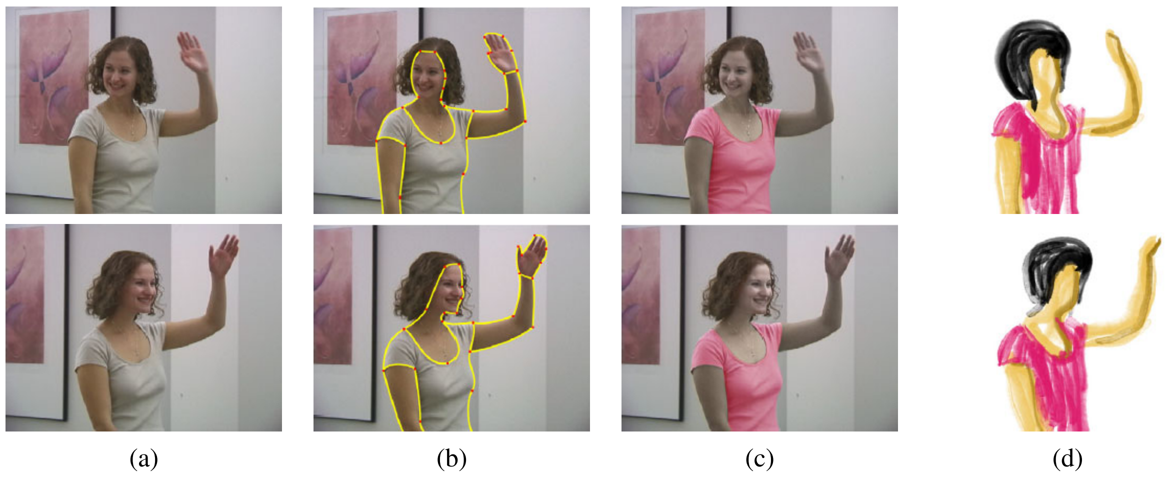

Figure 5.12 Keyframe-based rotoscoping (Agarwala, Hertzmann, Seitz *et al.* 2004) © 2004 ACM: (a) original frames; (b) rotoscoped contours; (c) re-colored blouse; (d) rotoscoped hand-drawn animation.

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).

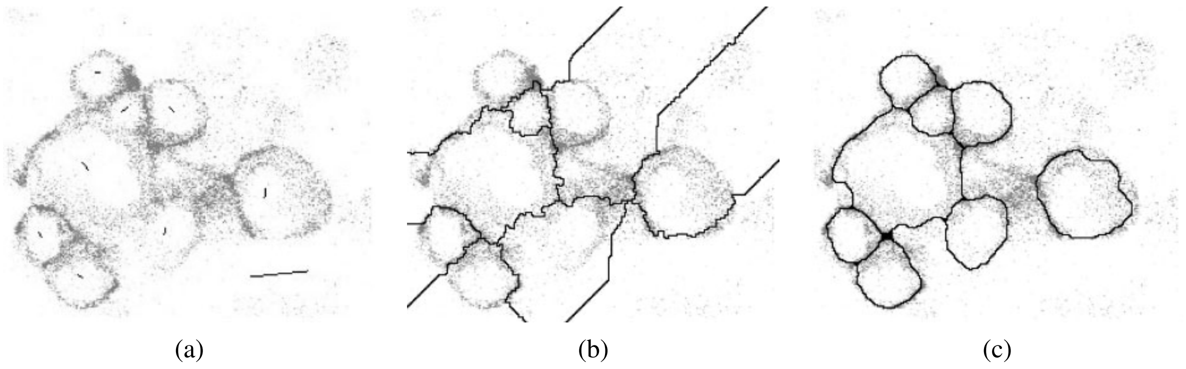

Figure 5.13 Locally constrained watershed segmentation (Beare 2006) © 2006 IEEE: (a) original confocal microscopy image with marked seeds (line segments); (b) standard watershed segmentation; (c) locally constrained watershed segmentation.

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

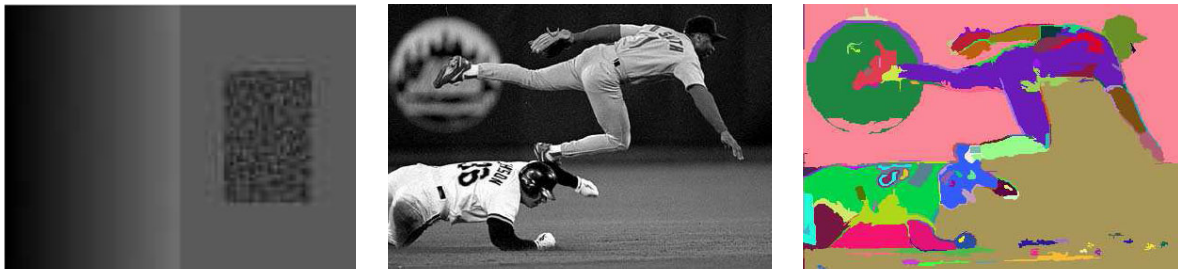

Figure 5.14 Graph-based merging segmentation (Felzenszwalb and Huttenlocher 2004b) © 2004 Springer: (a) input grayscale image that is successfully segmented into three regions even though the variation inside the smaller rectangle is larger than the variation across the middle edge; (b) input grayscale 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*,

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

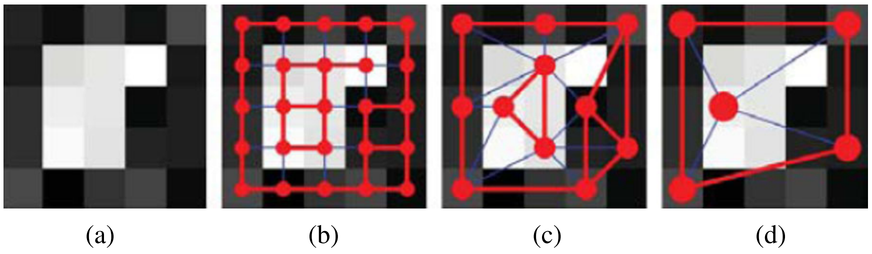

Figure 5.15 Coarse to fine node aggregation in segmentation by weighted aggregation (SWA) (Sharon, Galun, Sharon *et al.* 2006) © 2006 Macmillan Publishers Ltd [Nature]: (a) original gray-level pixel grid; (b) inter-pixel couplings, where thicker lines indicate stronger couplings; (c) after one level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

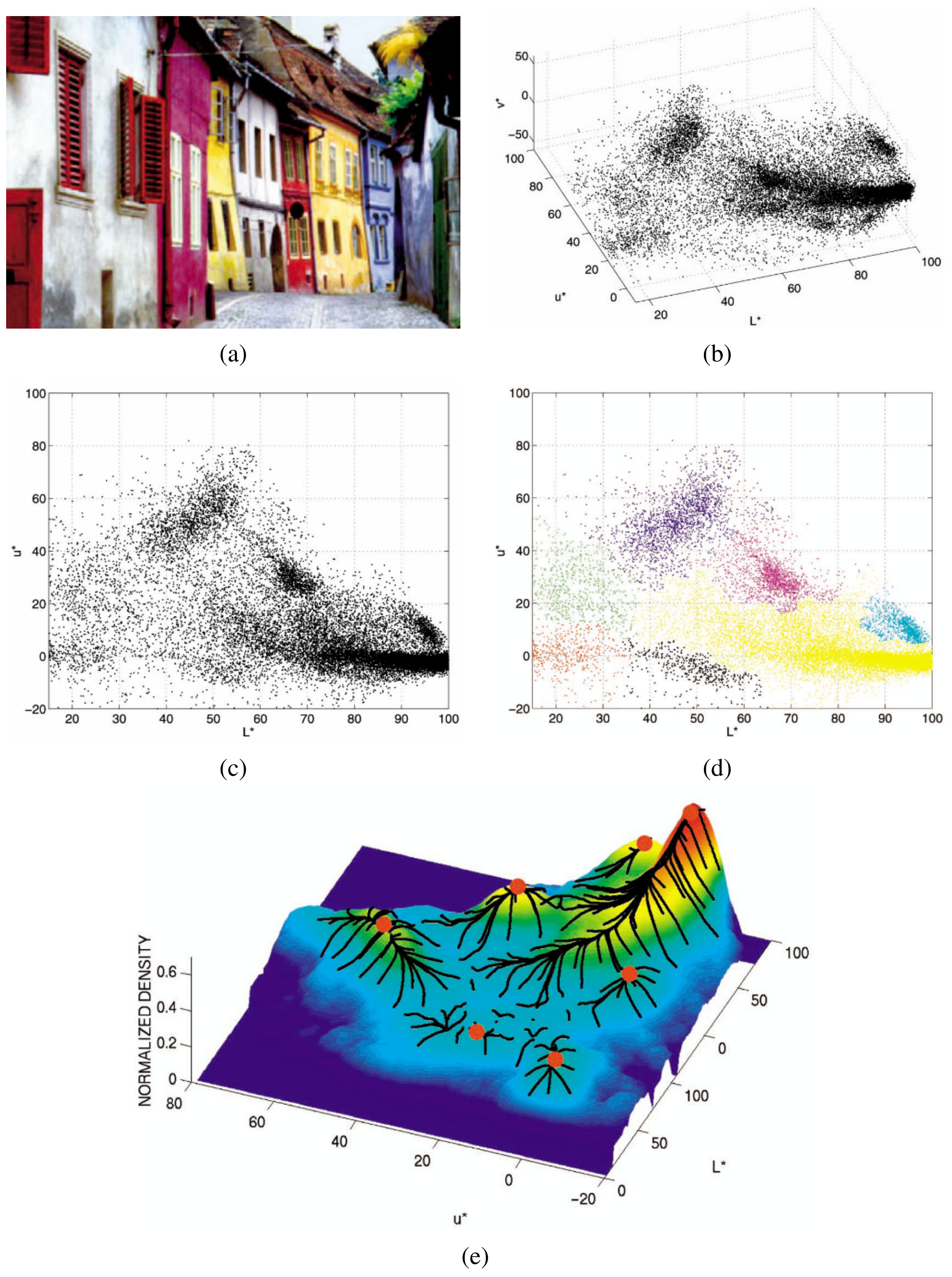

Figure 5.16 Mean-shift image segmentation (Comaniciu and Meer 2002) © 2002 IEEE: (a) input color image; (b) pixels plotted in L*u*v* space; (c) L*u* space distribution; (d) clustered results after 159 mean-shift procedures; (e) corresponding trajectories with peaks marked as red dots.

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.