixels plotted in L*u*v* space; (c) L*u* space distribution; (d) clustered results after 159 mean-shift procedures; (e) corresponding trajectories with peaks marked as red dots.

visualization simpler, let us only consider the L*u* coordinates, as shown in Figure 5.16c. How many obvious (elongated) clusters do you see? How would you go about finding these clusters?

The k-means and mixtures of Gaussians techniques use a *parametric* model of the density function to answer this question, i.e., they assume the density is the superposition of a small number of simpler distributions (e.g., Gaussians) whose locations (centers) and shape (covariance) can be estimated. Mean shift, on the other hand, smoothes the distribution and finds its peaks as well as the regions of feature space that correspond to each peak. Since a complete density is being modeled, this approach is called *non-parametric* (Bishop 2006). Let us look at these techniques in more detail.

5.3.1 K-means and mixtures of Gaussians

While k-means implicitly models the probability density as a superposition of spherically symmetric distributions, it does not require any probabilistic reasoning or modeling (Bishop 2006). Instead, the algorithm is given the number of clusters k it is supposed to find; it then iteratively updates the cluster center location based on the samples that are closest to each center. The algorithm can be initialized by randomly sampling k centers from the input feature vectors. Techniques have also been developed for splitting or merging cluster centers based on their statistics, and for accelerating the process of finding the nearest mean center (Bishop 2006).

In mixtures of Gaussians, each cluster center is augmented by a covariance matrix whose values are re-estimated from the corresponding samples. Instead of using nearest neighbors to associate input samples with cluster centers, a *Mahalanobis distance* (Appendix B.1.1) is used:

$$d(\boldsymbol{x}_i, \boldsymbol{\mu}_k; \boldsymbol{\Sigma}_k) = \|\boldsymbol{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k^{-1}} = (\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_k) \tag{5.26}$$

where $\boldsymbol{x}_i$ are the input samples, $\boldsymbol{\mu}_k$ are the cluster centers, and $\boldsymbol{\Sigma}_k$ are their covariance estimates. Samples can be associated with the nearest cluster center (a *hard assignment* of membership) or can be *softly assigned* to several nearby clusters.

This latter, more commonly used, approach corresponds to iteratively re-estimating the parameters for a mixture of Gaussians density function,

$$p(\boldsymbol{x}|\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}) = \sum_k \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k), \tag{5.27}$$

where π_k are the *mixing coefficients*, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the Gaussian means and covariances, and

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{|\boldsymbol{\Sigma}_k|} e^{-d(\boldsymbol{x}, \boldsymbol{\mu}_k; \boldsymbol{\Sigma}_k)} \tag{5.28}$$

is the *normal* (Gaussian) distribution (Bishop 2006).

To iteratively compute (a local) maximum likely estimate for the unknown mixture parameters $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$, the *expectation maximization* (EM) algorithm (Dempster, Laird, and Rubin 1977) proceeds in two alternating stages:

1. The *expectation* stage (E step) estimates the *responsibilities*

$$z_{ik} = \frac{1}{Z_i} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad \text{with} \quad \sum_k z_{ik} = 1, \tag{5.29}$$

which are the estimates of how likely a sample $\boldsymbol{x}_i$ was generated from the kth Gaussian cluster.

2. The *maximization* stage (M step) updates the parameter values

$$\boldsymbol{\mu}_k = \frac{1}{N_k}\sum_i z_{ik}\boldsymbol{x}_i, \tag{5.30}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k}\sum_i z_{ik}(\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T, \tag{5.31}$$

$$\pi_k = \frac{N_k}{N}, \tag{5.32}$$

where

$$N_k = \sum_i z_{ik}. \tag{5.33}$$

is an estimate of the number of sample points assigned to each cluster.

Bishop (2006) has a wonderful exposition of both mixture of Gaussians estimation and the more general topic of expectation maximization.

In the context of image segmentation, Ma, Derksen, Hong *et al.* (2007) present a nice review of segmentation using mixtures of Gaussians and develop their own extension based on Minimum Description Length (MDL) coding, which they show produces good results on the Berkeley segmentation database.

5.3.2 Mean shift

While k-means and mixtures of Gaussians use a parametric form to model the probability density function being segmented, mean shift implicitly models this distribution using a smooth continuous *non-parametric model*. The key to mean shift is a technique for efficiently finding peaks in this high-dimensional data distribution without ever computing the complete function explicitly (Fukunaga and Hostetler 1975; Cheng 1995; Comaniciu and Meer 2002).

Consider once again the data points shown in Figure 5.16c, which can be thought of as having been drawn from some probability density function. If we could compute this density function, as visualized in Figure 5.16e, we could find its major peaks (*modes*) and identify regions of the input space that climb to the same peak as being part of the same region. This is the inverse of the *watershed* algorithm described in Section 5.2.1, which climbs downhill to find *basins of attraction*.

The first question, then, is how to estimate the density function given a sparse set of samples. One of the simplest approaches is to just smooth the data, e.g., by convolving it with a fixed kernel of width h,

$$f(\boldsymbol{x}) = \sum_i K(\boldsymbol{x} - \boldsymbol{x}_i) = \sum_i k\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h^2}\right), \tag{5.34}$$

where $\boldsymbol{x}_i$ are the input samples and $k(r)$ is the kernel function (or *Parzen window*).[9] This approach is known as *kernel density estimation* or the *Parzen window technique* (Duda, Hart,

[9] In this simplified formula, a Euclidean metric is used. We discuss a little later (5.42) how to generalize this to non-uniform (scaled or oriented) metrics. Note also that this distribution may not be *proper*, i.e., integrate to 1. Since we are looking for maxima in the density, this does not matter.

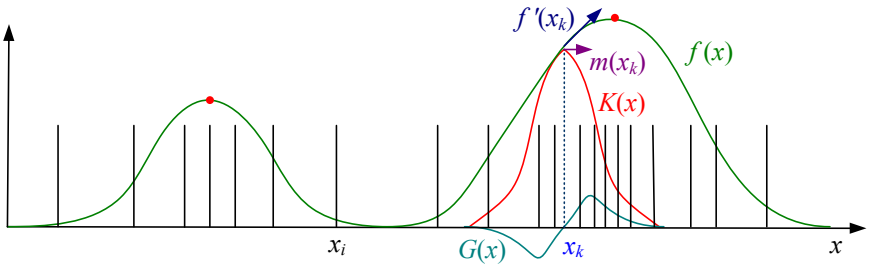


Figure 5.17 One-dimensional visualization of the kernel density estimate, its derivative, and a mean shift. The kernel density estimate $f(x)$ is obtained by convolving the sparse set of input samples x_i with the kernel function $K(x)$. The derivative of this function, $f'(x)$, can be obtained by convolving the inputs with the derivative kernel $G(x)$. Estimating the local displacement vectors around a current estimate x_k results in the mean-shift vector $m(x_k)$, which, in a multi-dimensional setting, point in the same direction as the function gradient $\nabla f(\boldsymbol{x}_k)$. The red dots indicate local maxima in $f(x)$ to which the mean shifts converge.

and Stork 2001, Section 4.3; Bishop 2006, Section 2.5.1). Once we have computed $f(\boldsymbol{x})$, as shown in Figures 5.16e and 5.17, we can find its local maxima using gradient ascent or some other optimization technique.

The problem with this "brute force" approach is that, for higher dimensions, it becomes computationally prohibitive to evaluate $f(\boldsymbol{x})$ over the complete search space.[10] Instead, mean shift uses a variant of what is known in the optimization literature as *multiple restart gradient descent*. Starting at some guess for a local maximum, $\boldsymbol{y}_k$, which can be a random input data point $\boldsymbol{x}_i$, mean shift computes the gradient of the density estimate $f(\boldsymbol{x})$ at $\boldsymbol{y}_k$ and takes an uphill step in that direction (Figure 5.17). The gradient of $f(\boldsymbol{x})$ is given by

$$\nabla f(\boldsymbol{x}) = \sum_i (\boldsymbol{x}_i - \boldsymbol{x}) G(\boldsymbol{x} - \boldsymbol{x}_i) = \sum_i (\boldsymbol{x}_i - \boldsymbol{x}) g\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h^2}\right), \tag{5.35}$$

where

$$g(r) = -k'(r), \tag{5.36}$$

and $k'(r)$ is the first derivative of $k(r)$. We can re-write the gradient of the density function as

$$\nabla f(\boldsymbol{x}) = \left[\sum_i G(\boldsymbol{x} - \boldsymbol{x}_i)\right] \boldsymbol{m}(\boldsymbol{x}), \tag{5.37}$$

where the vector

$$\boldsymbol{m}(\boldsymbol{x}) = \frac{\sum_i \boldsymbol{x}_i G(\boldsymbol{x} - \boldsymbol{x}_i)}{\sum_i G(\boldsymbol{x} - \boldsymbol{x}_i)} - \boldsymbol{x} \tag{5.38}$$

is called the *mean shift*, since it is the difference between the weighted mean of the neighbors $\boldsymbol{x}_i$ around $\boldsymbol{x}$ and the current value of $\boldsymbol{x}$.

In the mean-shift procedure, the current estimate of the mode $\boldsymbol{y}_k$ at iteration k is replaced by its locally weighted mean,

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \boldsymbol{m}(\boldsymbol{y}_k) = \frac{\sum_i \boldsymbol{x}_i G(\boldsymbol{y}_k - \boldsymbol{x}_i)}{\sum_i G(\boldsymbol{y}_k - \boldsymbol{x}_i)}. \tag{5.39}$$

[10] Even for one dimension, if the space is extremely sparse, it may be inefficient.

Comaniciu and Meer (2002) prove that this algorithm converges to a local maximum of $f(\boldsymbol{x})$ under reasonably weak conditions on the kernel $k(r)$, i.e., that it is monotonically decreasing. This convergence is not guaranteed for regular gradient descent unless appropriate step size control is used.

The two kernels that Comaniciu and Meer (2002) studied are the Epanechnikov kernel,

$$k_E(r) = \max(0, 1 - r), \tag{5.40}$$

which is a radial generalization of a bilinear kernel, and the Gaussian (normal) kernel,

$$k_N(r) = \exp\left(-\frac{1}{2}r\right). \tag{5.41}$$

The corresponding derivative kernels $g(r)$ are a unit ball and another Gaussian, respectively. Using the Epanechnikov kernel converges in a finite number of steps, while the Gaussian kernel has a smoother trajectory (and produces better results), but converges very slowly near a mode (Exercise 5.5).

The simplest way to apply mean shift is to start a separate mean-shift mode estimate $\boldsymbol{y}$ at every input point $\boldsymbol{x}_i$ and to iterate for a fixed number of steps or until the mean-shift magnitude is below a threshold. A faster approach is to randomly subsample the input points $\boldsymbol{x}_i$ and to keep track of each point's temporal evolution. The remaining points can then be classified based on the nearest evolution path (Comaniciu and Meer 2002). Paris and Durand (2007) review a number of other more efficient implementations of mean shift, including their own approach, which is based on using an efficient low-resolution estimate of the complete multi-dimensional space of $f(\boldsymbol{x})$ along with its stationary points.

The color-based segmentation shown in Figure 5.16 only looks at pixel colors when determining the best clustering. It may therefore cluster together small isolated pixels that happen to have the same color, which may not correspond to a semantically meaningful segmentation of the image.

Better results can usually be obtained by clustering in the *joint domain* of color and location. In this approach, the spatial coordinates of the image $\boldsymbol{x}_s = (x, y)$, which are called the *spatial domain*, are concatenated with the color values $\boldsymbol{x}_r$, which are known as the *range domain*, and mean-shift clustering is applied in this five-dimensional space $\boldsymbol{x}_j$. Since location and color may have different scales, the kernels are adjusted accordingly, i.e., we use a kernel of the form

$$K(\boldsymbol{x}_j) = k\left(\frac{\|\boldsymbol{x}_r\|^2}{h_r^2}\right) k\left(\frac{\|\boldsymbol{x}_s\|^2}{h_s^2}\right), \tag{5.42}$$

where separate parameters h_s and h_r are used to control the spatial and range bandwidths of the filter kernels. Figure 5.18 shows an example of mean-shift clustering in the joint domain, with parameters $(h_s, h_r, M) = (16, 19, 40)$, where spatial regions containing less than M pixels are eliminated.

The form of the joint domain filter kernel (5.42) is reminiscent of the bilateral filter kernel (3.34–3.37) discussed in Section 3.3.1. The difference between mean shift and bilateral filtering, however, is that in mean shift the spatial coordinates of each pixel are adjusted along with its color values, so that the pixel migrates more quickly towards other pixels with similar colors, and can therefore later be used for clustering and segmentation.

Determining the best bandwidth parameters h to use with mean shift remains something of an art, although a number of approaches have been explored. These include optimizing

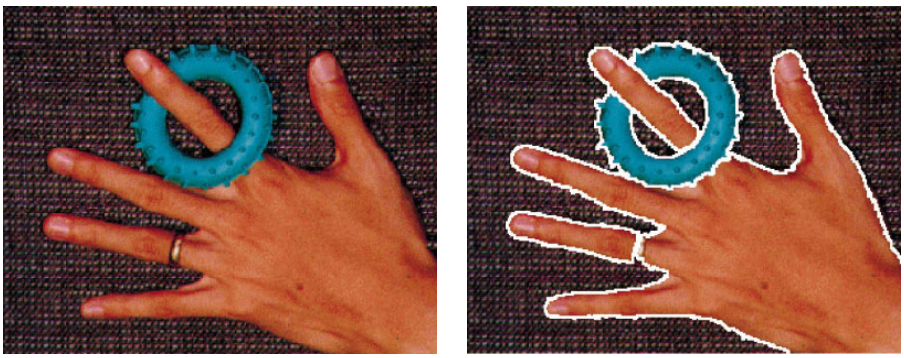


Figure 5.18 Mean-shift color image segmentation with parameters $(h_s, h_r, M) = (16, 19, 40)$ (Comaniciu and Meer 2002) © 2002 IEEE.

the bias–variance tradeoff, looking for parameter ranges where the number of clusters varies slowly, optimizing some external clustering criterion, or using top-down (application domain) knowledge (Comaniciu and Meer 2003). It is also possible to change the orientation of the kernel in joint parameter space for applications such as spatio-temporal (video) segmentations (Wang, Thiesson, Xu *et al.* 2004).

Mean shift has been applied to a number of different problems in computer vision, including face tracking, 2D shape extraction, and texture segmentation (Comaniciu and Meer 2002), and more recently in stereo matching (Chapter 11) (Wei and Quan 2004), non-photorealistic rendering (Section 10.5.2) (DeCarlo and Santella 2002), and video editing (Section 10.4.5) (Wang, Bhat, Colburn *et al.* 2005). Paris and Durand (2007) provide a nice review of such applications, as well as techniques for more efficiently solving the mean-shift equations and producing hierarchical segmentations.

5.4 Normalized cuts

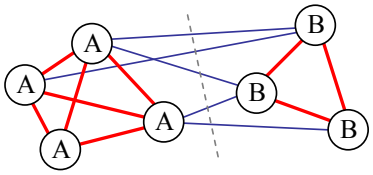
While bottom-up merging techniques aggregate regions into coherent wholes and mean-shift techniques try to find clusters of similar pixels using mode finding, the normalized cuts technique introduced by Shi and Malik (2000) examines the *affinities* (similarities) between nearby pixels and tries to separate groups that are connected by weak affinities.

Consider the simple graph shown in Figure 5.19a. The pixels in group A are all strongly connected with high affinities, shown as thick red lines, as are the pixels in group B. The connections between these two groups, shown as thinner blue lines, are much weaker. A *normalized cut* between the two groups, shown as a dashed line, separates them into two clusters.

The cut between two groups A and B is defined as the sum of all the weights being cut,

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}, \tag{5.43}$$

where the weights between two pixels (or regions) i and j measure their similarity. Using a minimum cut as a segmentation criterion, however, does not result in reasonable clusters, since the smallest cuts usually involve isolating a single pixel.



(a)

	A	B	sum
A	$assoc(A, A)$	$cut(A, B)$	$assoc(A, V)$
B	$cut(B, A)$	$assoc(B, B)$	$assoc(B, V)$
sum	$assoc(A, V)$	$assoc(B, v)$	

(b)

Figure 5.19 Sample weighted graph and its normalized cut: (a) a small sample graph and its smallest normalized cut; (b) tabular form of the associations and cuts for this graph. The $assoc$ and cut entries are computed as area sums of the associated weight matrix $\boldsymbol{W}$ (Figure 5.20). Normalizing the table entries by the row or column sums produces normalized associations and cuts $Nassoc$ and $Ncut$.

A better measure of segmentation is the normalized cut, which is defined as

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}, \tag{5.44}$$

where $assoc(A, A) = \sum_{i \in A, j \in A} w_{ij}$ is the *association* (sum of all the weights) within a cluster and $assoc(A, V) = assoc(A, A) + cut(A, B)$ is the sum of *all* the weights associated with nodes in A. Figure 5.19b shows how the cuts and associations can be thought of as area sums in the weight matrix $\boldsymbol{W} = [w_{ij}]$, where the entries of the matrix have been arranged so that the nodes in A come first and the nodes in B come second. Figure 5.20 shows an actual weight matrix for which these area sums can be computed. Dividing each of these areas by the corresponding row sum (the rightmost column of Figure 5.19b) results in the normalized cut and association values. These normalized values better reflect the fitness of a particular segmentation, since they look for collections of edges that are weak relative to all of the edges both inside and emanating from a particular region.

Unfortunately, computing the optimal normalized cut is NP-complete. Instead, Shi and Malik (2000) suggest computing a real-valued assignment of nodes to groups. Let $\boldsymbol{x}$ be the *indicator vector* where $x_i = +1$ iff $i \in A$ and $x_i = -1$ iff $i \in B$. Let $\boldsymbol{d} = \boldsymbol{W}\boldsymbol{1}$ be the row sums of the symmetric matrix $\boldsymbol{W}$ and $\boldsymbol{D} = \text{diag}(\boldsymbol{d})$ be the corresponding diagonal matrix. Shi and Malik (2000) show that minimizing the normalized cut over all possible indicator vectors $\boldsymbol{x}$ is equivalent to minimizing

$$\min_{\boldsymbol{y}} \frac{\boldsymbol{y}^T(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y}}, \tag{5.45}$$

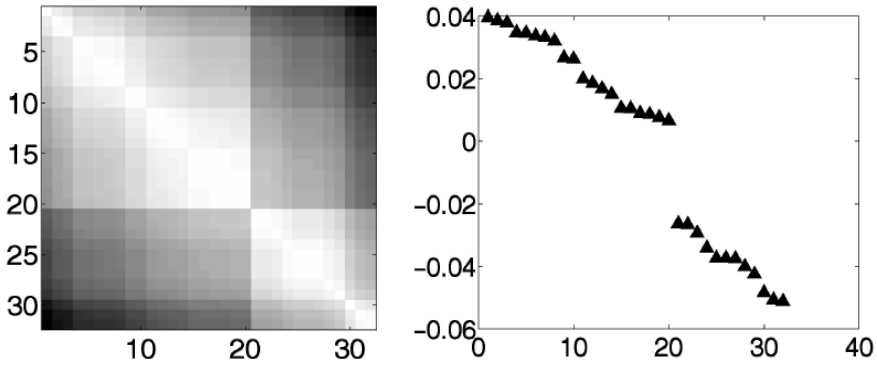
where $\boldsymbol{y} = ((\boldsymbol{1} + \boldsymbol{x}) - b(\boldsymbol{1} - \boldsymbol{x}))/2$ is a vector consisting of all 1s and $-b$s such that $\boldsymbol{y} \cdot \boldsymbol{d} = 0$. Minimizing this *Rayleigh quotient* is equivalent to solving the generalized eigenvalue system

$$(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{y} = \lambda \boldsymbol{D}\boldsymbol{y}, \tag{5.46}$$

which can be turned into a regular eigenvalue problem

$$(\boldsymbol{I} - \boldsymbol{N})\boldsymbol{z} = \lambda \boldsymbol{z}, \tag{5.47}$$

where $\boldsymbol{N} = \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$ is the *normalized* affinity matrix (Weiss 1999) and $\boldsymbol{z} = \boldsymbol{D}^{1/2}\boldsymbol{y}$. Because these eigenvectors can be interpreted as the large modes of vibration in



(a) (b)

Figure 5.20 Sample weight table and its second smallest eigenvector (Shi and Malik 2000) © 2000 IEEE: (a) sample 32×32 weight matrix $\boldsymbol{W}$; (b) eigenvector corresponding to the second smallest eigenvalue of the generalized eigenvalue problem $(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{y} = \lambda \boldsymbol{D}\boldsymbol{y}$.

a spring-mass system, normalized cuts is an example of a *spectral method* for image segmentation.

Extending an idea originally proposed by Scott and Longuet-Higgins (1990), Weiss (1999) suggests normalizing the affinity matrix and then using the top k eigenvectors to reconstitute a $\boldsymbol{Q}$ matrix. Other papers have extended the basic normalized cuts framework by modifying the affinity matrix in different ways, finding better discrete solutions to the minimization problem, or applying multi-scale techniques (Meilă and Shi 2000, 2001; Ng, Jordan, and Weiss 2001; Yu and Shi 2003; Cour, Bénézit, and Shi 2005; Tolliver and Miller 2006).

Figure 5.20b shows the second smallest (real-valued) eigenvector corresponding to the weight matrix shown in Figure 5.20a. (Here, the rows have been permuted to separate the two groups of variables that belong to the different components of this eigenvector.) After this real-valued vector is computed, the variables corresponding to positive and negative eigenvector values are associated with the two cut components. This process can be further repeated to hierarchically subdivide an image, as shown in Figure 5.21.

The original algorithm proposed by Shi and Malik (2000) used spatial position and image feature differences to compute the pixel-wise affinities,

$$w_{ij} = \exp\left(-\frac{\|\boldsymbol{F}_i - \boldsymbol{F}_j\|^2}{\sigma_F^2} - \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\sigma_s^2}\right), \tag{5.48}$$

for pixels within a radius $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| < r$, where $\boldsymbol{F}$ is a feature vector that consists of intensities, colors, or oriented filter histograms. (Note how (5.48) is the negative exponential of the joint feature space distance (5.42).)

In subsequent work, Malik, Belongie, Leung *et al.* (2001) look for *intervening contours* between pixels i and j and define an intervening contour weight

$$w_{ij}^{IC} = 1 - \max_{\boldsymbol{x} \in l_{ij}} p_{con}(\boldsymbol{x}), \tag{5.49}$$

where l_{ij} is the image line joining pixels i and j and $p_{con}(\boldsymbol{x})$ is the probability of an intervening contour perpendicular to this line, which is defined as the negative exponential of the

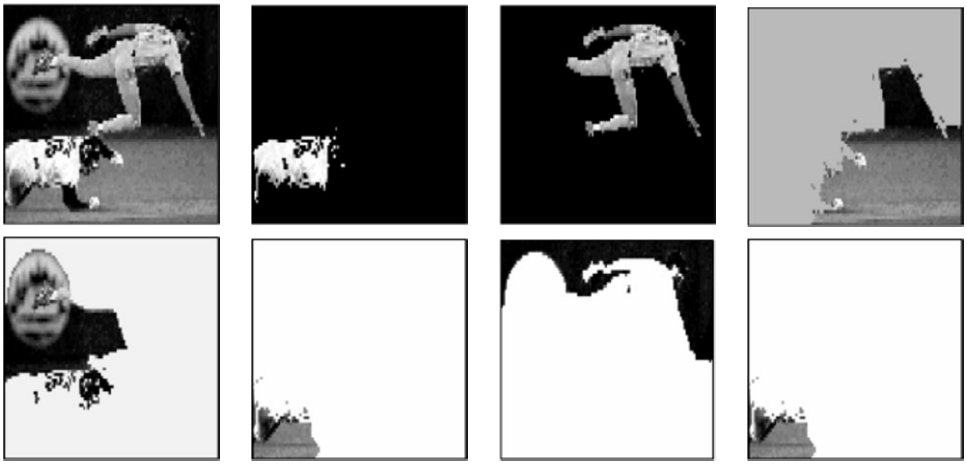


Figure 5.21 Normalized cuts segmentation (Shi and Malik 2000) © 2000 IEEE: The input image and the components returned by the normalized cuts algorithm.

oriented energy in the perpendicular direction. They multiply these weights with a texton-based texture similarity metric and use an initial over-segmentation based purely on local pixel-wise features to re-estimate intervening contours and texture statistics in a region-based manner. Figure 5.22 shows the results of running this improved algorithm on a number of test images.

Because it requires the solution of large sparse eigenvalue problems, normalized cuts can be quite slow. Sharon, Galun, Sharon *et al.* (2006) present a way to accelerate the computation of the normalized cuts using an approach inspired by algebraic multigrid (Brandt 1986; Briggs, Henson, and McCormick 2000). To coarsen the original problem, they select a smaller number of variables such that the remaining fine-level variables are *strongly coupled* to at least one coarse-level variable. Figure 5.15 shows this process schematically, while (5.25) gives the definition for strong coupling except that, in this case, the original weights w_{ij} in the normalized cut are used instead of merge probabilities p_{ij}.

Once a set of coarse variables has been selected, an inter-level interpolation matrix with elements similar to the left hand side of (5.25) is used to define a reduced version of the normalized cuts problem. In addition to computing the weight matrix using interpolation-based coarsening, additional region statistics are used to modulate the weights. After a normalized cut has been computed at the coarsest level of analysis, the membership values of finer-level nodes are computed by interpolating parent values and mapping values within $\epsilon = 0.1$ of 0 and 1 to pure Boolean values.

An example of the segmentation produced by weighted aggregation (SWA) is shown in Figure 5.22, along with the most recent probabilistic bottom-up merging algorithm by Alpert, Galun, Basri *et al.* (2007), which was described in Section 5.2. In even more recent work, Wang and Oliensis (2010) show how to estimate statistics over segmentations (e.g., mean region size) directly from the affinity graph. They use this to produce segmentations that are more *central* with respect to other possible segmentations.

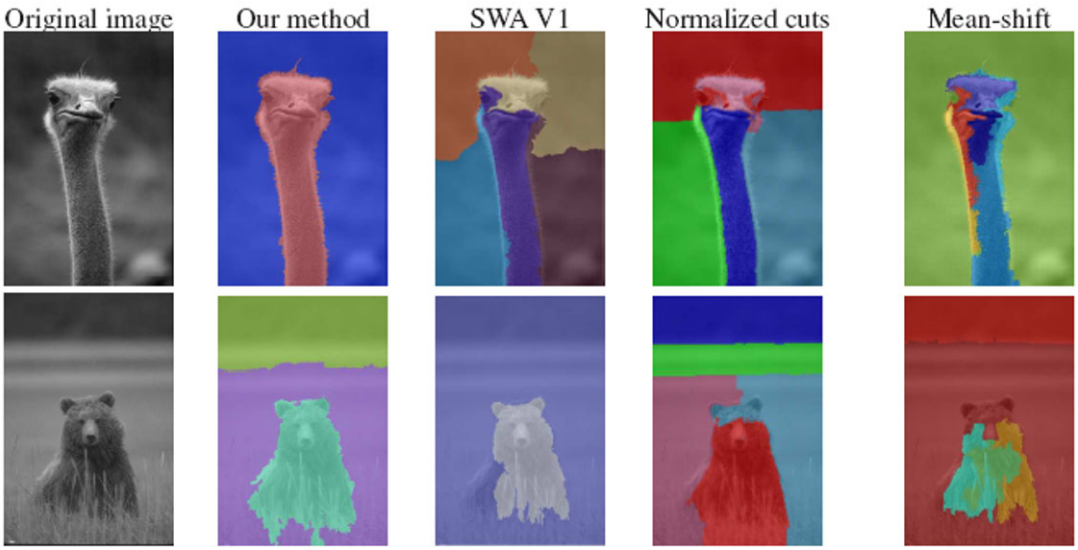


Figure 5.22 Comparative segmentation results (Alpert, Galun, Basri *et al.* 2007) © 2007 IEEE. "Our method" refers to the probabilistic bottom-up merging algorithm developed by Alpert *et al*.

5.5 Graph cuts and energy-based methods

A common theme in image segmentation algorithms is the desire to group pixels that have similar appearance (statistics) and to have the boundaries between pixels in different regions be of short length and across visible discontinuities. If we restrict the boundary measurements to be between immediate neighbors and compute region membership statistics by summing over pixels, we can formulate this as a classic pixel-based energy function using either a *variational formulation* (regularization, see Section 3.7.1) or as a binary Markov random field (Section 3.7.2).

Examples of the continuous approach include (Mumford and Shah 1989; Chan and Vese 1992; Zhu and Yuille 1996; Tabb and Ahuja 1997) along with the level set approaches discussed in Section 5.1.4. An early example of a discrete labeling problem that combines both region-based and boundary-based energy terms is the work of Leclerc (1989), who used minimum description length (MDL) coding to derive the energy function being minimized. Boykov and Funka-Lea (2006) present a wonderful survey of various energy-based techniques for binary object segmentation, some of which we discuss below.

As we saw in Section 3.7.2, the energy corresponding to a segmentation problem can be written (c.f. Equations (3.100) and (3.108–3.113)) as

$$E(f) = \sum_{i,j} E_r(i,j) + E_b(i,j), \tag{5.50}$$

where the region term

$$E_r(i,j) = E_S(I(i,j); R(f(i,j))) \tag{5.51}$$

is the negative log likelihood that pixel intensity (or color) $I(i,j)$ is consistent with the statis-

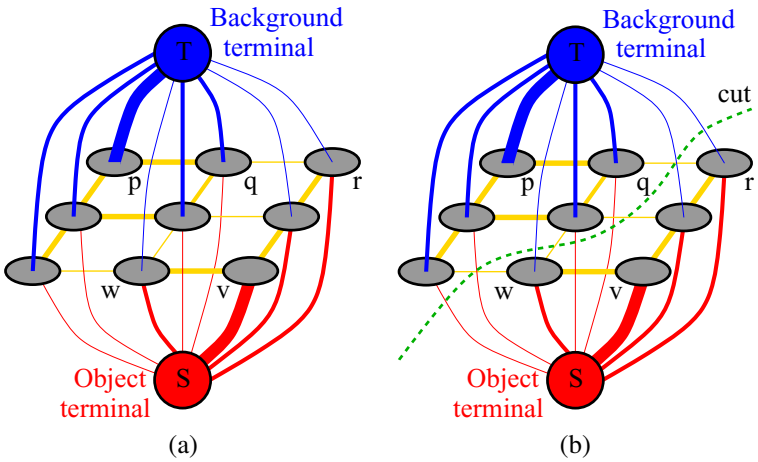


Figure 5.23 Graph cuts for region segmentation (Boykov and Jolly 2001) © 2001 IEEE: (a) the energy function is encoded as a maximum flow problem; (b) the minimum cut determines the region boundary.

tics of region $R(f(i,j))$ and the boundary term

$$E_b(i,j) = s_x(i,j)\delta(f(i,j) - f(i+1,j)) + s_y(i,j)\delta(f(i,j) - f(i,j+1)) \tag{5.52}$$

measures the inconsistency between $\mathcal{N}_4$ neighbors modulated by local horizontal and vertical smoothness terms $s_x(i,j)$ and $s_y(i,j)$.

Region statistics can be something as simple as the mean gray level or color (Leclerc 1989), in which case

$$E_S(I;\mu_k) = \|I - \mu_k\|^2. \tag{5.53}$$

Alternatively, they can be more complex, such as region intensity histograms (Boykov and Jolly 2001) or color Gaussian mixture models (Rother, Kolmogorov, and Blake 2004). For smoothness (boundary) terms, it is common to make the strength of the smoothness $s_x(i,j)$ inversely proportional to the local edge strength (Boykov, Veksler, and Zabih 2001).

Originally, energy-based segmentation problems were optimized using iterative gradient descent techniques, which were slow and prone to getting trapped in local minima. Boykov and Jolly (2001) were the first to apply the binary MRF optimization algorithm developed by Greig, Porteous, and Seheult (1989) to binary object segmentation.

In this approach, the user first delineates pixels in the background and foreground regions using a few strokes of an image brush (Figure 3.61). These pixels then become the *seeds* that tie nodes in the *S–T graph* to the source and sink labels S and T (Figure 5.23a). Seed pixels can also be used to estimate foreground and background region statistics (intensity or color histograms).

The capacities of the other edges in the graph are derived from the region and boundary energy terms, i.e., pixels that are more compatible with the foreground or background region get stronger connections to the respective source or sink; adjacent pixels with greater smoothness also get stronger links. Once the minimum-cut/maximum-flow problem has been solved using a polynomial time algorithm (Goldberg and Tarjan 1988; Boykov and Kolmogorov 2004), pixels on either side of the computed cut are labeled according to the source or sink to

(a) (b) (c)

Figure 5.24 GrabCut image segmentation (Rother, Kolmogorov, and Blake 2004) © 2004 ACM: (a) the user draws a bounding box in red; (b) the algorithm guesses color distributions for the object and background and performs a binary segmentation; (c) the process is repeated with better region statistics.

which they remain connected (Figure 5.23b). While graph cuts is just one of several known techniques for MRF energy minimization (Appendix B.5.4), it is still the one most commonly used for solving binary MRF problems.

The basic binary segmentation algorithm of Boykov and Jolly (2001) has been extended in a number of directions. The *GrabCut* system of Rother, Kolmogorov, and Blake (2004) iteratively re-estimates the region statistics, which are modeled as a mixtures of Gaussians in color space. This allows their system to operate given minimal user input, such as a single bounding box (Figure 5.24a)—the background color model is initialized from a strip of pixels around the box outline. (The foreground color model is initialized from the interior pixels, but quickly converges to a better estimate of the object.) The user can also place additional strokes to refine the segmentation as the solution progresses. In more recent work, Cui, Yang, Wen *et al.* (2008) use color and edge models derived from previous segmentations of similar objects to improve the local models used in GrabCut.

Another major extension to the original binary segmentation formulation is the addition of *directed edges*, which allows boundary regions to be oriented, e.g., to prefer light to dark transitions or *vice versa* (Kolmogorov and Boykov 2005). Figure 5.25 shows an example where the directed graph cut correctly segments the light gray liver from its dark gray surround. The same approach can be used to measure the *flux* exiting a region, i.e., the signed gradient projected normal to the region boundary. Combining oriented graphs with larger neighborhoods enables approximating continuous problems such as those traditionally solved using level sets in the globally optimal graph cut framework (Boykov and Kolmogorov 2003; Kolmogorov and Boykov 2005).

Even more recent developments in graph cut-based segmentation techniques include the addition of connectivity priors to force the foreground to be in a single piece (Vicente, Kolmogorov, and Rother 2008) and shape priors to use knowledge about an object's shape during the segmentation process (Lempitsky and Boykov 2007; Lempitsky, Blake, and Rother 2008).

While optimizing the binary MRF energy (5.50) requires the use of combinatorial optimization techniques, such as maximum flow, an approximate solution can be obtained by converting the binary energy terms into quadratic energy terms defined over a continuous $[0, 1]$ random field, which then becomes a classical membrane-based regularization problem (3.100–3.102). The resulting quadratic energy function can then be solved using standard linear system solvers (3.102–3.103), although if speed is an issue, you should use multigrid

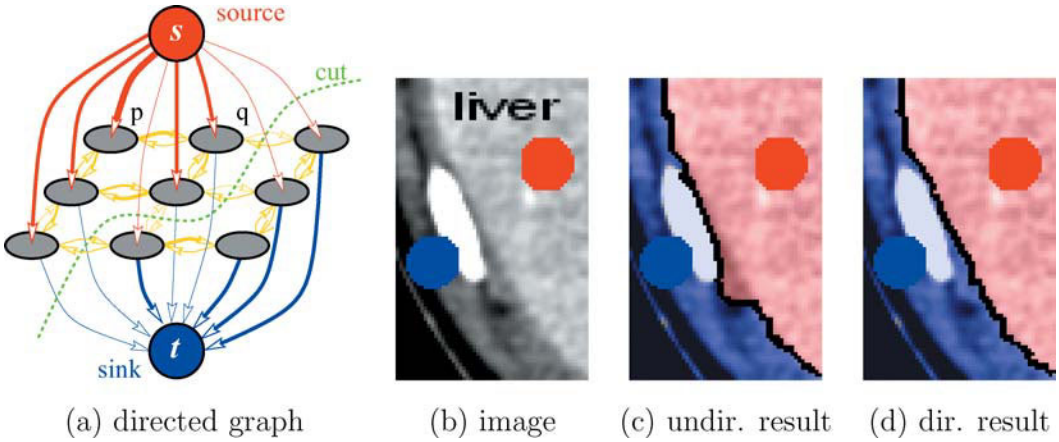
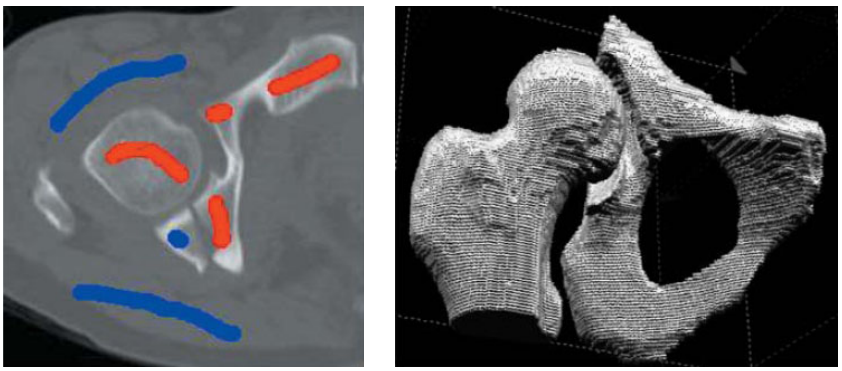


Figure 5.25 Segmentation with a directed graph cut (Boykov and Funka-Lea 2006) © 2006 Springer: (a) directed graph; (b) image with seed points; (c) the undirected graph incorrectly continues the boundary along the bright object; (d) the directed graph correctly segments the light gray region from its darker surround.

or one of its variants (Appendix A.5). Once the continuous solution has been computed, it can be thresholded at 0.5 to yield a binary segmentation.

The $[0, 1]$ continuous optimization problem can also be interpreted as computing the probability at each pixel that a *random walker* starting at that pixel ends up at one of the labeled seed pixels, which is also equivalent to computing the potential in a resistive grid where the resistors are equal to the edge weights (Grady 2006; Sinop and Grady 2007). K-way segmentations can also be computed by iterating through the seed labels, using a binary problem with one label set to 1 and all the others set to 0 to compute the relative membership probabilities for each pixel. In follow-on work, Grady and Ali (2008) use a precomputation of the eigenvectors of the linear system to make the solution with a novel set of seeds faster, which is related to the Laplacian matting problem presented in Section 10.4.3 (Levin, Acha, and Lischinski 2008). Couprie, Grady, Najman *et al.* (2009) relate the random walker to watersheds and other segmentation techniques. Singaraju, Grady, and Vidal (2008) add directed-edge constraints in order to support flux, which makes the energy piecewise quadratic and hence not solvable as a single linear system. The random walker algorithm can also be used to solve the Mumford–Shah segmentation problem (Grady and Alvino 2008) and to compute fast multigrid solutions (Grady 2008). A nice review of these techniques is given by Singaraju, Grady, Sinop *et al.* (2010).

An even faster way to compute a continuous $[0, 1]$ approximate segmentation is to compute *weighted geodesic distances* between the 0 and 1 seed regions (Bai and Sapiro 2009), which can also be used to estimate soft alpha mattes (Section 10.4.3). A related approach by Criminisi, Sharp, and Blake (2008) can be used to find fast approximate solutions to general binary Markov random field optimization problems.



(a) (b)

Figure 5.26 3D volumetric medical image segmentation using graph cuts (Boykov and Funka-Lea 2006) © 2006 Springer: (a) computed tomography (CT) slice with some seeds; (b) recovered 3D volumetric bone model (on a $256 \times 256 \times 119$ voxel grid).

5.5.1 *Application*: Medical image segmentation

One of the most promising applications of image segmentation is in the medical imaging domain, where it can be used to segment anatomical tissues for later quantitative analysis. Figure 5.25 shows a binary graph cut with directed edges being used to segment the liver tissue (light gray) from its surrounding bone (white) and muscle (dark gray) tissue. Figure 5.26 shows the segmentation of bones in a $256 \times 256 \times 119$ computed X-ray tomography (CT) volume. Without the powerful optimization techniques available in today's image segmentation algorithms, such processing used to require much more laborious manual tracing of individual X-ray slices.

The fields of medical image segmentation (McInerney and Terzopoulos 1996) and medical image registration (Kybic and Unser 2003) (Section 8.3.1) are rich research fields with their own specialized conferences, such as *Medical Imaging Computing and Computer Assisted Intervention (MICCAI)*,[11] and journals, such as *Medical Image Analysis* and *IEEE Transactions on Medical Imaging*. These can be great sources of references and ideas for research in this area.

5.6 Additional reading

The topic of image segmentation is closely related to clustering techniques, which are treated in a number of monographs and review articles (Jain and Dubes 1988; Kaufman and Rousseeuw 1990; Jain, Duin, and Mao 2000; Jain, Topchy, Law *et al.* 2004). Some early segmentation techniques include those describerd by Brice and Fennema (1970); Pavlidis (1977); Riseman and Arbib (1977); Ohlander, Price, and Reddy (1978); Rosenfeld and Davis (1979); Haralick and Shapiro (1985), while examples of newer techniques are developed by Leclerc (1989); Mumford and Shah (1989); Shi and Malik (2000); Felzenszwalb and Huttenlocher (2004b).

[11] http://www.miccai.org/.

Arbeláez, Maire, Fowlkes *et al.* (2010) provide a good review of automatic segmentation techniques and also compare their performance on the Berkeley Segmentation Dataset and Benchmark (Martin, Fowlkes, Tal *et al.* 2001).[12] Additional comparison papers and databases include those by Unnikrishnan, Pantofaru, and Hebert (2007); Alpert, Galun, Basri *et al.* (2007); Estrada and Jepson (2009).

The topic of active contours has a long history, beginning with the seminal work on snakes and other energy-minimizing variational methods (Kass, Witkin, and Terzopoulos 1988; Cootes, Cooper, Taylor *et al.* 1995; Blake and Isard 1998), continuing through techniques such as intelligent scissors (Mortensen and Barrett 1995, 1999; Pérez, Blake, and Gangnet 2001), and culminating in level sets (Malladi, Sethian, and Vemuri 1995; Caselles, Kimmel, and Sapiro 1997; Sethian 1999; Paragios and Deriche 2000; Sapiro 2001; Osher and Paragios 2003; Paragios, Faugeras, Chan *et al.* 2005; Cremers, Rousson, and Deriche 2007; Rousson and Paragios 2008; Paragios and Sgallari 2009), which are currently the most widely used active contour methods.

Techniques for segmenting images based on local pixel similarities combined with aggregation or splitting methods include watersheds (Vincent and Soille 1991; Beare 2006; Arbeláez, Maire, Fowlkes *et al.* 2010), region splitting (Ohlander, Price, and Reddy 1978), region merging (Brice and Fennema 1970; Pavlidis and Liow 1990; Jain, Topchy, Law *et al.* 2004), as well as graph-based and probabilistic multi-scale approaches (Felzenszwalb and Huttenlocher 2004b; Alpert, Galun, Basri *et al.* 2007).

Mean-shift algorithms, which find modes (peaks) in a density function representation of the pixels, are presented by Comaniciu and Meer (2002); Paris and Durand (2007). Parametric mixtures of Gaussians can also be used to represent and segment such pixel densities (Bishop 2006; Ma, Derksen, Hong *et al.* 2007).

The seminal work on spectral (eigenvalue) methods for image segmentation is the *normalized cut* algorithm of Shi and Malik (2000). Related work includes that by Weiss (1999); Meilă and Shi (2000, 2001); Malik, Belongie, Leung *et al.* (2001); Ng, Jordan, and Weiss (2001); Yu and Shi (2003); Cour, Bénézit, and Shi (2005); Sharon, Galun, Sharon *et al.* (2006); Tolliver and Miller (2006); Wang and Oliensis (2010).

Continuous-energy-based (variational) approaches to interactive segmentation include Leclerc (1989); Mumford and Shah (1989); Chan and Vese (1992); Zhu and Yuille (1996); Tabb and Ahuja (1997). Discrete variants of such problems are usually optimized using binary graph cuts or other combinatorial energy minimization methods (Boykov and Jolly 2001; Boykov and Kolmogorov 2003; Rother, Kolmogorov, and Blake 2004; Kolmogorov and Boykov 2005; Cui, Yang, Wen *et al.* 2008; Vicente, Kolmogorov, and Rother 2008; Lempitsky and Boykov 2007; Lempitsky, Blake, and Rother 2008), although continuous optimization techniques followed by thresholding can also be used (Grady 2006; Grady and Ali 2008; Singaraju, Grady, and Vidal 2008; Criminisi, Sharp, and Blake 2008; Grady 2008; Bai and Sapiro 2009; Couprie, Grady, Najman *et al.* 2009). Boykov and Funka-Lea (2006) present a good survey of various energy-based techniques for binary object segmentation.

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

5.7 Exercises

Ex 5.1: Snake evolution Prove that, in the absence of external forces, a snake will always shrink to a small circle and eventually a single point, regardless of whether first- or second-order smoothness (or some combination) is used.

(Hint: If you can show that the evolution of the $x(s)$ and $y(s)$ components are independent, you can analyze the 1D case more easily.)

Ex 5.2: Snake tracker Implement a snake-based contour tracker:

1. Decide whether to use a large number of contour points or a smaller number interpolated with a B-spline.
2. Define your internal smoothness energy function and decide what image-based attractive forces to use.
3. At each iteration, set up the banded linear system of equations (quadratic energy function) and solve it using banded Cholesky factorization (Appendix A.4).

Ex 5.3: Intelligent scissors Implement the intelligent scissors (live-wire) interactive segmentation algorithm (Mortensen and Barrett 1995) and design a graphical user interface (GUI) to let you draw such curves over an image and use them for segmentation.

Ex 5.4: Region segmentation Implement one of the region segmentation algorithms described in this chapter. Some popular segmentation algorithms include:

- k-means (Section 5.3.1);
- mixtures of Gaussians (Section 5.3.1);
- mean shift (Section 5.3.2) and Exercise 5.5;
- normalized cuts (Section 5.4);
- similarity graph-based segmentation (Section 5.2.4);
- binary Markov random fields solved using graph cuts (Section 5.5).

Apply your region segmentation to a video sequence and use it to track moving regions from frame to frame.

Alternatively, test out your segmentation algorithm on the Berkeley segmentation database (Martin, Fowlkes, Tal *et al.* 2001).

Ex 5.5: Mean shift Develop a mean-shift segmentation algorithm for color images (Comaniciu and Meer 2002).

1. Convert your image to L*a*b* space, or keep the original RGB colors, and augment them with the pixel (x, y) locations.
2. For every pixel (L, a, b, x, y), compute the weighted mean of its neighbors using either a unit ball (Epanechnikov kernel) or finite-radius Gaussian, or some other kernel of your choosing. Weight the color and spatial scales differently, e.g., using values of $(h_s, h_r, M) = (16, 19, 40)$ as shown in Figure 5.18.

3. Replace the current value with this weighted mean and iterate until either the motion is below a threshold or a finite number of steps has been taken.

4. Cluster all final values (modes) that are within a threshold, i.e., find the connected components. Since each pixel is associated with a final mean-shift (mode) value, this results in an image segmentation, i.e., each pixel is labeled with its final component.

5. (Optional) Use a random subset of the pixels as starting points and find which component each unlabeled pixel belongs to, either by finding its nearest neighbor or by iterating the mean shift until it finds a neighboring track of mean-shift values. Describe the data structures you use to make this efficient.

6. (Optional) Mean shift divides the kernel density function estimate by the local weighting to obtain a step size that is guaranteed to converge but may be slow. Use an alternative step size estimation algorithm from the optimization literature to see if you can make the algorithm converge faster.