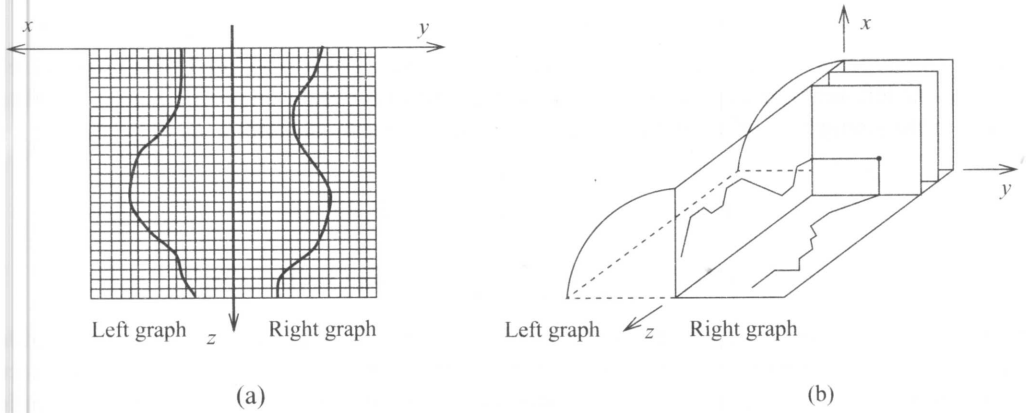


Figure 7.30: Three-dimensional graph construction. (a) Separate identification of the left and right borders by linking nodes in individual two-dimensional graphs corresponding to the left and right halves of the region segment of interest. (b) By rotating up the left graph, a three-dimensional graph results in which paths correspond to pairs of region borders.

The process of constructing the three-dimensional graph can be visualized as one of rotating up the 2D graph corresponding to the pixels left of the approximate region centerline (Figure 7.30b). The result is a three-dimensional array of nodes in which each node corresponds to possible positions of the left and right region borders for a given point along the length of the elongated region, and a path through the graph corresponds to a possible pair of left and right region borders. Nodes in the 3D graph are referenced by their (x, y, z) co-ordinates; for a point along the region centerline defined by the co-ordinate z, a node with co-ordinates (x_l, y_l, z) corresponds to a left border that is x_l pixels to the left of the centerline and a right border that is y_l pixels to the right of the centerline.

As in the 2D case, it is necessary to specify a node successor rule, that is, the rule for linking nodes into complete paths. Since the left border must be continuous, each parent node in the 2D graph corresponding to the left border has three successors as discussed earlier, corresponding to a left border whose distance from the centerline decreases (successor co-ordinate of $(x - 1, z + 1)$), increases (successor co-ordinate of $(x + 1, z + 1)$), or stays the same (successor co-ordinate of $(x, z + 1)$) as a function of position along the centerline. A similar statement holds for the right border. In the 3D graph, each parent node has nine successors corresponding to the possible combinations of change of positions of the left and right borders with respect to the centerline, thus forming a 3×3 successor window. With this successor rule, all paths through the 3D graph contain one and only one node from each **profile plane** in the 3D graph; that is, every path contains a single node derived from each of the left and right profile lines. This link definition ensures that region borders are continuous in the straightened image space.

Key aspects of the simultaneous approach for accurately identifying region borders are the assignment of costs to pairs of candidate borders and the identification of the optimal pair of region borders or lowest-cost path in the 3D graph. The cost function for a node in the 3D graph is derived by combining the edge costs associated with the corresponding pixels on the left and right profiles in a way that allows the position of the left border to influence the position of the right border and vice versa. This strategy resembles that employed by a human observer in situations where border positions are ambiguous. In designing the cost function, the aim is to discriminate against border pairs that are unlikely to correspond to the true region borders and to identify the border pairs that have the greatest overall probability of matching the actual borders. After the cost function is defined, either heuristic graph searching or dynamic programming methods can be used for optimal border detection.

Similarly to the 2D case, the cost of a path in the 3D graph is defined as the sum of the costs of the nodes forming the path. While many different cost functions can be designed corresponding to the general recommendations given in Section 6.2.4, the following one was found appropriate for description of border properties of a mutually inter-related border pair. Considering the cost minimization scheme, costs are assigned to nodes using the following function:

$$C_{\text{total}}(x, y, z) = \{C_s(x, y, z) + C_{pp}(x, y, z)\ \omega(x, y, z) - P_L(z) + P_R(z). \tag{7.62}$$

Each of the components of the cost function depends on the edge costs associated with image pixels. The edge costs of the left and right edge candidates located at positions x and y on profile z are inversely related to effective edge strength or other appropriate local border property descriptor $E_L(x,z)$, $E_R(y, z)$ and are given by

$$C_L(x, z) = \max_{\pi \in X,\, z \in Z} (E_L(x, z)) - E_L(x, z),$$

$$C_R^*(y, z) = \max_{\pi \in Y,\, z \in Z} (E_R(y, z)) - E_R(y, z), \tag{7.63}$$

X and Y are sets of integers ranging from 1 to the length of the left and right halves of the region profiles, and Z is the set of integers ranging from 1 to the length of the region centerline. To help avoid detection of regions adjacent to the region of interest, knowledge about the probable direction of the actual border may be incorporated into the local edge property descriptors $E_R(x, z)$, $E_R(y, z)$.

Considering the individual terms of the cost function (7.62), the term C_s is the sum of the costs for the left and right border candidates and causes the detected borders to *follow* image positions with low cost values. It is given by

$$C_s(x, y, z) = C_L(x, z) + C_R(y, z). \tag{7.64}$$

The C_{pp} term is useful in cases where one border has higher contrast (or other stronger border evidence) than the opposite border and causes the position of the low contrast border to be influenced by the position of the high-contrast border. It is given by

$$C_{pp}(x, y, z) = (C_L(x, z) - P_L(z))\ (C_R(y, z) - P_R(z)). \tag{7.65}$$

where

$$P_L(z) = \max_{\pi \in X,\, z \in Z} (E_L(x, z)) - \max_{\pi \in X} E_L(x, z),$$

$$P_R(z) = \max_{y \in Y,\, z \in Z} (E_R(y, z)) - \max_{y \in Y} E_R(y, z), \tag{7.66}$$

Combining equations (7.63), (7.65), and (7.66), the C_{pp} term can also be expressed as

$$C_{pp}(x, y, z) = \left(\max_{\pi \in X} (E_L(x, z))\, E_L(x, z)\right)\left(\max_{y \in Y} (E_R(y, z))\, E_R(y, z)\right) \tag{7.67}$$

The $\omega(x, y, z)$ component of the cost function incorporates a model of the region boundary in a way that causes the positions of the left and right borders to follow certain preferred directions relative to the model. This component has the effect of discriminating against borders that are unlikely to correspond to the actual region borders when considered as a pair. This is accomplished by including a weighting factor that depends on the direction by which a node is reached from its predecessor. For example, if the region is known to be approximately symmetric and its approximate centerline is known, the weighting factor may be given by (Figure 7.31)

$$\begin{aligned}
&\omega(x, y, z) = 1 \quad \text{for } (x, y) \in \{(\hat{x} - 1, \hat{y} - 1), (\hat{x}, \hat{y}), (\hat{x} + 1, \hat{y} + 1)\},\\
&\omega(x, y, z) = \alpha \quad \text{for } (x, y) \in \{(\hat{x} - 1, \hat{y}), (\hat{x} + 1, \hat{y}), (\hat{x}, \hat{y} - 1), (\hat{x}, \hat{y} + 1)\},\\
&\omega(x, y, z) = \beta \quad \text{for } (x, y) \in \{(\hat{x} - 1, \hat{y} + 1), (\hat{x} + 1, \hat{y} - 1)\},
\end{aligned} \tag{7.68}$$

where the node at co-ordinates (x, y, z) is the successor of the node at $(\hat{x}, \hat{y}, z - 1)$. In this case, the influence of the region model is determined by the values of α and β, typically $\alpha > \beta$. In coronary border detection applications, the values of α ranged from 1.2 to 1.8 and β from 1.4 to 2.2 [Sonka et al., 1995]. The larger the values of α and β, the stronger is the model's influence on the detected borders.

As the number of possible paths in a 3D graph is very large, the identification of the optimal path can be computationally very demanding. For example, for a 3D graph with xyz nodes, where z is the length in pixels of the region centerline, the number of possible paths is approximately 9^Z. With conventional border detection, described in Sections 6.2.4 and 6.2.5, the number of possible paths in the two two-dimensional graphs of the same size is about 3^Z. Thus, the improvement in border detection accuracy

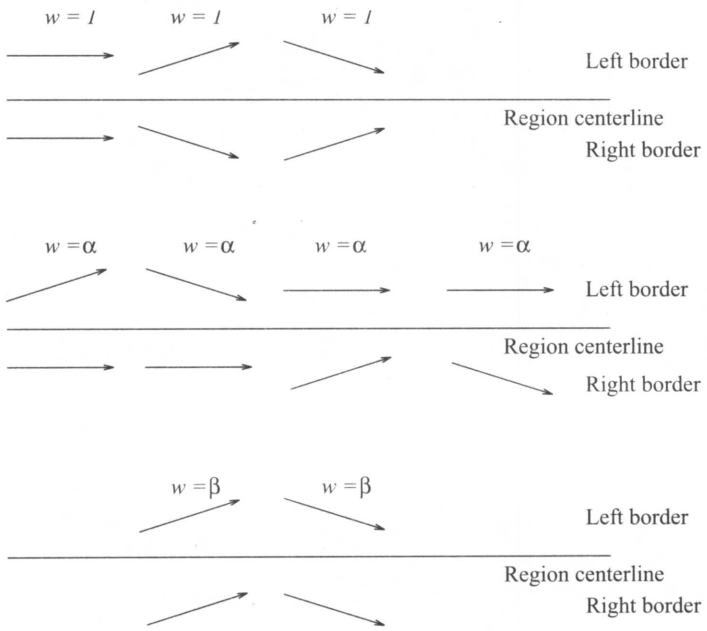


Figure 7.31: The weighting factors $\omega(x, y, z)$ associated with local directions of the potential border elements for a symmetric region model.

achieved with simultaneous border detection is accomplished at the expense of an increase in computational complexity, that is quite substantial for the heuristic graph search approach but less so for dynamic programming (Sections 6.2.4 and 6.2.5).

Improving the graph search performance is of great importance, and the $P_L(z) + P_R(z)$ term in the cost function represents the lower-bound heuristic introduced in Section 6.2.4, and does not influence the detected border; it does, however, substantially improve search efficiency if a heuristic graph searching approach is used [Sonka et al., 1993].

A second way to increase search efficiency is to use a multi-resolution approach (Section 10.1.5). First, the approximate positions of the region borders are identified in a low-resolution image; these approximate borders are used to guide the full-resolution search by limiting the portion of the full-resolution three-dimensional graph that is searched to find the precise region border positions.

To enhance border detection accuracy, a multi-stage border identification process may also be included. The goal of the first stage is to identify reliably the approximate borders of the region segment of interest while avoiding detection of other structures. Having identified the approximate border positions, the second stage is designed to localize the actual region borders accurately. In the first stage, the 3D simultaneous border detection algorithm is used to identify approximate region borders in a half-resolution image. Since this first stage is designed in part to avoid detection of structures other than the region of interest, a relatively strong region model is used. Region boundaries identified in the low-resolution image are used in the second stage to guide the search for the optimal borders in the full-resolution cost image, as described in the previous paragraph. A somewhat weaker region model may be used in the second stage to allow more influence from the image data (Section 10.1.5). Further details about the cost function design can be found in [Sonka et al., 1993, 1995].

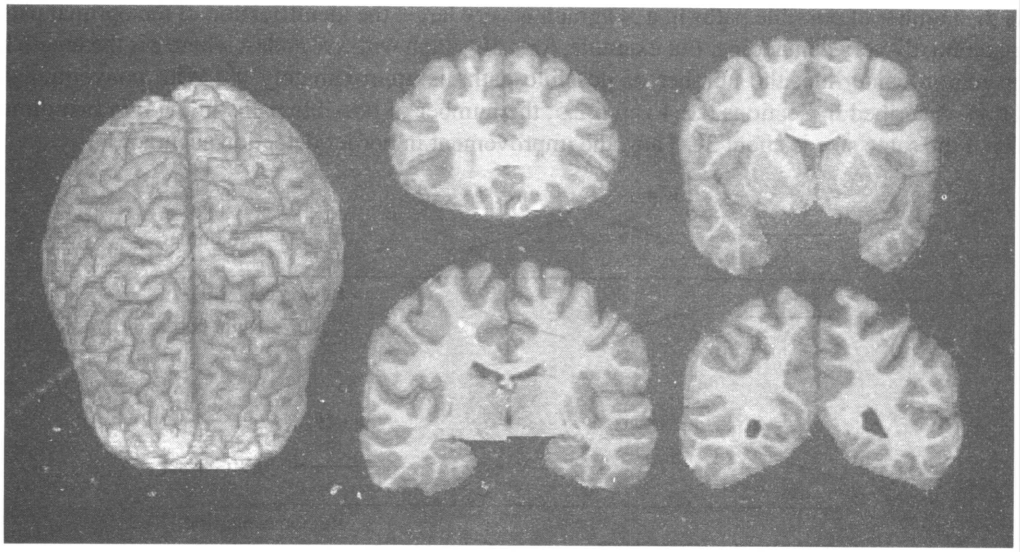


Figure 7.32: Magnetic resonance images of human brain. Left: Three-dimensional surface rendering of original MR image data after segmentation of the brain from the skull. Right: Four of 120 two-dimensional slices that form the three-dimensional image volume. *Courtesy of R. J. Frank and H. Damasio, The University of Iowa.*

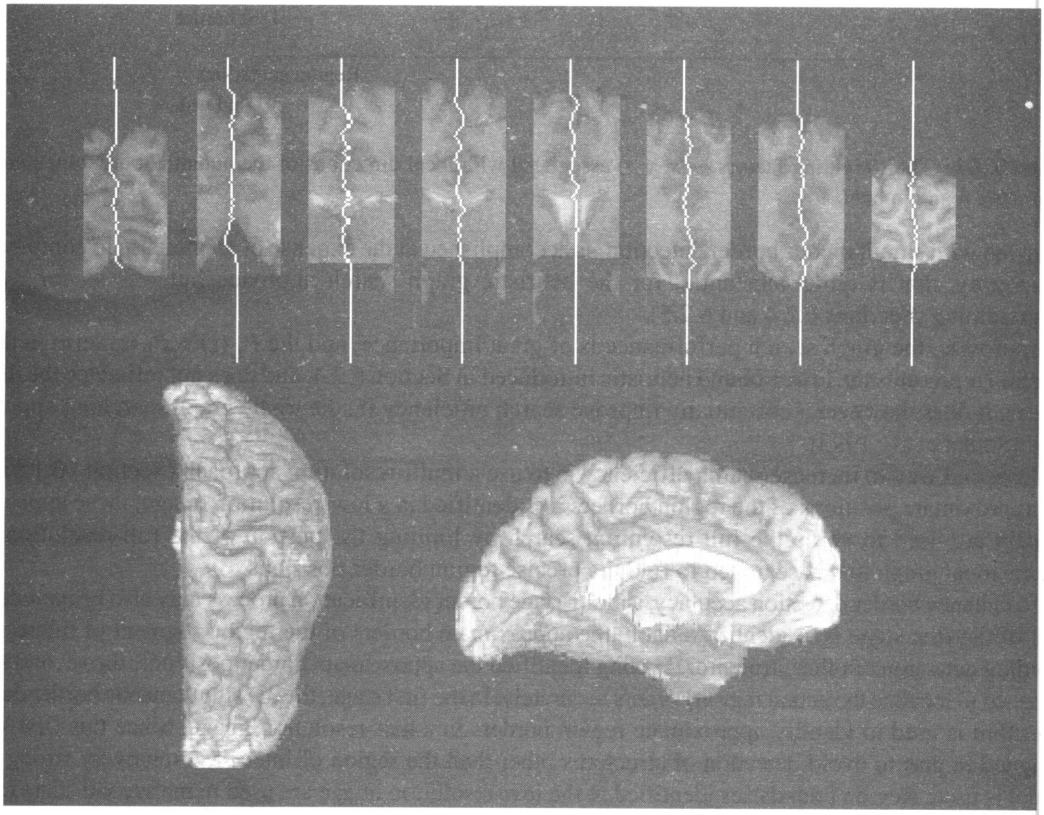


Figure 7.33: Surface detection. Top: Borders between the left and right hemispheres forming the 3D surface are shown in eight of 120 individual slices. Bottom: After the segmentation in the left and right hemispheres, the internal cortex surfaces may be visualized. *Courtesy of R. J. Frank and H. Damasio, The University of Iowa.*

7.5.2 Sub-optimal surface detection

If three-dimensional volumetric data are available, the task may be to identify three-dimensional surfaces representing object boundaries in the three-dimensional space. This task is common in segmentation of volumetric medical image data sets from magnetic resonance, X-ray, ultrasound, or other tomographic scanners, which produce 3D volumes consisting of stacked 2D image slices. Usually, the 2D images are more or less independently analyzed and the 2D results stacked to form the final 3D segmentation. It is intuitively obvious that a set of 2D borders that were detected in individual slices may be far from optimal if the entire 3D volume is considered, and concurrent analysis of the entire 3D volume may give better results if a globally optimal surface is determined (see Section 7.7). Consider an example of brain cortex visualization from three-dimensional magnetic resonance (MR) data sets of a human brain (Figure 7.32). Note that the internal cortex surfaces are not directly visible unless the brain is segmented into the right and left hemispheres. An example of such brain segmentation applied to an individual MR slice was given earlier in Figure 6.36. If the 3D case is considered, the goal is to identify the 3D surface that optimally divides the brain (Figure 7.33).

It is necessary to define a criterion of optimality for the surface. Since it must be contiguous in 3D space, it will consist of a mesh of 3D connected voxels. Consider a 3D graph that corresponds in size with the 3D image data volume; the graph nodes correspond to image voxels. If a cost is associated with each graph node, the optimal surface can be defined as that with the minimum total cost of all *legal* surfaces that can be defined in the 3D volume. The legality of the surface is defined by the 3D surface connectivity requirements that depend on the application at hand, and the total cost associated with a surface can be calculated as the sum of individual costs of all nodes forming the surface. Therefore, it should be possible to determine the optimal surface by application of optimal graph searching principles similar to those presented in Sections 6.2.4 and 6.2.5. Unfortunately, standard graph searching approaches cannot be directly extended from a search for a path to a search for a **surface** [Thedens et al., 1995]. Generally, two distinct approaches can be developed to overcome this problem. New graph searching algorithms may be designed to search directly for a surface, or a surface detection task may be represented in a way that permits conventional graph searching algorithms to be used.

Compared to the search for an optimal path through a graph (even through a 3D graph as shown in Section 7.5.1), the search for an optimal surface results in combinatorial explosion of the task's complexity, and the absence of an efficient searching algorithm has represented a limiting factor on 3D surface detection. One approach to optimal surface detection based on cost minimization in a graph was given in [Thedens et al., 1990, 1995]. The method used standard graph searching principles applied to a transformed graph in which standard graph searching for a *path* was used to define a *surface*. While the method guaranteed surface optimality, it was impractical due to its enormous computational requirements. The same authors developed a heuristic approach to surface detection that was computationally feasible [Thedens et al., 1995].

Using several ideas from [Thedens et al., 1995], a sub-optimal approach to direct detection of surfaces was introduced in [Frank, 1996; Frank et al., 1996]. This approach is based on dynamic programming and avoids the problem of combinatorial explosion by introducing local conditions that must be satisfied by all legal surfaces. The paradigm is called **surface growing**. The graph size corresponds directly to the image size, and due to the local character of surface growing, the graph construction is straightforward and orderly. The entire approach is simple, elegant, computationally efficient, and fast. Additionally, it can be generalized to searching higher-dimensional spaces, e.g., time-variant three-dimensional surfaces. While the resulting surfaces typically represent good solutions, surface optimality is not guaranteed.

The sub-optimal three-dimensional graph searching method was applied to brain cortex segmentation shown in Figures 7.32 and 7.33. The cost function was based on inverted gray-level values of the image voxels after the ventricles were three-dimensionally filled not to represent a large low-cost region.

7.6 GRAPH CUT SEGMENTATION

The *direct* use of minimum cut/maximum flow combinatorial optimization algorithms in image processing was first reported in [Greig et al., 1989], where the approach was employed for binary image reconstruction. Using the same family of graph optimization algorithms, a powerful technique for optimal boundary and region segmentation in n-D image data was presented in [Boykov and Jolly, 2001; Boykov and Kolmogorov, 2001; Boykov and Funka-Lea, 2006]. The method is initiated by interactive or automated identification of one or more points representing the 'object' and one or more points representing the 'background'–these points are called **seeds** and serve as segmentation **hard constraints**. Additional **soft constraints** reflect boundary and/or region information. As with other optimal graph searching techniques, the segmentation solution is globally optimal with respect to an objective function. The general version of the cost function C calculated on image segmentation f follows the **Gibbs model** [Geman and Geman, 1984] (compare this with cost functions discussed in Section 10.8)

$$C(f) = C_{\text{data}}(f) + C_{\text{smooth}}(f). \tag{7.69}$$

To minimize $C(f)$, a special class of arc-weighted graphs $G_{st} = (V \cup \{s, t\}, E)$ is employed. In addition to the set of nodes V corresponding to pixels (voxels) of the image I, the node set of G_{st} contains two special *terminal* nodes, namely the *source* s and the *sink* t. These terminals are hard-linked with the segmentation seed points (bold links in Figure 7.34) and represent the segmentation labels (object, background).

The arcs E in G_{st} can be classified into two categories: *n-links* and *t-links*. The n-links connect pairs of neighboring pixels whose costs are derived from the smoothness term $C_{\text{smooth}}(f)$. The t-links connect pixels and terminals with costs derived from the data term $C_{\text{data}}(f)$. An $s-t$ cut in G_{st} is a set of arcs whose removal partitions the nodes into two disjoint subsets S and T, such that $s \in S$ (all nodes linked to source) and $t \in T$ (all nodes linked to sink) and no directed path can be established from s to t. The cost of a cut is the total cost of arcs in the cut, and a minimum $s-t$ cut is a cut whose cost is minimal. The **minimum $s-t$ cut** problem and its dual, the **maximum flow** problem, are classic combinatorial problems that can be solved by various polynomial-time algorithms [Ford and Fulkerson, 1956; Goldberg and Tarjan, 1988; Goldberg and Rao, 1998]. Figure 7.34 shows a simple example of the use of graph cut for segmentation.

Let O, B be sets of image pixels corresponding to the object and background seeds, respectively; $O \subset V$, $B \subset V$, $O \cap B = \emptyset$. The seeds are used to form hard t-links in the graph. Then, the graph cut shall be determined to form the object(s) and background from the image pixels in a way that all object pixels are connected to the object seed terminal and all background pixels to the background seed terminal. This is accomplished by searching for a graph cut that minimizes a cost function (equation 7.69), the terms of which are a weighted combination of regional and boundary properties of the object with respect to the background.

Let the set of all image pixels be denoted by I, and let N denote a set of all directed pairs of pixels (p, q), $p, q \in I$ representing neighborhood pixel relationships. For example, 2D image pixels form a rectangular 2D grid with 4- or 8-neighborhood connectivity links contained in N. In the 3D case, image voxels form a three-dimensional grid and all their pairwise neighborhood relationships (e.g., reflecting 26-connectivity) are contained in N. This concept can be directly extended to n – D. A cost of (p, q) may differ from that of (q, p) allowing incorporation of asymmetric neighborhood relationships.

Let each image pixel i_k take a binary label $L_k \in \{obj, bgd\}$ where *obj* and *bgd* represent the object and background labels, respectively. The labeling vector $\mathbf{L} = (L_1, L_2, \ldots, L|I|)$ defines the resulting binary segmentation. The cost function C that is minimized to achieve optimal labeling may be defined as a λ-weighted combination of a regional property term $R(\mathbf{L})$ and a boundary property term $B(\boldsymbol{L})$ [Greig et al., 1989; Boykov and Jolly, 2001] (compare equation (7.69))

$$C(\mathbf{L}) = \lambda R(\mathbf{L}) + B(\mathbf{L}), \tag{7.70}$$

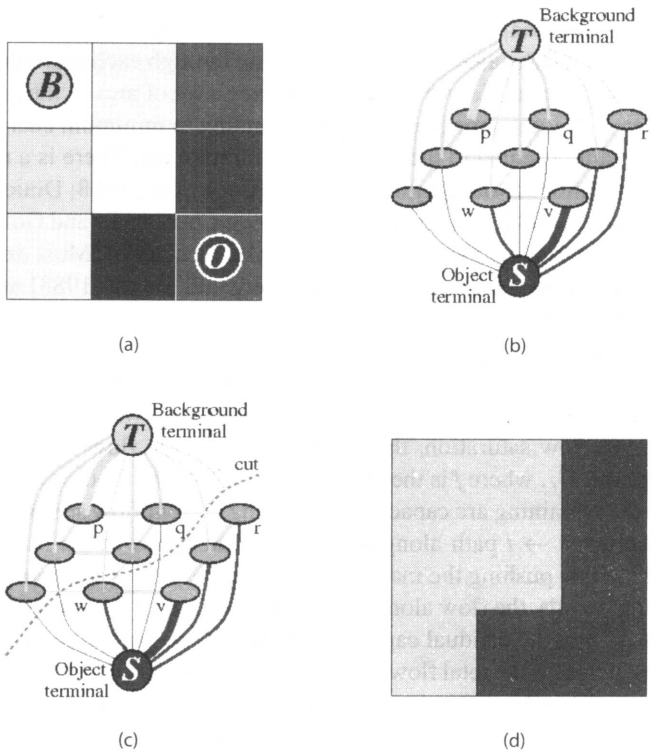


Figure 7.34: Graph cut segmentation-simple segmentation example. (a) Image with seeds - seed *B* corresponding to background and seed *O* to object. (b) Graph. (c) Graph cut. (d) Segmentation result. *Courtesy of Y. Boykov, University of Western Ontario, ©2001 IEEE [Boykov and Jolly, 2001].*

where

$$R(\mathbf{L}) = \sum_{p \in I} R_p\left(L_p\right) \tag{7.71}$$

$$B(\mathbf{L}) = \sum_{(p,q) \in N} B_{(p,q)}\, \delta\left(L_p, L_q\right), \tag{7.72}$$

and

$$\delta\left(L_p, L_q\right) = \begin{cases} 1 & \text{if } L_p \neq L_p, \\ 0 & \text{otherwise.} \end{cases}$$

Here, $R_p(obj)$ may be understood as a pixel-specific cost associated with labeling pixel p as *object* and $R_p(bgd)$ a cost of labeling the same pixel p as *background.* For example, expecting bright objects on a dark background, the cost $R_p(obj)$ will be large in dark pixels (low I_p values) and small in bright pixels. Similarly, $B_{(p,q)}$ is a cost associated with a local labeling discontinuity between neighboring pixels p, q. $B_{(p,q)}$ should be large for both p and q belonging to either object or background, and small if one of p, q belongs to object and the other to background, i.e., across object/background boundaries. Thus, $B_{(p,q)}$ may correspond, e.g., to the inverted image gradient magnitude between pixels p and q [Mortensen and Barrett, 1998]. As described above, the complete graph includes n-links and t-links. The weights of the individual graph arcs are assigned to the graph according to Table 7.1. The minimum cost cut on the graph G can be computed in polynomial time for two-terminal graph cuts assuming the are weights are non-negative [Ford and Fulkerson, 1962].

The minimum $s-t$ cut problem can be solved by finding a maximum flow from the source s to the sink t. In maximum flow algorithms, the 'maximum amount of water' flowing from the source to the sink is sent through the directed graph arcs and the amount of water flowing through each individual arc is specified by its capacity–or arc cost. The maximum flow from s to t saturates a set of arcs in the graph. These saturated arcs divide the nodes into two disjoint parts S and T, corresponding to minimum cuts [Ford and Fulkerson, 1962]. The maximum flow value is equal to the cost of the minimum cut. There is a number of algorithms that can be used to solve this combinatorial optimization task [Cook et al., 1998; Dinic, 1970; Edmonds and Karp, 1972; Goldberg and Tarjan, 1988; Goldberg and Rao, 1998; Cherkassky and Goldberg, 1997; Corman et al., 1990; Boykov and Kolmogorov, 2004; Boykov and Funka-Lea, 2006]. Most existing algorithms can be categorized in two groups–**push-relabel** methods [Goldberg and Tarjan, 1988] and **augmenting path** methods [Ford and Fulkerson, 1962]; a comparison of major graph cut algorithms with applications in vision can be found in [Boykov and Kolmogorov, 2004].

Augmenting path algorithms (e.g., [Dinic, 1970]) push the flow through the graph from s to t until the maximum flow is reached. The process is initialized with zero flow status when no flow exists between s and t. During the steps leading to flow saturation, the current status of the flow distribution is continuously maintained in a **residual graph** G_f, where f is the current flow. While the topology of G_f is identical to that of G_{st}, the arc values keep the remaining arc capacity considering current flow status. At each iteration step, the algorithm finds the shortest $s \rightarrow t$ path along the non-saturated arcs of the residual graph. The flow through this path is augmented by pushing the maximum possible flow so that at least one of the arcs along this path is saturated. In other words, the flow along the path is increased by Δf, the residual capacities of the path arcs are decreased by Δf, and the residual capacities of the reverse path arcs are increased by Δf. Each of these augmentation steps increases the total flow from the source to sink. Once the flow cannot be increased any more (so no new $s \rightarrow t$ path can be defined consisting exclusively of non-saturated arcs) the maximum flow is reached and the optimization process terminates. The separation of the S and T graph nodes defining the segmentation-the minimum $s-t$ cut–is defined by the saturated graph arcs.

Table 7.1: Cost terms for Graph Cut segmentation. K may be interpreted as the maximum needed flow capacity of the arc from source s to $p \in O$ (or from $p \in B$ to sink t), increased by one so that the arc gets never saturated; $K = 1 + \max_{p \in I} \sum_{q:(p,q) \in N} B_{(p,q)}$.

Graph arc	*Cost*	
(p, q)	$B_{(p,q)}$	for $(p, q) \in N$
(s, p)	$\lambda R_p(bgd)$	for $p \in I, p \notin (O \cup B)$
	K	for $p \in O$
	0	for $p \in B$
(p, t)	$\lambda R_p(obj)$	for $p \in I, p \notin (O \cup B)$
	0	for $p \in O$
	K	for $p \in B$

The algorithm given in [Dinic, 1970] identifies the shortest path from s to t using a breadth-first search. Once all paths of length k are saturated, the algorithm starts with exploring $s \rightarrow t$ paths of lengths $k+1$. The algorithm's complexity is $\mathcal{O}(mn^2)$, where n is the number of nodes and m is the number of arcs in the graph. Figure 7.35 gives an example of the steps needed for determining the minimum cut using an augmenting path maximum flow algorithm.

Push-relabel algorithms for maximum flow optimization [Goldberg and Tarjan, 1988] maintain a labeling of nodes with a lower bound estimate of its distance to the sink node along the shortest non-saturated path. The algorithm's functionality attempts to 'push' excess flow towards the nodes with shorter estimated

distances to the sink. In each step, the node with the largest distance label is subjected to the push operation. Alternatively perhaps, a first-in-first-out strategy may be used. The distance estimate labels increase as more and more arcs are saturated after the push operations. As excessive flow may be pushed onto a node, this is eventually drained back to the source. Details of this approach can be found in [Cook et al., 1998].

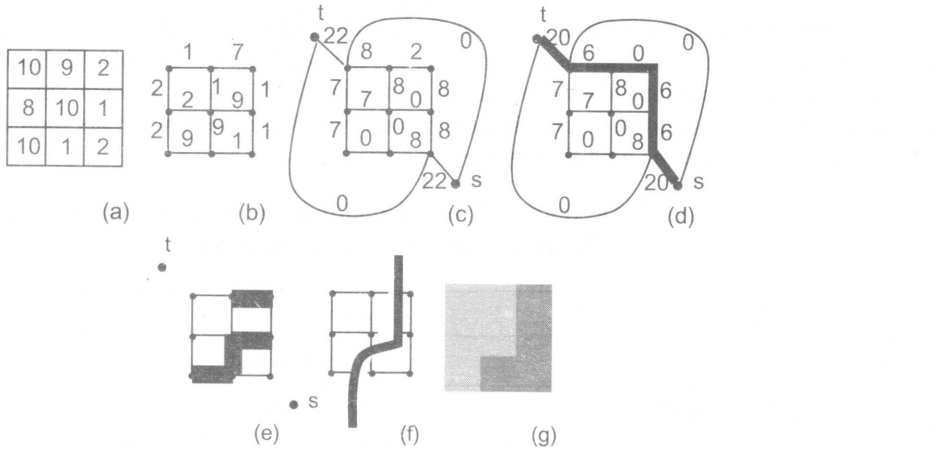


Figure 7.35: Image segmentation using graph cuts and maximum flow optimization. (a) Original image data-corresponding to Figure 7.34a. (b) Edge magnitudes calculated as image intensity differences in 4-connectivity. (c) G_{st} graph constructed according to Table 7.1; $\lambda = 0$; n-link costs calculated as in equation (6.18); reverse path residual capacities are not shown. (d) Residual graph G_f after the one and only shortest path with non-saturated $s \rightarrow t$ connection was identified and saturated. No new non-saturated $s \rightarrow t$ path can be found. (e) Saturated graph arcs identified by thick black lines. (f) Resulting minimum $s - t$ cut separating S and T nodes. (g) Corresponding image segmentation.

Recall that the goal of graph-cut segmentation is to minimize the objective function given in equation (7.70) subject to the requirement of labeling all seeds according to the initial hard constraints. Algorithm 7.7 describes this optimization process.

Algorithm 7.7: Graph cut segmentation

1. Create an arc-weighted directed graph corresponding in size and dimensionality to the image to be segmented.
2. Identify object and background seeds-example points required to be part of the background or object(s) in the final segmentation. Create two special graph nodes-source s and sink t; connect all seeds with either the source or the sink node based on their object or background label.
3. Associate appropriate arc cost with each link of the formed graph according to Table 7.1.
4. Use one of the available maximum flow graph optimization algorithms to determine the graph cut.
5. The minimum $s - t$ cut solution identifies the graph nodes that correspond to the image boundaries separating the object(s) and the background.

An important feature of this approach is its ability to interactively improve a previously obtained segmentation in an efficient way. Assume that the user has identified the initial seeds, the cost function is available, and the graph cut optimization yielded a segmentation that was not as good as required. The segmentation can be improved by adding supplemental object or background seeds. Suppose the user adds a new object seed-while it is possible to recompute the graph cut segmentation from scratch, an efficient way does not require restarting. Rather, the previous status of the graph optimization can be used to initialize the next graph cut optimization process.

Let a maximum flow algorithm be used for identifying the optimal graph $s - t$ cut. In this case, the algorithmic solution is characterized by saturation of the graph by maximum flow. Adding a new object seed p requires forming corresponding hard t-links according to Table 7.1: weight of (s, p) set to K and weight (p, t) set to 0. The latter may lead to appearance of negative capacities in the residual network of the current flow. This is easily compensated for by increasing values c_p of t-links as specified in Table 7.2. The new costs are consistent with the costs of pixels in O since the additional constant c_p appears at both t-links and thus does not change the optimal cut. Therefore, the new optimal cut can be efficiently obtained starting from the previous flow solution without starting from scratch. Of course, the same approach can be used if a new background seed is added. Again, the cost constants added to the new t-links should be consistent with the cost table and need to be modified by the same constant.

Table 7.2: Cost term $c_p = \lambda\,(R_p\,(bgd) + R_p\,(obj))$ modification for sequential improvement of graph cut segmentation after adding object seed p.

t-link	*initial cost*	*added cost*	*new cost*
(s,p)	$\lambda R_p\,(bgd)$	$K + \lambda Rp(obj)$	$K + c_p$
(p,t)	$\lambda R_p\,(obj)$	$\lambda R_p\,(bgd)$	c_p

As is always the case with optimization techniques, cost function design influences the method's performance in real-world applications. For example, the seeds identifying the object and background exemplars may consist of small patches and may thus be used to sample the object and background image properties, e.g., calculating histograms of object and background patches. Let $P\,(I|O)$ and $P\,(I|B)$ represent probabilities of a particular gray level belonging to object or background, respectively. These probabilities can be derived from the patch histograms. (It will be obvious that more complex probability functions can be used instead). Then, the regional R_p and boundary $B(p, q)$ costs can be determined as [Boykov and Jolly, 2001]

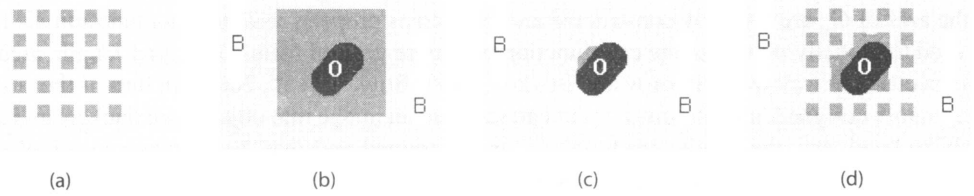
$$R_p\,(obj) = -\ln P\,(I_p|O),$$

$$R_p\,(bgd) = -\ln P\,(I_p|B),$$

$$B\,(p, q) = \exp\left(-\frac{\left(I_p - I_q\right)^2}{2\sigma^2}\right)\frac{1}{\|p, q\|}, \tag{7.73}$$

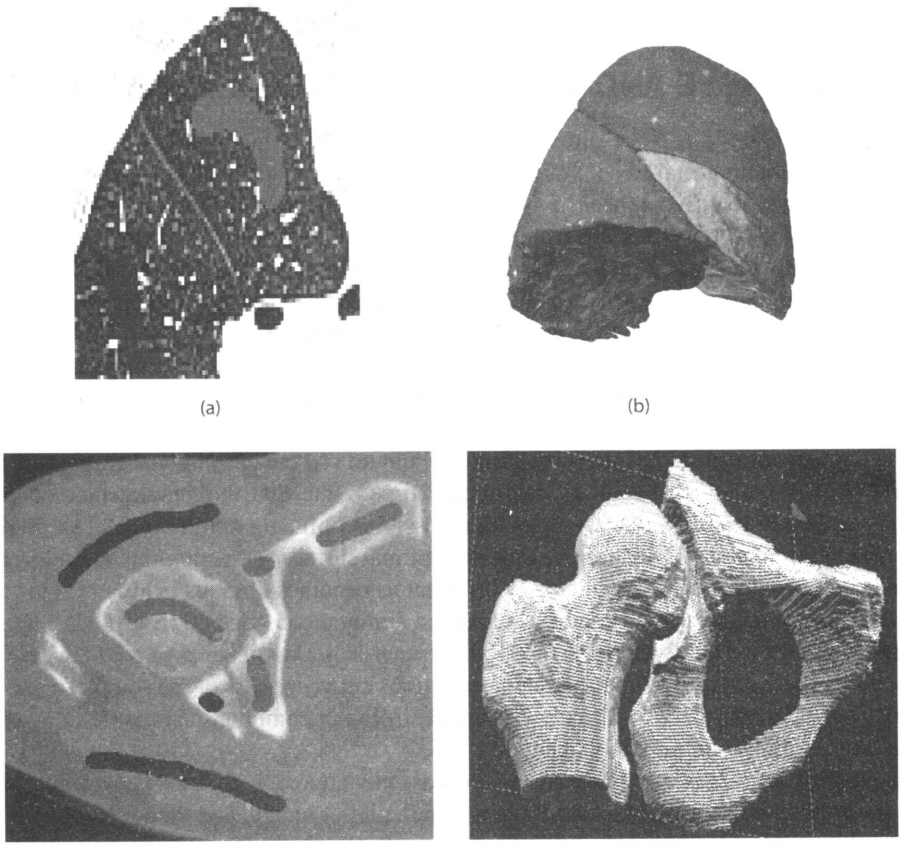
where $\|p, q\|$ denotes distance between pixels p, q. Thus, $B\,(p, q)$ is high for small differences between image values $|I_p - I_q| < \sigma$ (within object or background). Cost $B\,(p, q)$ is low for boundary locations where $|I_p - I_q| > \sigma$. Here, σ represents allowed or expected intensity variation within the object and/or background.

Using the cost functions given in equations (7.73), Figure 7.36 demonstrates the method's behavior and the role of the weighting coefficient λ in equation (7.70). Graph cut applications range from stereo through multi-view image stitching, video texture synthesis, or image reconstruction, to n-dimensional image segmentation. Figure 7.37 demonstrates the capability of the method to segment lung lobes from X-ray computed tomography data.



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

Figure 7.36: Graph cut segmentation behavior on a synthetic image. In all cases, the segmentation was initialized using the object patch as marked in black and background patch marked in white. The resulting segmentation is shown in light gray (background) and dark gray (objects). The initialization patches are parts of the segmented object (s) or background. (a) Original image. (b) Segmentation result for $\lambda \in [7, 43]$, i.e., only using a wide weighting range of region and boundary cost terms. (c) Segmentation result for $\lambda = 0$, i.e., only using the boundary cost term. (d) Segmentation result for $\lambda = 60$, i.e., using almost solely the region cost term. Notice the ability of the method to change the topology of the segmentation result. *Courtesy of Y. Boykov, University of Western Ontario, ©2001 IEEE [Boykov and Jolly, 2001].*



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

Figure 7.37: Graph cut segmentation in a 3D X-ray computed tomography image of human lungs and bones. (a) Original 3D image data with lung lobe and background initialization shown in two shades of gray–the segmentation works in a full 3D volumetric dataset. (b) Resulting lung lobe segmentation. (c) Bone and background initialization. (d) Resulting 3D segmentation. *Courtesy of Y. Boykov, University of Western Ontario and B. Geiger, Siemens Research. A color version of this figure may be seen in the color inset—Plate 12.*

Boykov and Jolly's method is flexible and shares some of the elegance of level set methods. It was proven that if the arcs of G_{st} are properly constructed and their costs properly assigned, a minimum $s-t$ cut in G_{st} can be used to globally minimize the cost function of a more general form combining length, area, and flux terms in an efficient way [Kolmogorov and Boykov, 2005; Boykov et al., 2006]. In turn, if the cost function is appropriately designed, a minimum $s-t$ cut can segment an image into objects and background as desired. Similarly to level sets, the results are topology-unconstrained and may be sensitive to initial seed point selections unless a priori shape knowledge about the objects is incorporated. While the graph cut approach provides an inherently binary segmentation, it can be extended to multi-label segmentation problems as described in [Boykov and Veksler, 2006]. Unfortunately the multi-way cut problem is NP-complete and an á-expansion algorithm may be used to obtain a good approximate solution [Boykov et al., 2001]. The development of graph-cut image segmentation methods is ongoing [Boykov and Funka-Lea, 2006]. A combination of graph cuts and geodesic active contours is reported in [Boykov and Kolmogorov, 2003]. A connection between discrete graph cut algorithms and global optimization of a wide class of continuous surface functionals can be found in [Kolmogorov and Boykov, 2005]. In-depth discussion of associations between the level set and graph cut approaches can be found in [Boykov and Kolmogorov, 2003; Boykov and Funka-Lea, 2006; Boykov et al., 2006]. Experimental comparison of performance of several min-cut / max-flow algorithms for energy minimization in vision applications can be found in [Boykov et al., 2001].

7.7 OPTIMAL SINGLE AND MULTIPLE SURFACE SEGMENTATION

The task of optimally identifying three-dimensional surfaces representing object boundaries is important in segmentation and quantitative analysis of volumetric images. In addition to single standalone surfaces, many surfaces that need to be identified appear in mutual interactions. These surfaces are **coupled** in a way that their topology and relative positions are usually known, and they appear in some specific relationship. Clearly, incorporating such surface-interrelation information into the segmentation will further improve its accuracy and robustness. Simultaneous segmentation of coupled surfaces in volumetric images is an under-explored topic, especially when more than two surfaces are involved.

A polynomial time method was developed for n-D ($n \geq 3$) optimal hyper-surface detection with hard smoothness constraints, making globally optimal surface segmentation in volumetric images practical [Wu and Chen, 2002; Li et al., 2004b]. By modeling the problem with a weighted *geometric graph*, the method transforms the segmentation problem into computing a minimum $s-t$ cut in a directed graph, which simplifies the problem and consequently solves it in polynomial time. Note that the general method of graph cut optimization is again employed, which accounts for a possibly confusing terminological similarity between the direct graph cut segmentation (Section 7.6) and the optimal surface segmentation methods reported here. Nevertheless, the two approaches are principally different as becomes obvious below.

The optimal surface segmentation method facilitates simultaneous detection of k ($k \geq 2$) interrelated surfaces by modeling the n-D problem in an $(n+1)$-D geometric graph (or simply *graph*), where the $(n+1)$-th dimension holds special arcs that control the interrelations between pairs of the sought surfaces [Li et al., 2004a, 2006]. The apparently daunting combinatorial explosion in computation is avoided by transforming the problems into computing minimum s - t cuts.

Like other graph-search based segmentation methods, this approach first builds a graph that contains information about the boundaries of the target objects in the input image, and then searches the graph for a segmentation solution. However, to make this approach work effectively for segmentation problems, several key issues must be handled: (i) How to obtain relevant information about the target object boundaries; (ii) how to capture such information in a graph; and (iii) how to search the graph for the *optimal* surfaces of the target objects. The general approach consists of five main steps, which constitute a high level solution to these three key issues. Of course, in solving different segmentation problems, variations of these steps may be applied.

Algorithm 7.8: Optimal surface segmentation

1. *Pre-segmentation.* Given an input image, perform a pre-segmentation to obtain an approximation to the (unknown) surfaces for the target object boundaries. This gives useful information on the topological structures of the target object(s). Quite a few approximate surface detection methods are available, such as active appearance models, level sets, and atlas-based registration. For surfaces with a geometry that is known to be relatively *simple* and thus allows the *unfolding* process (e.g., terrain-like, cylindrical, tubular, or spherical surfaces), this first step may not be needed.
2. *Mesh Generation.* From the resulting approximate surface(s), a mesh is computed. The mesh is used to specify the structure of a graph G_B, called the *base graph*. G_B defines the neighboring relations among voxels on the sought (optimal) surfaces. Voronoi diagram and Delaunay triangulation algorithms or isosurfacing methods (e.g., the marching cubes) can be used for the mesh generation. For surfaces allowing an *unfolding* operation, this step may not be needed, since in many cases a mesh can be obtained easily.
3. *Image Resampling.* For each voxel v on the sought surfaces, a vector of voxels is created that is expected to contain v. This is done by resampling the input image along a ray intersecting every vertex u of the mesh (one ray per mesh vertex). The direction of the ray is either an approximate normal of the meshed surface at u, or is defined by a center point/line of the target object. These voxel vectors produced by the resampling form a new image. Steps 1–3 are for handling issue (i) above.
4. *Graph Construction.* A weighted directed graph G is built on the vectors of voxels in the image that resulted from the resampling. Each voxel vector corresponds to a list of nodes in G (called a *column*). G is a *geometric* graph since it is naturally embedded in an n-D space ($n = 3$). The neighboring relations among voxels on the sought surfaces are represented by the adjacency relations among the columns of G, as specified by the arcs in the base graph G_B. Each column contains exactly one voxel located on the sought surfaces. The arcs of G are used to enforce constraints on the sought surfaces, such as the smoothness constraints and inter-surface separation constraints. The intensity of each voxel in the vectors is related to the cost of the corresponding node in G. The node costs of G can also encode edge-based and region-based cost functions. Information on the constraints and cost functions of a target segmentation problem needs to be obtained. This step is for handling issue (ii).
5. *Graph Search.* The graph construction scheme ensures that the sought optimal surfaces correspond to an *optimal closed set* in the weighted directed graph G (as proven in [Wu and Chen, 2002; Li et al., 2006]). Thus, the sought optimal surfaces are obtained by searching for an optimal closed set in G using efficient closed set algorithms in graph theory and can be achieved by using standard $s-t$ cut algorithms. This step is for handling the issue (iii).

Simple example The formal description of the graph searching algorithms given here is precise but not very intuitive. To reach an intuitive understanding of the underlying processes before the formal description, a very simple 2D example is presented corresponding to a tiny 2×4 image. Let graph nodes correspond to image pixels with a cost associated with each node (Figure 7.38a). The goal is to find the *minimum-cost path* from left to right. The cost of the path is calculated as the sum of its node costs. The list of all paths in the graph includes (considering that the maximum allowed vertical distance between the two next-column nodes of the path is 1): *ae*, *af*, *be*, *bf*, *bg*, *cf*, *cg*, *ch*, *dg* and *dh*. The minimum-cost path can be easily identified as *cg* with the cost of 2.

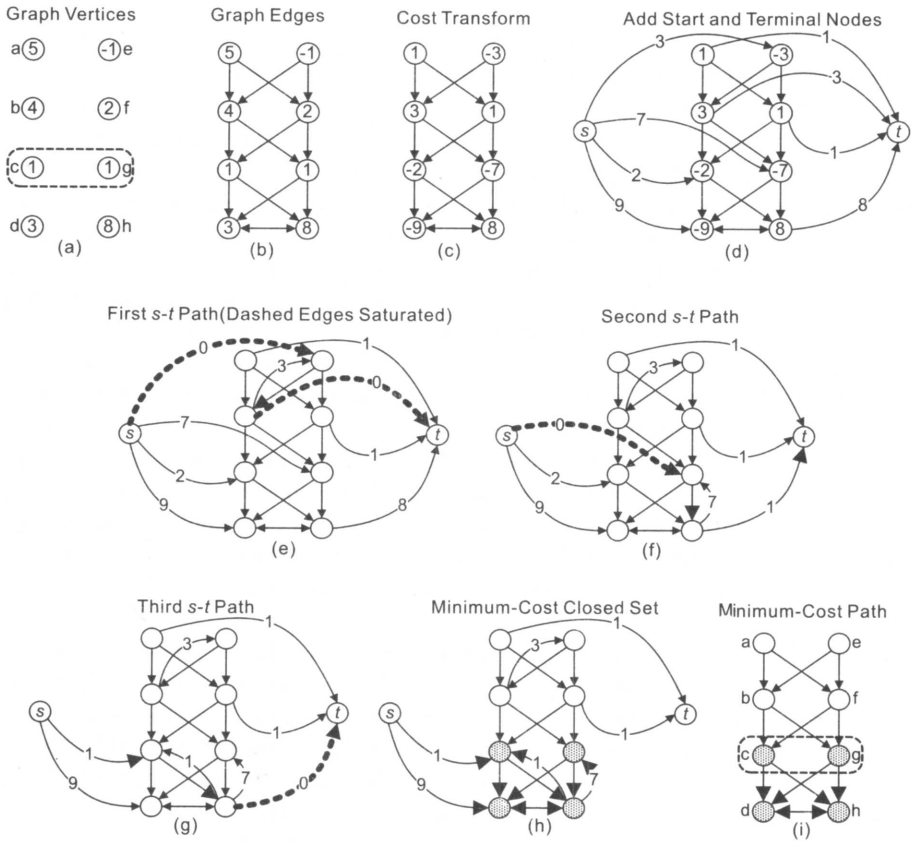


Figure 7.38: Simple 2D example of the proposed minimum cost graph-search detection algorithm. See text for details.

The arcs of the graph are constructed as shown in Figure 7.38b. Cost transformation is performed by subtracting the cost of the node immediately below from the cost of the node under consideration (Figure 7.38c). The costs of the bottommost two nodes are left unchanged, unless their cost sum is greater or equal to 0. If so, the sum of the bottom node costs is increased by one and subtracted from any single one of the bottommost nodes. In this example, the sum of the bottommost two nodes is 11; let's select node d and subtract 12 from its cost (Figure 7.38c). A closed set is a subset of the graph nodes with no arcs leaving the set. Every potential path in Figure 7.38a uniquely corresponds to a closed set in Figure 7.38c. Importantly, the *minimum-cost path* corresponds to the *minimum-cost closed set* in Figure 7.38c.

To compute the minimum-cost closed set of the graph, a transform to an arc-weighted directed graph is performed. Two new auxiliary nodes are added to the graph-a start node s with a connecting arc to every negative-cost node, and a terminal node t with an arc from every non-negative-cost node. Every arc is assigned a capacity. The capacities of the arcs from (to) the start (terminal) node are the absolute values of the costs of the nodes they are connected to (from) (Figure 7.38d). All other arcs have infinite capacity. The node costs are no longer used and are ignored. The minimum-cost closed set of the graph in Figure 7.38c can be obtained by computing the *minimum* $s - t$ *cut* or *maximum flow* from s to t in the graph.

The graph transforms described above represent the core of this approach. To solve the next step, several algorithms for computing the minimum $s - t$ cut exist as outlined in the previous section. Following the maximum flow optimization approach (Section 7.6), the negative-cost (non-negative-cost) nodes are tunnels allowing water to flow in (out). The arcs are pipes connecting the source, tunnels and the sink. The pipes are directional, and the cumulative water flow cannot exceed the pipe capacity. Due to the limited pipe capacities,

the amount of water that can flow from the source to the sink will have some maximum. To achieve this maximum flow, some pipes will be saturated, meaning that the water flowing through them will equal their capacities. In Figure 7.38e, the path from s to t was found with a capacity of 3. This will saturate the path's pipes (arcs) from the source s and to the sink t. These two saturated pipes are removed and a new pipe is created in the reverse direction along the path having a capacity of 3. In Figure 7.38f, another $s - t$ path is found with a capacity 7. Similarly, a reverse path with capacity 7 is created. Figure 7.38g identifies the third and final path that can be found-its capacity of 1 saturates the pipe to the sink that was not completely saturated in the previous step. Since this was the last path, all tunnels that can be reached from the source are identified (Figure 7.38h) as belonging to the minimum cost closed set (Figure 7.38i). The uppermost nodes of the minimum closed set form the minimum cost path thus determining the solution.

Graph construction A key innovation of this method is its non-trivial graph construction, aiming to transform the surface segmentation problem into computing a minimum *closed set* in a node-weighted directed graph. A closed set Z in a digraph is a subset of nodes such that all successors of any nodes in Z are also contained in Z. The *cost* of a closed set is the total cost of the nodes in the set. The minimum closed set problem is to search for a closed set with the minimum cost, which can be solved in polynomial time by computing a minimum $s - t$ cut in a derived arc-weighted digraph [Hochbaum, 2001].

Single surface graph construction A volumetric image can be viewed as a 3-D matrix I(x, y, z) (Fig. 7.39). Without loss of generality, a *surface* in I is considered to be terrain-like and oriented as shown in Figure 7.40. Let X, Y and Z denote the image sizes in x, y and z directions, respectively. We utilize a *multi-column* modeling technique. A surface is defined by a function $S : (x, y) \rightarrow S(x, y)$, where $x \in \mathrm{x} = \{0,..., X - 1\}$, $y \in y = \{0,..., Y - 1\}$ and $S(x, y) \in \mathrm{z} = \{0,..., Z - 1\}$. Thus, any surface in I intersects with exactly one voxel of each *column* (of voxels) parallel to the z-axis, and it consists of exactly $X \times Y$ voxels.

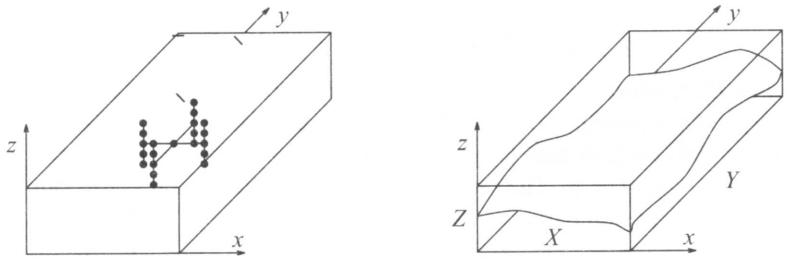


Figure 7.39: Graph construction. (a) Graph node neighbors-considering the smoothness constraint $\Delta_x = \Delta_y = 2$). (b) 3D graph XYZ and the 3D surface dividing the graph into upper and lower parts.

A surface is regarded as *feasible* if it satisfies some application-specific *smoothness constraint,* defined by two smoothness parameters, Δ_x and Δ_y. The smoothness constraint guarantees surface connectivity in 3-D. More precisely, if $I(x, y, z)$ and $I(x + 1, y, z')$ are two voxels on a feasible surface, then $|z - z'| \leq \Delta_x$. Likewise, *if* $I(x, y, z)$ and $I(x, y + 1, z')$ are two voxels on a feasible surface, then $|z - z'| < \Delta_y$. If Δ_x (Δ_y) is small, any feasible surface is stiff along the **x** (**y**) direction, and the stiffness decreases with larger Δ_x (Δ_y).

By defining a cost function, a cost value is computed for each voxel $I(x, y, z)$ of I, denoted by $c(x, y, z)$. Generally, $c(x, y, z)$ is an arbitrary real value that is inversely related to the likelihood that the desired surface contains the voxel $I(x, y, z)$. The cost of a surface is the total cost of all voxels on the surface. An *optimal surface* is the surface with the minimum cost among all feasible surfaces definable in the 3D volume.

A node-weighted directed graph $G = (V, E)$ is constructed according to I as follows. Every node $V(x, y, z) \in V$ represents one and only one voxel $I(x, y, z) \in I$, whose cost $w(x, y, z)$ is assigned according to:

$$w(x, y, z) = \begin{cases} c(x, y, z) & \text{if } z = 0, \\ c(x, y, z) - c(x, y, z-1) & \text{otherwise.} \end{cases} \tag{7.74}$$

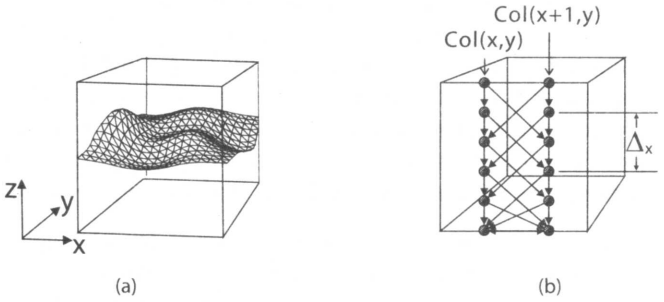


Figure 7.40: The single surface detection problem. (a) The surface orientation. (b) Two adjacent columns of the constructed directed graph. Arcs shown in dashed lines are optional.

A node $V(x, y, z)$ is *above* (resp., *below*) another node $V(x', y', z')$ if $z > z'$ (resp., $z < z'$). For each (x, y) pair with $x \in \mathbf{x}$ and $y \in \mathbf{y}$, the node subset $\{V(x, y, z) | z \in \mathbf{z}\}$ is called the (x, y)-*column* of G, denoted by $Col(x, y)$. Two (x, y)-columns are *adjacent* if their (x, y) coordinates are neighbors under a given neighborhood system. For instance, under the 4-neighbor setting, the column $Col(x, y)$ is adjacent to $Col(x + 1, y)$, $Col(x - 1, y)$, $Col(x, y + 1)$, and $Col(x, y - 1)$. Hereafter, the 4-neighbor system is assumed. The arcs of G consist of two types, *intra-column* arcs and *inter-column* arcs.

Intra-column arcs E^a*:* Along each column $Col(x, y)$, every node $V(x, y, z)$, $z > 0$ has a directed arc to the node $V(x, y, z - 1)$, i.e.,

$$E^a = \{\langle V(\mathbf{x}, \mathbf{y}, z), V(\mathbf{x}, \mathbf{y}, z-1)\rangle | z > 0\}. \tag{7.75}$$

Inter-column arcs E^r*:* Consider any two adjacent columns, $Col(x, y)$ and $Col(x + 1, y)$. Along the x-direction and for any $x \in \mathbf{x}$, a directed arc is constructed from each node $V(x, y, z) \in Col(x, y)$ to node $V(x + 1, y, \max(0, z - \Delta_x)) \in Col(x + 1, y)$. Similarly, a directed arc is connected from $V(x + 1, y, z) \in Col(x + 1, y)$ to $V(x, y, \max(0, z - \Delta_x)) \in Col(x, y)$. The same construction is done for the y-direction. These arcs enforce the smoothness constraints. In summary,

$$\begin{aligned} E^r = \; & \{\langle V(x, \mathbf{y}, z), V\{x + 1, \mathbf{y}, \max(0, z - \Delta_x)\}\rangle & | x \in \{0, \ldots, X - 2\}, z \in \mathbf{z}\} \cup \\ & \{\langle V(x, \mathbf{y}, z), V[x - 1, \mathbf{y}, \max(0, z - \Delta_x)]\rangle & | x \in \{1, \ldots, X - 1\}, z \in \mathbf{z}\} \cup \\ & \{\langle V(\mathbf{x}, y, z), V(\mathbf{x}, y + 1, \max(0, z - \Delta_y))\rangle & | y \in \{0, \ldots, Y - 2\}, z \in \mathbf{z}\} \cup \\ & \{\langle V(\mathbf{x}, y, z), V(\mathbf{x}, y - 1, \max(0, z - \Delta_y))\rangle & | y \in \{1, \ldots, Y - 1\}, z \in \mathbf{z}\} \end{aligned} \tag{7.76}$$

Intuitively, the inter-column arcs guarantee that if voxel $I(x, y, z)$ is on a feasible surface S, then its neighboring voxels on S along the **x**-direction, $I(x + 1, y, z')$ and $I(x - 1, y, z'')$, must be no 'lower' than voxel $I(x, y, \max(0, z - \Delta_x))$, i.e., $z', z'' = \max(0, z - \Delta_x)$. The same rule applies to the **y**-direction. The inter-column arcs make the node set $V(\mathbf{x}, \mathbf{y}, 0)$ *strongly connected,* meaning that in $V(\mathbf{x}, \mathbf{y}, 0)$, every node is reachable from every other node through some directed path. $V(\mathbf{x}, \mathbf{y}, 0)$ also forms the 'lowest' feasible surface that can be defined in G. Because of this, the node set $V(\mathbf{x}, \mathbf{y}, 0)$ is given a special name called the *base set,* denoted by V^B.

As presented above, the graph searching approach would only facilitate plane-like surface detection (see Figure 7.39b). However, the 3D surface to be searched often has a cylindrical shape. The method can detect circular surfaces after a straightforward extension. Let's assume that the desired surface is required to be *wraparound* along the **x**- (or **y**-) direction. The cylindrical surface is first unfolded into a terrain-like surface using cylindrical coordinate transform before applying the algorithm (Figure 7.41). Then, the first and last

rows along the unfolding plane shall satisfy the smoothness constraints. In the **x**-wraparound case, each node $V(0, y, z)$, resp., $V(X-1, y, z)$, also connects to $V\{X-1, y, \max(0, z-\Delta_x))$, resp., $V(0, y, \max(0, z-\Delta_x))$. The same rule applies to the y-wraparound case.

Multiple surface graph construction For simultaneously segmenting k ($k \geq 2$) distinct but interrelated surfaces, the optimality is not only determined by the inherent costs and smoothness properties of the individual surfaces, but also confined by their interrelations.

If surface interactions are not considered, the k surfaces S_i can be detected in k separate 3D graphs $G_i = (V_i, E_i) = (V_i, E_i^a \cup E_i^r)$, $i = 1,..., k$. Each G_i is constructed in the way presented above. The node costs are computed utilizing k cost functions (not necessarily distinct), each of which is designed for searching one surface. Taking the surface interrelations into account, another set of arcs E^s is needed, forming a directed graph $G(V, E)$ in 4D space with $V = U^k_{i=1} V_i$ and $E = U^k_{=1} E_i \cup E^s$. The arcs in E^s are called *inter-surface* arcs, which model the pairwise relations between surfaces. For each pair of the surfaces, their relations are described using two parameters, $\delta^l \geq 0$ and $\delta^u \geq 0$, representing the **surface separation constraint**.

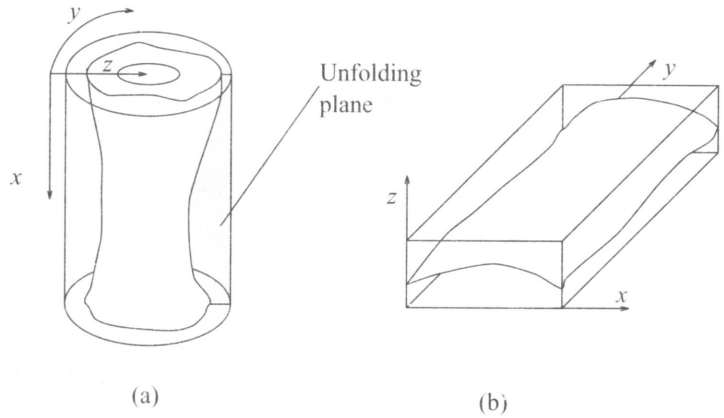


Figure 7.41: Image unfolding. (a) A tubular object in a volumetric image. (b) 'Unfolding' the tubular object in (a) to form a new 3D image. The boundary of the tubular object in the original data corresponds to the surface to be detected in the unfolded image.

The construction of E^s for double-surface segmentation is detailed below. The ideas can easily be generalized to handling more than two surfaces. In many practical problems, the surfaces are expected not to intersect or overlap. Suppose that for two surfaces S_1 and S_2 to be detected, the prior knowledge requires S_2 being below S_1. Let the minimum distance between them be δ^l voxel units, and the maximum distance be δ^u voxel units. Let the 3D graphs used for the search of S_1 and S_2 be G_1 and G_2, respectively, and let $Col_1(x, y)$ and $Col_2(x, y)$ denote two corresponding columns in G_1 and G_2.

For any node $V_1(x, y, z)$ in $Col_1(x, y)$ with $z \geq \delta^u$, a directed arc in E^s connecting $V_1(x, y, z)$ to $V_2(x, y, z - \delta^u)$ is constructed. Also, for each node $V_2(x, y, z)$ in $Col_2(x, y)$ with $z < Z - \delta^l$, a directed arc in E^s connecting $V_2(x, y, z)$ to $V_1(x, y, z + \delta^l)$ is introduced. This construction is applied to every pair of corresponding columns of G_1 and G_2.

Because of the separation constraint (S_2 is at least δ^l voxel units below S_1), any node $V_1(x, y, z)$ with $z < \delta^l$ cannot be on surface S_1. Otherwise, no node in $Col_2(x, y)$ could be on surface S_2. Likewise, any node $V_2(x, y, z)$ with $z \geq Z - \delta^l$ cannot belong to surface S_2. These nodes that are impossible to appear in any feasible solution for the problem are called *deficient* nodes. Hence, for each column $Col_1(x, y) \in G_1$, it is safe to remove all nodes $V_1(x, y, z)$ with $z \geq \delta^l$ and their incident arcs in E_1. Similarly, for each column $Col_2(x, y) \in G_2$, all nodes $V_2(x, y, z)$ with $z \geq Z - \delta^l$ and their incident arcs in E_2 can be safely eliminated.

Due to the removal of deficient nodes, the base set of G_1 becomes $V_1(x, y, \delta^l)$. Correspondingly, the cost of each node $V_1(x, y, \delta^l)$ is modified as $w_1(x, y, \delta^l) = c_1(x, y, \delta^l)$, where $c_1(x, y, \delta^l)$ is the original cost of voxel $I(x, y, \delta^l)$ for surface S_1. The inter-column arcs of G_1 are modified to make $V_1(x, y, \delta^l)$ strongly connected. The base set of G then becomes $V^B = V_1(x, y, \delta^l) \cup V_2(x, y, 0)$. The directed arcs $(V_1(0,0, \delta^l), V_2(0,0,0))$ and $(V_2(0, 0, 0), V_1(0,0, \delta^l))$ are introduced to E^s to make V_B strongly connected.

In summary, the inter-surface arc set E^s for modeling non-crossing surfaces is constructed as

$$E^s = \{\langle V_1(\mathbf{x}, \mathbf{y}, z), V_2(\mathbf{x}, \mathbf{y}, z - \delta^u)\rangle \mid z \geq \delta^u\} \cup \{\langle < V_1(0, 0, \delta^l), V_2(0,0, 0)\rangle\}$$
$$\cup \{\langle V_2(\mathbf{x}, \mathbf{y}, z), V_1(\mathbf{x}, \mathbf{y}, z + \delta^l)\rangle \mid z < Z - \delta^l\} \cup \{\langle V_2(0,0,0), V_1(0, 0, \delta^l)\rangle\} \quad (7.77)$$

In other situations, two interacting surfaces may be allowed to cross each other. This may be encountered when tracking a moving surface over time. For these problems, instead of modeling the minimum and maximum distances between them, δ^l and δ^u specify the maximum distances that a surface can vary below and above the other surface, respectively. The inter-surface arcs for this case consist of the following: $(V_1(x, y, z), V_2(x, y, \max(0, z - \delta^l)))$ and $(V_2(x, y, z), V_1(x, y, \max(0, z - \delta^u)))$ for all $x \in x$, $y \in y$ and $z \in z$. A summary of all cases is illustrated in Figure 7.42.

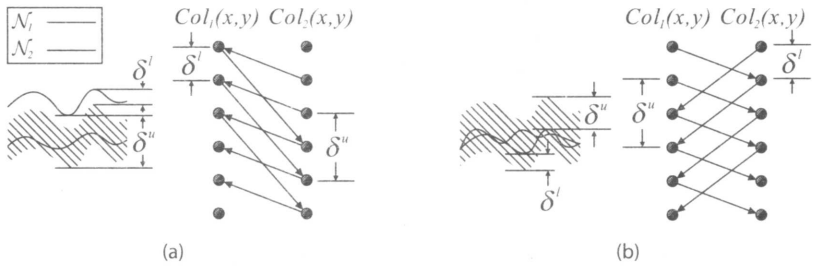


Figure 7.42: Summary of surface interrelation modeling. S_1 and S_2 are two desired surfaces. $Col_1(x, y)$ and $Col_2(x, y)$ are two corresponding columns in the constructed graphs. Arcs shown in dashed lines are optional. (a) The non-crossing case. (b) The case with crossing allowed.

Surface detection algorithm The segmentation of optimal surfaces is formulated as computing a minimum closed set in a geometric graph constructed from I. The time bound of the algorithm is independent of both the smoothness parameters (Δ_{xi} and Δ_{yi}, $i = 1,..., k$) and the surface separation parameters ($\delta^l_{i,i+1}$ and $\delta^u_{i,i+1}$, $i = 1,..., k - 1$). Note that improper specifications of these constraints may lead to an infeasible problem, i.e., the constraints are self-conflicting and thus no k surfaces satisfying all the constraints exist in I.

In the single-surface case, for any feasible surface S in I, the subset of nodes on or below S in G, namely $Z = \{V(x, y, z) \mid z < S(x, y)\}$, forms a closed set in G. It can be observed that if $V(x, y, z)$ is in the closed set Z, then all nodes below it on $Col(x, y)$ are also in Z. Moreover, due to the node cost assignments in equation (7.74), the costs of S and Z are equal. In fact, as proven in [Wu and Chen, 2002], any feasible S *in* I uniquely corresponds to a nonempty closed set Z in G with the same cost. This is a key observation to transforming the optimal surface problem into seeking a minimum closed set in G.

Computation of a minimum-cost *nonempty* closed set $Z*$ in G is a well studied problem in graph theory. As given elsewhere [Picard, 1976; Hochbaum, 2001; Wu and Chen, 2002], $Z*$ in G can be obtained by computing a minimum $s - t$ cut in a related graph G_{st}. Let V^+ and V^- denote the sets of nodes in G with non-negative and negative costs, respectively. Define a new directed graph $G_{st} = (V \cup \{s, t\}, E \cup E_{st})$. An infinite cost is assigned to each arc in E. E_{st} consists of the following arcs: The source s is connected to each node $v \in V^-$ by a directed arc of cost $-w(v)$; every node $v \in V+$ is connected to the sink t by a directed arc of cost $w(v)$. Let (S, T) denote a finite-cost $s - t$ cut in G_{st}, and $c(S, T)$ denote the total cost of the cut. It was shown that

$$c(S, T) = -w(V^-) + \sum_{v \in S - \{s\}} w(v), \quad (7.78)$$

where $\omega(V^-)$ is fixed and is the cost sum of all nodes with negative costs in G. Since $S \setminus \{s\}$ is a closed set in G [Picard, 1976; Hochbaum, 2001], the cost of a cut (S, T) in G_{st} and the cost of the corresponding closed set in G differ by a constant. Hence, the source set $S* \setminus \{s\}$ of a minimum cut in G_{st} corresponds to a minimum closed set $Z*$ in G. Because the graph G_{st} has $\mathcal{O}(kn)$ nodes and $\mathcal{O}(kn)$ arcs, the minimum closed set $Z*$ in G can be computed in $T(kn, kn)$ time.

For the multiple surface case, the optimal k surfaces correspond to the upper envelope of the minimum closed set $Z*$. For each i ($i = 1,..., k$), the subgraph G_i is used to search for the target surface S_i. For every $x \in x$ and $y \in y$, let $V_i^B(x, y)$ be the subset of nodes in both $Z*$ and the (x, y)-column $Col_i(x, y)$ of G_i, i.e., $V_i^B(x, y) = Z* \cap Col_i(x, y)$. Denote by $V_i(x, y, z*)$ the node in $V_i^B(x, y)$ with the largest z-coordinate. Then, voxel $I(x, y, z*)$ is on the i-th optimal surface S_i*. In this way, the minimum closed set $Z*$ of G uniquely defines the optimal k surfaces $\{S_1*,... ,S_k*\}$ in I.

Algorithm 7.9: Multiple optimal surface segmentation

1. Determine parameters representing a priori knowledge about the number of surfaces and the hard and soft segmentation constraints: k, Δ_x, Δ_y, δ^l, δ^u, cost function(s).
2. Construct graph $G_{st} = (V \cup \{s, t\}, E \cup E_{st})$.
3. Compute the minimum $s - t$ cut $(S*, T*)$ in G_{st}.
4. Recover the k optimal surfaces from $S* \setminus \{s\}$.

Cost functions Designing appropriate cost functions is of paramount importance for any graph-based segmentation method. In real-world problems, the cost function usually reflects either a region-based or edge-based property of the surface to be identified.

Edge-based cost functions A typical edge-based cost function aims to accurately position the boundary surface in the volumetric image. Several alternative cost functions were presented in Section 6.2.4. An advanced version of an edge-based cost function may utilize a combination of the first and second derivatives of the image intensity function [Sonka et al., 1997], and may consider preferred directions of the identified surface. The combination of the first and second derivatives permits fine-tuning of the cost function to maximize border positioning accuracy.

Let the analyzed volumetric image be I (x, y, z). Then, the cost $c(x, y, z)$ assigned to the image voxel $I(x, y, z)$ can be constructed as:

$$c(x, y, z) = -e(x, y, z) \cdot p(\phi(x, y, z)) + q(x, y, z), \tag{7.79}$$

where $e(x, y, z)$ is a raw edge response derived from the first and second derivatives of the image, $\phi(x, y, z)$ denotes the edge orientation at location (x, y, z) that is reflected in the cost function via an orientation penalty $p(\phi(x, y, z))$. $0 < p < 1$ when $\phi(x, y, z)$ falls outside a specific range around the preferred edge orientation; otherwise $p = 1$. A position penalty term $q(x, y, z) > 0$ may be incorporated so that a priori knowledge about expected border position can be modeled.

$$e(x, y, z) = (1 - |\omega|) \cdot (I * \mathcal{M}_{\text{first derivative}})(x, y, z) \dotplus \omega \cdot (I * \mathcal{M}_{\text{second derivative}})(x, y, z). \tag{7.80}$$

The $\dotplus$ operator stands for a pixel-wise summation, and $*$ is a convolution operator. The weighting coefficient $-1 \leq \omega \leq 1$ controls the relative strength of the first and second derivatives, allowing accurate edge positioning. The values of ω, p, q may be determined from a desired boundary surface positioning information in a training set of images; values of ω are frequently scale dependent.

Region based cost functions The object boundaries do not have to be defined by gradients as discussed in Section 7.3 (and shown in 2D in equation (7.45)). In 3D, the Chan-Vese functional is

$$C(S, a_1, a_2) = \int_{\text{inside}(S)} (I(x, y, z) - a_1)^2 \, dx \, dy \, dz + \int_{\text{outside}(S)} (I(x, y, z) - a_2)^2 \, dx \, dy \, dz. \tag{7.81}$$

As in equation (7.45), a_1 and a_2 are the mean intensities in the interior and exterior of the surface S and the energy $C(S, a_1, a_2)$ is minimized when S coincides with the object boundary, and best separates the object and background with respect to their mean intensities.

The variance functional can be approximated using a per-voxel cost model, and in turn be minimized using a graph-based algorithm. Since the application of the Chan-Vese cost functional may not be immediately obvious, consider a single-surface segmentation example. Any feasible surface uniquely partitions the graph into two disjoint subgraphs. One subgraph consists of all nodes that are on or below the surface, and the other subgraph consists of all nodes that are above the surface. Without loss of generality, let a node on or below a feasible surface be considered as being inside the surface; otherwise let it be outside the surface. Then, if a node $V(x', y', z')$ is on a feasible surface S, then the nodes $V(x',y',z)$ in $Col(x',y')$ with $z = z'$ are all inside S, while the nodes $V(x', y', z)$ with $z \leq z'$ are all outside S. Hence, the voxel cost $c(x', y', z')$ is assigned as the sum of the inside and outside variances computed in the column $Col(x', y')$, as follows

$$c(x', y', z') = \sum_{z \leq z'} \left(I(x', y', z) - a_1\right)^2 + \sum_{z > z'} \left(I(x', y', z) - a_2\right)^2. \tag{7.82}$$

Then, the total cost of S will be equal to cost $C(S, a_1, a_2)$ (discretized on the grid (x, y, z)). However, the constants a_1 and a_2 are not easily obtained, since the surface is not well-defined before the global optimization is performed. Therefore, the knowledge of which part of the graph is inside and outside is unavailable. Fortunately, the graph construction guarantees that if $V(x', y', z')$ is on S, then the nodes $V(x, y, z_1)$ with $z_1 = \{z \mid z \leq \max(0, z' - |x - x'|\Delta_x - |y - y'|\Delta_y)\}$ are in the closed set Z corresponding to S. Accordingly, the nodes $V(x, y, z_2)$ with $z_2 = \{z \mid z' + x - x'| \Delta_x + |y - y'|\Delta_y < z < Z\}$ must not be in Z. This implies that if the node $V(x', y', z')$ is on a feasible surface S, then the nodes $V(x, y, z_1)$ are inside S, while the nodes $V(\mathbf{x}, \mathbf{y}, \mathbf{z}_2)$ are outside S.

Consequently, $a_1(x', y', z')$ and $\hat{a}_2(x', y', z')$ can be computed, that are approximations of the constants a_1 and a_2 for each voxel $I(x', y', z')$

$$a_1(x', y', z') = \text{mean}(I(\mathbf{x}, \mathbf{y}, \mathbf{z}_1)), \tag{7.83}$$

$$\hat{a}_2(x', y', z') = \text{mean}(I(\mathbf{x}, \mathbf{y}, \mathbf{z}_2)). \tag{7.84}$$

The estimates are then used in equation (7.82) instead of a_1 and a_2.

Examples To demonstrate the method's behavior, let's first look at segmenting a simple computer-generated volumetric image shown in Figure 7.43a, which however is difficult to segment. This image consists of 3 identical slices stacked together to form a 3D volume. The gradual change of intensity causes the gradient strengths to locally vanish. Consequently, border detection using an edge-based cost function fails locally (Figure 7.43b). Using a cost function that includes a shape term produces a good result (Figure 7.43c). Figure 7.43d demonstrates the method's ability to segment both borders of the sample image.

Figure 7.44 presents segmentation examples obtained using the minimum-variance cost function in images with no apparent edges. The objects and background were differentiated by their respective textures. In Figure 7.44, curvature and edge orientation were used instead of original image data [Chan and Vese, 2001]. The two boundaries in Figure 7.44c,d were segmented simultaneously.

The optimal surface detection method has been used in a number of medical image analysis applications involving volumetric medical images from CT, MR, and ultrasound scanners. Figure 7.45 shows a comparison of segmentation performance in human pulmonary CT images. To demonstrate the ability of handling more than two interacting surfaces, four surfaces of excised human ilio-femoral specimens-lumen, intima-media (internal elastic lamina (IEL)), media-adventitia (external elastic lamina (EEL)), and the outer wall-were segmented in vascular MR images. The optimal multiple-surface segmentation clearly outperformed the previously used 2D approach [Yang et al., 2003] and did not require any interactive guidance (Figure 7.46).

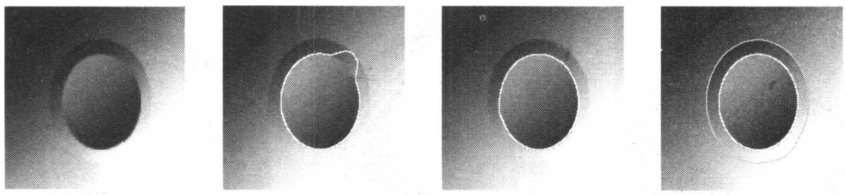


Figure 7.43: Single-surface versus coupled-surfaces. (a) Cross-section of the original image. (b) Single surface detection using the method with standard edge-based cost function. (c) Single surface detection using the algorithm and a cost function with a shape term. (d) Double-surface segmentation.

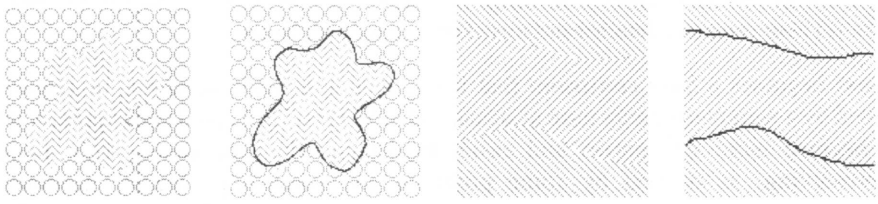


Figure 7.44: Segmentation using the minimum-variance cost function. (a,c) Original images. (b,d) The segmentation results.

The optimal surface detection method remains fully compatible with conventional graph searching. For example, when employed in 2D, it produces an identical result when the same objective function and hard constraints are employed. Consequently, many existing problems that were tackled using graph-searching in a slice-by-slice manner can be migrated to this framework with little or no change to the underlying objective function. Comparing to other techniques, one of the major innovations is that the smoothness constraint can be modeled in a graph with a non-trivial arc construction.

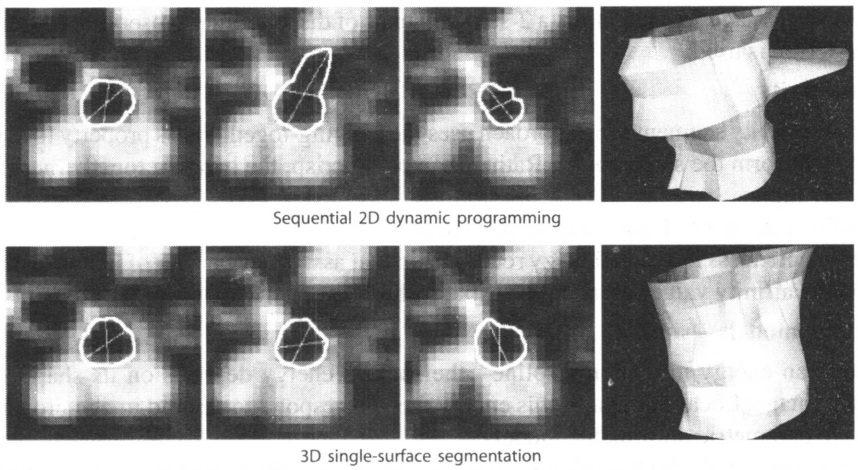
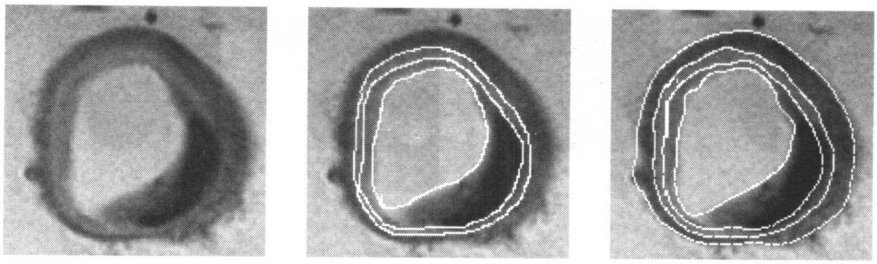


Figure 7.45: Comparison of 2D and 3D inner airway wall segmentation results. A preliminary airway tree segmentation is shown in Figure 7.21. The three bottom and top left panels demonstrate resampling of airway segments to obtain orthogonal slices on which the border detection is performed. Results of 2D slice-by-slice dynamic programming approach in three consecutive slices together with 3D surface rendering of the entire segment (10 slices) is shown in the upper row. The bottom row shows the same segment with the luminal surface detected using the optimal 3D graph searching approach. Note the failure of the 2D approach in one of the slices.



(a) (b) (c)

Figure 7.46: Multi-surface segmentation of arterial wall and plaque in volumetric MR images. (a) Original MR image of femoral artery cross-section–the volumetric 3D image consisted of 16 cross sections. (b) Three manually identified wall layer borders. (c) Four computer-detected surfaces of plaque and wall layers.

Thus, smoothness becomes a *hard constraint* that has a clear geometric meaning, as opposed to a *soft constraint* defined by a weighted energy term as discussed in Section 7.5.1. As a consequence, the objective function may become more transparent and easier to design. The smoothness thus modeled is not discontinuity-preserving, as desired by some problems in vision (e.g., stereo, multicamera scene construction). However, discontinuity-preservation is not always desirable. The presented ability to identify multiple coupled surfaces in an optimal way is a major advance in graph search-based segmentation.

Summary

- **Mean shift segmentation**
  - Mean shift approach is a non-parametric technique for the analysis of a complex multi-modal feature space and identification of feature clusters.
  - The only free parameters of the mean shift process are the size and shape of the region of interest, i.e., the multivariate density kernel estimator.
  - Density estimation is modified so that the density gradient is estimated.
  - For mean shift image segmentation, a 2-step sequence of discontinuity preserving filtering and mean shift clustering is used.
- **Fuzzy connectivity**
  - Fuzzy connectivity segmentation approach uses the hanging-togetherness property to identify image elements that form the same object. Rather than being crisp, the hanging togetherness is described using fuzzy logic.
  - Fuzzy affinity describes local fuzzy relationships.
  - Fuzzy connectedness is a global fuzzy relationship that assigns every pair of image elements a value based on the affinity values along all possible paths between these two image elements.
- **Active contour models—snakes**
  - A snake is an energy minimizing spline—the snake's energy depends on its shape and location within the image. Local minima of this energy then correspond to desired image properties.
  - Snakes are parametric deformable models.
  - The energy functional which is minimized is a weighted combination of internal and external forces.
  - Gradient vector flow field increases the effective area of snake attraction decreasing the snake's sensitivity to initialization and allowing to segment concave boundaries.
- **Geometric deformable models**
  - Geometric deformable models represent the developing surfaces by partial differential equations.
  - The movements of the propagating fronts are described by speed functions.

  - The evolving curves and/or surfaces are represented as level sets of higher dimensional functions yielding seamless treatment of topologic changes.

- **Simultaneous border detection**
  - Simultaneous border detection facilitates optimal identification of border pairs by finding an optimal path in a three-dimensional graph.
  - It is based on the observation that there is information contained in the position of one border that might be useful in identifying the position of the other border. After a cost function that combines edge information from the left and right borders has been defined, either heuristic graph searching or dynamic programming methods can be used for optimal border detection.
- **Sub-optimal surface detection**
  - Sub-optimal surface detection uses multi-dimensional graph search to identify legal surfaces in three- or higher-dimensional image data.
  - Surface growing is based on dynamic programming and avoids the problem of combinatorial explosion by introducing local conditions that must be satisfied by all legal surfaces.
- **Direct graph cut segmentation**
  - Graph cuts solve a region-based segmentation problem by the use of minimum $s - t$ cut / maximum flow combinatorial optimization algorithms.
  - The segmentation outcome is controlled by hard and soft constraints, and a cost function.
  - The minimum $s - t$ cut problem is solved by finding a maximum flow from the source s to the sink t.
- **Optimal single and multiple surface segmentation**
  - Single and multiple interactive surfaces are identified by optimal graph searching in a transformed graph.
  - Combinatorial explosion in computation is avoided by transforming the problems into computing minimum $s - t$ cuts.
  - Despite the used graph-cut optimization, the method is principally different from the direct graph cut segmentation approach.
  - Multiple interacting surfaces can be identified by incorporating mutual surface-to-surface interrelationships as inter-surfaces arcs in $n + 1$-dimensional graphs.a

8

Chapter

Shape Representation and Description

The last chapter was devoted to image segmentation methods and showed how to construct homogeneous regions of images and/or their boundaries. Recognition of image regions is an important step on the way to understanding image data, and requires an exact region description in a form suitable for a classifier (Chapter 9). This description should generate a numeric feature vector, or a non-numeric syntactic description word, which characterizes properties (for example, shape) of the region. Region description is the third of the four levels given in Chapter 4, implying that the description already comprises some abstraction—for example, 3D objects can be represented in a 2D plane and shape properties that are used for description are usually computed in two dimensions. If we are interested in a 3D object description, we have to process at least two images of the same object taken from different viewpoints (stereo vision), or derive the 3D shape from a sequence of images if the object is in motion. A 2D shape representation is sufficient in the majority of practical applications, but if 3D information is necessary—if, say, 3D object reconstruction is the processing goal, or the 3D characteristics bear the important information—the object description task is much more difficult; these topics are introduced in Chapter 11. In the following sections, we will limit our discussion to 2D shape features and proceed under the assumption that object descriptions result from the image segmentation process.

Defining the shape of an object can prove to be very difficult. Shape is usually represented verbally or in figures, and people use terms such as *elongated*, *rounded*, *with sharp edges*, etc. The computer era has introduced the necessity to describe even very complicated shapes precisely, and while many practical shape description methods exist, there is no generally accepted methodology of shape description. Further, it is not known what is important in shape. Current approaches have both positive and negative attributes; computer graphics [Woodwark, 1986] or mathematics [Lord and Wilson, 1984] use effective shape representations which are unusable in shape recognition [Juday, 1988] and vice versa. In spite of this, it is possible to find features common to most shape description approaches. Location and description of substantial variations in the first derivative of object boundaries often yield suitable information. Examples include alphanumeric optical character recognition (OCR), technical drawings, electro-cardiogram (ECG) curve characterization, etc.

Shape is an object property which has been carefully investigated in recent years and many papers may be found dealing with numerous applications—OCR, ECG analysis, electro-encephalogram (EEG) analysis, cell classification, chromosome recognition, automatic inspection, technical diagnostics, etc. Despite this variety, differences among many approaches are limited mostly to terminology. These common methods can be characterized from different points of view:

- Input representation form: Object description can be based on boundaries (contour-based, external) or on more complex knowledge of whole regions (region-based, internal).

- Object reconstruction ability: That is, whether an object's shape can or cannot be reconstructed from the description. Many varieties of shape-preserving methods exist. They differ in the degree of precision with respect to object reconstruction.
- Incomplete shape recognition ability: That is, to what extent an object's shape can be recognized from the description if objects are occluded and only partial shape information is available.
- Local/global description character: Global descriptors can only be used if complete object data are available for analysis. Local descriptors describe local object properties using partial information about the objects. Thus, local descriptors can be used for description of occluded objects.
- Mathematical and heuristic techniques: A typical mathematical technique is shape description based on the Fourier transform. A representative heuristic method may be elongatedness.
- Statistical or syntactic object description (Chapter 9).
- A robustness of description to translation, rotation, and scale transformations: Shape description properties in different resolutions.

The role of different description methods in image analysis and image understanding is illustrated by the flowchart shown in Figure 8.1.

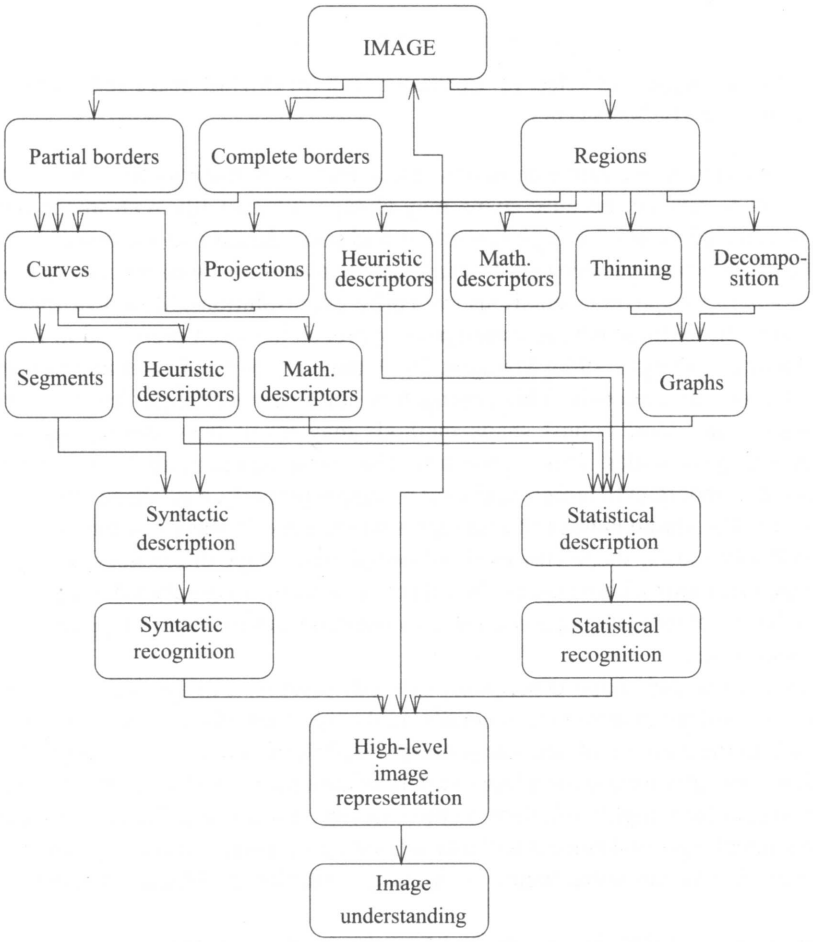


Figure 8.1: Image analysis and understanding methods.

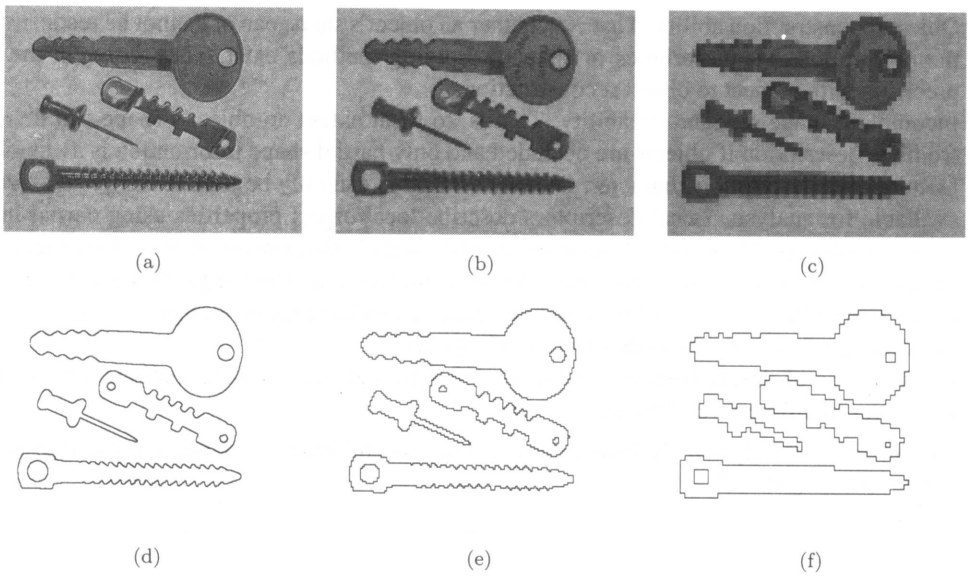


Figure 8.2: (a) Original image 640 × 480. (d) Contours of (a). (b) Original image 160 × 120. (e) Contours of (b). (c) Original image 64 × 48. (f) Contours of (c).

Problems of scale (resolution) are common in digital images. Sensitivity to scale is even more serious if a shape description is derived, because shape may change substantially with image resolution. Contour detection may be affected by noise in high resolution, and small details may disappear in low resolution (see Figure 8.2). Therefore, shape has been studied in multiple resolutions which again causes difficulties with matching corresponding shape representations from different resolutions. Moreover, the conventional shape descriptions change discontinuously. A **scale-space** approach has been presented in [Babaud et al., 1986; Witkin, 1986; Yuille and Poggio, 1986; Maragos, 1989] that aims to obtain continuous shape descriptions if the resolution changes continuously. This approach is not a new technique itself, but is an extension of existing techniques, and more robust shape methods may result from developing and retaining their parameters over a range of scales. This approach will be mentioned in more detail in Section 8.2.4.

In many tasks, it is important to represent classes of shapes properly, e.g., shape classes of apples, oranges, pears, bananas, etc. The **shape classes** should represent the generic shapes of the objects belonging to the same classes well. Obviously, shape classes should emphasize shape differences among classes, while the influence of shape variations within classes should not be reflected in the class description. Current research challenges includes development of approaches to automated learning about shape and reliable definition of shape classes (Section 8.4).

Object representation and shape description methods discussed in the following sections are not an exhaustive list—we will try to introduce generally applicable methods. It is necessary to apply a problem-oriented approach to the solution of specific problems of description and recognition. This means that the following methods are appropriate for a large variety of descriptive tasks and the following ideas may be used to build a specialized, highly efficient method suitable for a particular problem description. Such a method will no longer be general since it will take advantage of a priori knowledge about the problem. This is the way human beings can solve their vision and recognition problems, by using highly specialized knowledge.

It should be understood that despite the fact that we are dealing with two-dimensional shape and its description, our world is three-dimensional and the same objects, if seen from different angles (or changing position/orientation in space), may form very different 2D projections (see Chapter 11). The ideal case would be to have a universal shape descriptor capable of overcoming these changes—to design projection-invariant

descriptors. Consider an object with planar faces and imagine how many very different 2D shapes may result from a given face if the position and 3D orientation of this simple object changes with respect to an observer. In some special cases, such as circles which transform to ellipses, or planar polygons, projectively invariant features (called **invariants**) can be found. Unfortunately, no existing shape descriptor is perfect; in fact, they are all far from being perfect. Therefore, a very careful choice of descriptors resulting from detailed analysis of the shape recognition problem must precede any implementation, and whether or not a 2D representation is capable of describing a 3D shape must also be considered. For some 3D shapes, their 2D projection may bear enough information for recognition—aircraft contours are a good example; successful recognition of airplanes from projections are known even if they change their position and orientation in space. In many other cases, objects must be seen from a specific direction to get enough descriptive information—human faces are such a case.

Object occlusion is another hard problem in shape recognition. However, the situation is easier here (if pure occlusion is considered, not combined with orientation variations yielding changes in 2D projections as discussed above), since visible parts of objects may be used for description. Here, the shape descriptor choice must be based on its ability to describe local object properties—if the descriptor gives only a global object description (e.g., object size, average boundary curvature, perimeter), such a description is useless if only a part of an object is visible. If a local descriptor is applied (e.g., description of local boundary changes), this information may be used to compare the visible part of the object to all objects which may appear in the image. Clearly, if object occlusion occurs, the local or global character of the shape descriptor must be considered first.

In Sections 8.2 and 8.3, descriptors are sorted according to whether they are based on object boundary information (contour-based, external description) or whether the information from object regions is used (region-based, internal description). This classification of shape description methods corresponds to previously described boundary-based and region-based segmentation methods. However, both contour-based and region-based shape descriptors may be local or global and differ in sensitivity to translation, rotation, scaling, etc.

8.1 REGION IDENTIFICATION

Region identification is necessary for region description. One of the many methods for region identification is to label each region (or each boundary) with a unique (integer) number; such identification is called **labeling** or **coloring** (also connected component labeling), and the largest integer label usually gives the number of regions in the image. Another method is to use a smaller number of labels (four is theoretically sufficient [Appel and Haken, 1977; Saaty and Kainen, 1977; Nishizeki and Chiba, 1988; Wilson and Nelson, 1990]), and ensure that no two neighboring regions have the same label; then information about some region pixel must be added to the description to provide full region reference. This information is usually stored in a separate data structure. Alternatively, mathematical morphology approaches (Chapter 13) may be used for region identification.

Assume that the segmented image R consists of m disjoint regions R_i (as in equation (6.1)). The image R often consists of objects and a background

$$R_b^C = \bigcup_{i=1 i \neq b}^{m} R_i ,$$

where R^C is the set complement, R_b is considered background, and other regions are considered objects. Input to a labeling algorithm is usually either a binary or multi-level image, where background may be represented by zero pixels, and objects by non-zero values. A multi-level image is often used to represent the labeling result, background being represented by zero values, and regions represented by their non-zero labels. Algorithm 8.1 presents a sequential approach to labeling a segmented image.

Algorithm 8.1: 4-neighborhood and 8-neighborhood region identification

1. First pass: Search the entire image R row by row and assign a non-zero value v to each non-zero pixel $R(i, j)$. The value v is chosen according to the labels of the pixel's neighbors, where the property *neighboring* is defined by Figure 8.3. ('neighbors' outside the image R are not considered),
   - If all the neighbors are background pixels (with pixel value zero), $R(i, j)$ is assigned a new (and as yet) unused label.
   - If there is just one neighboring pixel with a non-zero label, assign this label to the pixel $R(i, j)$.
   - If there is more than one non-zero pixel among the neighbors, assign the label of any one to the labeled pixel. If the labels of any of the neighbors differ (*label collision*), store the label pair as being equivalent. Equivalence pairs are stored in a separate data structure—an equivalence table.
2. Second pass: All of the region pixels were labeled during the first pass, but some regions have pixels with different labels (due to label collisions). The whole image is scanned again, and pixels are re-labeled using the equivalence table information (for example, with the lowest value in an equivalence class).

Label collision is a very common occurrence—examples of image shapes experiencing this are U-shaped objects, mirrored E (∃) objects, etc. (see Figure 8.3c). The equivalence table is a list of all label pairs present in an image; all equivalent labels are replaced by a unique label in the second step. Since the number of label collisions is usually not known beforehand, it is necessary to allocate sufficient memory to store the equivalence table in an array. A dynamically allocated data structure is recommended. Further, if pointers are used for label specification, scanning the image for the second time is not necessary (the second pass of the algorithm) and only rewriting labels to which these pointers are pointing is much faster.

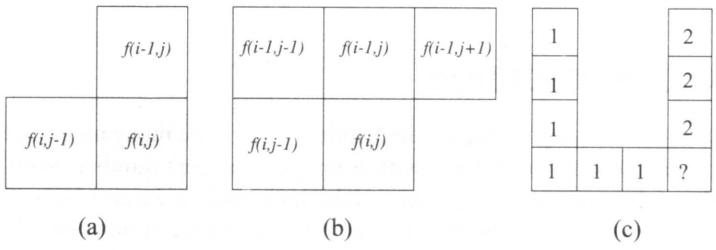


Figure 8.3: Masks for region identification. (a) 4-connectivity. (b) 8-connectivity. (c) Label collision.

The algorithm is basically the same in 4-connectivity and 8-connectivity, the only difference being in the neighborhood mask shape (Figure 8.3b). It is useful to assign the region labels incrementally to permit the regions to be counted easily in the second pass. An example of partial results is given in Figure 8.4.

Region identification can be performed on images that are not represented as straightforward matrices; the following algorithm [Rosenfeld and Kak, 1982] may be applied to images that are run length encoded (see Chapter 4).

Algorithmic details and the procedure for looking for neighboring leaf nodes can be found in [Rosenfeld and Kak, 1982; Samet, 1984].

The **region counting** task is closely related to the region identification problem. As we have seen, object counting can be an intermediate result of region identification. If it is only necessary to count regions with no need to identify them, a one-pass algorithm is sufficient [Rosenfeld and Kak, 1982; Atkinson et al., 1985].

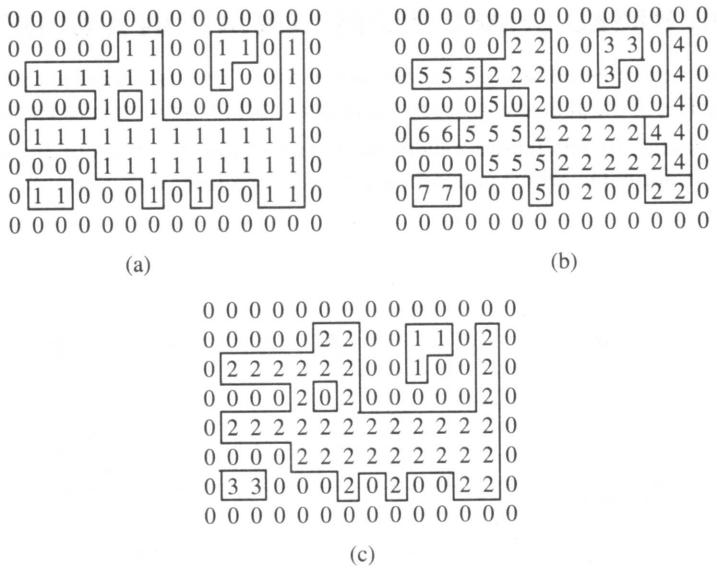


Figure 8.4: Object identification in 8-connectivity. (a), (b), (c) Algorithm steps. Equivalence table after step (b): 2-5, 5-6, 2-4.

Algorithm 8.2: Region identification in run length encoded data

1. First pass: Use a new label for each continuous run in the first image row that is not part of the background.
2. For the second and subsequent rows, compare positions of runs.
   - If a run in a row does not neighbor (in the 4- or 8-sense) any run in the previous row, assign a new label.
   - If a run neighbors precisely one run in the previous row, assign its label to the new run.
   - If the new run neighbors more than one run in the previous row, a label collision has occurred.

   Collision information is stored in an equivalence table, and the new run is labeled using the label of any one of its neighbors.
3. Second pass: Search the image row by row and re-label the image according to the equivalence table information.

If the segmented image is represented by a quadtree data structure, the following algorithm may be applied.

Algorithm 8.3: Quadtree region identification

1. First pass: Search quadtree nodes in a given order—e.g., beginning from the root and in the NW, NE, SW, SE directions. Whenever an unlabeled non-zero leaf node is entered, a new label is assigned to it. Then search for neighboring leaf nodes in the E and S directions (plus SE in 8-connectivity). If those leaves are non-zero and have not yet been labeled, assign the label of the node from which the search started. If the neighboring leaf node has already been labeled, store the collision information in an equivalence table.
2. Repeat step 1 until the whole tree has been searched.
3. Second pass: Re-label the leaf nodes of the quadtree according to the equivalence table.

8.2 CONTOUR-BASED SHAPE REPRESENTATION AND DESCRIPTION

Region borders must be expressed in some mathematical form. The **rectangular** representation of x_n pixel co-ordinates as a function of the path length n is most common. Other useful representations are (see Figure 8.5):

- **Polar** co-ordinates, in which border elements are represented as pairs of angle ö and distance r;
- **Tangential** co-ordinates, which codes the tangential directions $\theta(x_n)$ of curve points as a function of path length n.

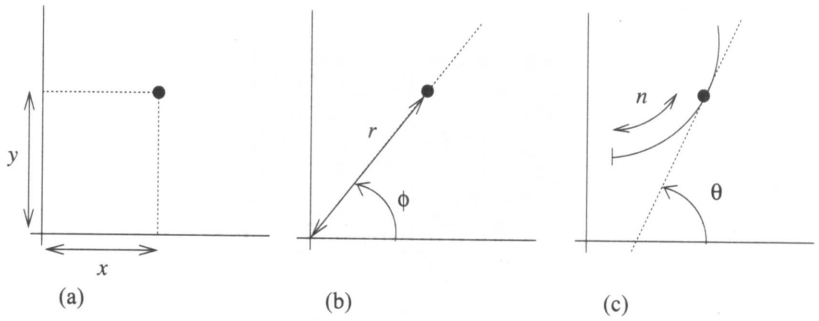


Figure 8.5: Co-ordinate systems. (a) Rectangular (Cartesian). (b) Polar. (c) Tangential.

8.2.1 Chain codes

Chain codes describe an object by a sequence of unit-size line segments with a given orientation (see Section 4.2.2). The first element of such a sequence must bear information about its position to permit the region to be reconstructed. The process results in a sequence of numbers (see Figure 8.6); to exploit the position invariance of chain codes the first element, which contains the position information, is omitted. This definition of the chain code is known as **Freeman's** code [Freeman, 1961]. Note that a chain code object description may easily be obtained as a by-product of border detection; see Section 6.2.3 for a description of border detection algorithms.

If the chain code is used for matching, it must be independent of the choice of the first border pixel in the sequence. One possibility for normalizing the chain code is to find the pixel in the border sequence which results in the minimum integer number if the description chain is interpreted as a base 4 number—that pixel is then used as the starting pixel [Tsai and Yu, 1985]. A *mod 4* or *mod 8* difference code, called a chain code **derivative**, is another numbered sequence that represents relative directions of region boundary elements, measured as multiples of counter-clockwise 90° or 45° direction changes (Figure 8.6). A chain code is very sensitive to noise, and arbitrary changes in scale and rotation may cause problems if used for recognition. The smoothed version of the chain code (averaged directions along a specified path length) is less noise sensitive.

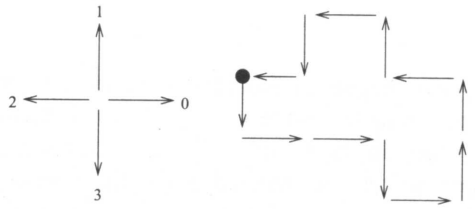


Figure 8.6: Chain code in 4-connectivity, and its derivative. Cod e: 3, 0, 0, 3, 0, 1, 1, 2, 1, 2, 3, 2; derivative: 1, 0, 3, 1, 1, 0, 1, 3, 1, 1, 3, 1.

8.2.2 Simple geometric border representation

The following descriptors are based mostly on geometric properties of described regions. Because of the discrete character of digital images, all of them are sensitive to image resolution.

Boundary length

Boundary length is an elementary region property, that is simply derived from the chain code representation. Vertical and horizontal steps have unit length, and the length of diagonal steps in 8-connectivity is $\sqrt{2}$. It can be shown that the boundary is longer in 4-connectivity, where a diagonal step consists of two rectangular steps with a total length of 2. A closed-boundary length (**perimeter**) can also be easily evaluated from run length or quadtree representations. Boundary length increases as the image raster resolution increases; on the other hand, region area is not affected by higher resolution and converges to some limit (see also the description of fractal dimension in Section 15.1.6). To provide continuous-space perimeter properties (area computation from the boundary length, shape features, etc.), it is better to define the region border as being the outer or extended border (see Section 6.2.3). If inner borders are used, some properties are not satisfied—e.g., the perimeter of a 1-pixel region is 4 if the outer boundary is used, and 1 if the inner is used.

Curvature

In the continuous case, curvature is defined as the rate of change of slope. In discrete space, the curvature description must be slightly modified to overcome difficulties resulting from violation of curve smoothness. The curvature scalar descriptor (also called boundary straightness) finds the ratio between the total number of boundary pixels (length) and the number of boundary pixels where the boundary direction changes significantly. The smaller the number of direction changes, the straighter the boundary. The evaluation algorithm is based on the detection of angles between line segments positioned b boundary pixels from the evaluated boundary pixel in both directions. The angle need not be represented numerically; rather, relative position of line segments can be used as a property. The parameter b determines sensitivity to local changes of the boundary direction (Figure 8.7). Curvature computed from the chain code can be found in [Rosenfeld, 1974], and the tangential border representation is also suitable for curvature computation. Values of the curvature at all boundary pixels can be represented by a histogram; relative numbers then provide information on how common specific boundary direction changes are. Histograms of boundary angles, such as the β angle in Figure 8.7, can be built in a similar way—such histograms can be used for region description. Another approach to calculating curvature from digital curves is based on convolution with the truncated Gaussian kernel [Lowe, 1989].

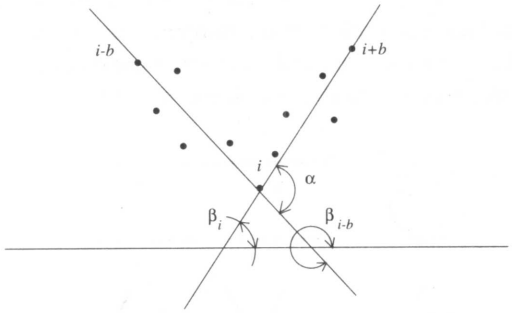


Figure 8.7: Curvature.

Bending energy

The bending energy (BE) of a border (curve) may be understood as the energy necessary to bend a rod to the desired shape, and can be computed as a sum of squares of the border curvature $c(k)$ over the border length L.

$$\mathrm{BE} = \frac{1}{L}\sum_{k=1}^{L} c^2(k) \tag{8.1}$$

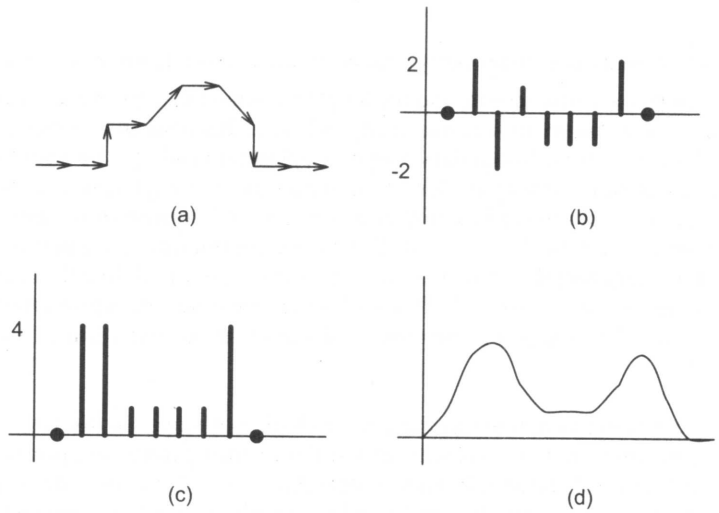


Figure 8.8: Bending energy. (a) Chain code 0, 0, 2, 0, 1, 0, 7, 6, 0, 0. (b) Curvature 0, 2, -2, 1, -1, -1, -1, 2, 0. (c) Sum of squares gives the bending energy. (d) Smoothed version.

Bending energy can easily be computed from Fourier descriptors using Parseval's theorem [Oppenheim et al., 1983; Papoulis, 1991]. To represent the border, Freeman's chain code or its smoothed version may be used; see Figure 8.8. Bending energy does not permit shape reconstruction.

Signature

The signature of a region may be obtained as a sequence of normal contour distances. The normal contour distance is calculated for each boundary element as a function of the path length. For each border point A, the shortest distance to an opposite border point B is sought in a direction perpendicular to the border tangent at point A; see Figure 8.9. Note that *being opposite* is not a symmetric relation (compare Algorithm 6.16). Signatures are noise sensitive, and using smoothed signatures or signatures of smoothed contours reduces noise sensitivity. Signatures may be applied to the recognition of overlapping objects or whenever only partial contours are available [Vernon, 1987]. Position, rotation, and scale-invariant modifications based on gradient-perimeter and angle-perimeter plots are discussed in [Safaee-Rad et al., 1989].

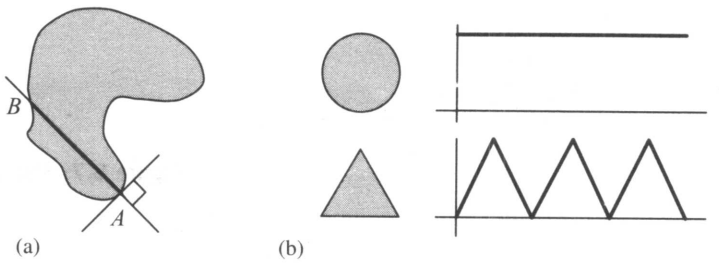


Figure 8.9: Signature. (a) Construction. (b) Signatures for a circle and a triangle.

Chord distribution

A line joining any two points of the region boundary is a chord, and the distribution of lengths and angles of all chords on a contour may be used for shape description. Let $b(x, y) = 1$ represent the contour points, and $b(x, y) = 0$ represent all other points. The chord distribution can be computed (see Figure 8.10a) as

$$h(\Delta x, \Delta y) = \iint b(x, y)\, b(x + \Delta x, y)\, dx\, dy \tag{8.2}$$

or in digital images as

$$h(\Delta x, \Delta y) = \sum_i \sum_j b(i, j) b(i + \Delta x, j + \Delta y). \tag{8.3}$$

To obtain the rotation-independent radial distribution $h_r(r)$, the integral over all angles is computed (Figure 8.10b).

$$h_r(r) = \int_{-\pi/2}^{\pi/2} h(\Delta x, \Delta y)\, r\, \mathrm{d}\theta, \tag{8.4}$$

where $r = \sqrt{\Delta x^2 + \Delta y^2}$, $\theta = \sin^{-1}(\Delta y / r)$. The distribution $h_r(r)$ varies linearly with scale. The angular distribution $h_a(\theta)$ is independent of scale, while rotation causes a proportional offset.

$$h_a(\theta) = \int_0^{\max(r)} h(\Delta x, \Delta y)\, \mathrm{d}r. \tag{8.5}$$

Combination of both distributions gives a robust shape descriptor [Smith and Jain, 1982; Cooteset al., 1992].

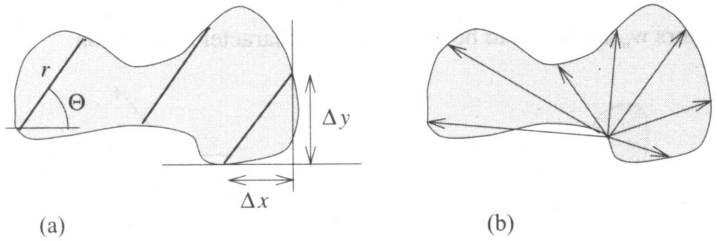


Figure 8.10: Chord distribution

8.2.3 Fourier transforms of boundaries

Suppose C is a closed curve (boundary) in the complex plane (Figure 8.11a). Traveling anti-clockwise along this curve keeping constant speed, a complex function $z(t)$ is obtained, where t is a time variable. The speed should be chosen such that one circumnavigation of the boundary takes time 2π; then a periodic function with period 2π is obtained after multiple passes around the curve. This permits a Fourier representation of $z(t)$ (see Section 3.2.4),

$$z(t) = \sum_n T_n e^{\mathrm{int}}. \tag{8.6}$$

The coefficients T_n of the series are called the **Fourier descriptors** of the curve C. It is more useful to consider the curve distance s in comparison to time

$$t = 2\pi s / L, \tag{8.7}$$

where L is the curve length. The Fourier descriptors T_n are given by

$$T_n = \frac{1}{L} \int_0^L z(s)\, e^{-i(2\pi/L)ns}\, \mathrm{d}s. \tag{8.8}$$

The descriptors are influenced by the curve shape and by the initial point of the curve. Working with digital image data, boundary co-ordinates are discrete and the function $z(s)$ is not continuous. Assume that $z(k)$ is a discrete version of $z(s)$, where 4-connectivity is used to get a constant sampling interval; the descriptors T_n can be computed from the discrete Fourier transform (DFT, Section 3.2) of $z(k)$

$$z(k) \longleftarrow \mathrm{DFT} \longrightarrow T_n. \tag{8.9}$$

The Fourier descriptors can be invariant to translation and rotation if the co-ordinate system is appropriately chosen [Pavlidis, 1977; Persoon and Fu, 1977; Wallace and Wintz, 1980; Grimmins, 1982; Lin and Chellappa, 1987]. They have been used for handwritten alphanumeric character description in [Shridhar and Badreldin, 1984]; the character boundary in this description was represented by co-ordinate pairs (x_m, y_m) in 4-connectivity, $(x_i, y_1) = (x_L, y_L)$. Then

$$a_n = \frac{1}{L-1} \sum_{m=1}^{L-1} x_m \, e^{-i[2\pi/(L-1)]nm}, \tag{8.10}$$

$$b_n = \frac{1}{L-1} \sum_{m=1}^{L-1} y_m \, e^{-i[2\pi/(L-1)]nm}. \tag{8.11}$$

The coefficients a_n, b_n are not invariant but after the transform

$$r_n = \left(|a_n|^2 + |b_n|^2\right)^{1/2}, \tag{8.12}$$

r_n are translation and rotation invariant. To achieve a magnification in variance the descriptors w_n are used

$$w_n = r_n / r_1. \tag{8.13}$$

The first 10-15 descriptors w_n are found to be sufficient for character description.

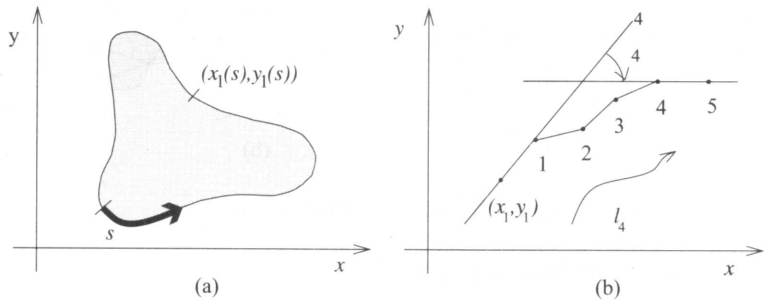


Figure 8.11: Fourier description of boundaries. (a) Descriptors T_n. (b) Descriptors S_n.

A closed boundary can be represented as a function of angle tangents versus the distance between the boundary points from which the angles were determined (Figure 8.11b). Let φ_k be the angle measured at the k^{th} boundary point, and let l_k be the distance between the boundary starting point and the k^{th} boundary point. A periodic function can be defined

$$a(l_k) = \varphi_k + u_k, \tag{8.14}$$

$$u_k = 2\pi \, l_k / L. \tag{8.15}$$

The descriptor set is then

$$S_n = \frac{1}{2\pi} \int_0^{2\pi} a(u) \, e^{-inu} \, du. \tag{8.16}$$

The discrete Fourier transform is used in all practical applications [Pavlidis, 1977].

The high-quality boundary shape representation obtained using only a few lower-order coefficients is a favorable property common to Fourier descriptors. We can compare the results of using the S_n and T_n descriptors: The S_n descriptors have more high-frequency components present in the boundary function due to more significant changes of tangent angles, and as a result, they do not decrease as fast as the T_n descriptors. In addition, the S_n descriptors are not suitable for boundary reconstruction since they often result in a non-closed boundary. A method for obtaining a closed boundary using S_n descriptors is given in [Strackee and Nagelkerke, 1983]. The T_n descriptor values decrease quickly for higher frequencies, and their reconstruction always results in a closed boundary. Moreover, the S_n descriptors cannot be applied for squares, equilateral triangles, etc. [Wallace, 1981] unless the solution methods introduced in [Wallace and Wintz, 1980] are applied.

Fourier descriptors can also be used for calculation of region area, location of centroid, and computation of second-order moments [Kiryati and Maydan, 1989]. Fourier descriptors are a general technique, but problems with describing local information exist. A modified technique using a combined frequency-position space that deals better with local curve properties exists, another modification that is invariant under rotation, translation, scale, mirror reflection, and shifts in starting points is discussed in [Krzyzak et al., 1989]. Conventional Fourier descriptors cannot be used for recognition of occluded objects. Nevertheless, classification of partial shapes using Fourier descriptors is introduced in [Lin and Chellappa, 1987]. Boundary detection and description using elliptic Fourier decomposition of the boundary is described in [Staib and Duncan, 1992].

8.2.4 Boundary description using segment sequences

Representation of a boundary using **segments** with specified properties is another option for boundary (and curve) description. If the segment type is known for all segments, the boundary can be described as a chain of segment types, a code word consisting of representatives of a type alphabet. An example is given in Figure 8.14 which will be discussed later in more detail. This sort of description is suitable for syntactic recognition (see Section 9.4). A trivial segment chain is used to obtain the Freeman code description discussed in Section 8.2.1.

A **polygonal representation** approximates a region by a polygon, the region being represented using its vertices. Polygonal representations are obtained as a result of a simple boundary segmentation. The boundary can be approximated with varying precision; if a more precise description is necessary, a larger number of line segments may be employed. Any two boundary points $\mathbf{x}_1$, $\mathbf{x}_2$ define a line segment, and a sequence of points $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ represents a chain of line segments—from the point $\mathbf{x}_1$ to the point $\mathbf{x}_2$, and from $\mathbf{x}_2$ to $\mathbf{x}_3$. If $\mathbf{x}_1 = \mathbf{x}_3$, a closed boundary results. There are many types of straight-segment boundary representations [Pavlidis, 1977; Lindenbaum and Bruckstein, 1993]; the problem lies in determining the location of boundary vertices, one solution to which is to apply a split-and-merge algorithm. The merging step consists of going through a set of boundary points and adding them to a straight segment as long as a segment straightness criterion is satisfied. If the straightness characteristic of the segment is lost, the last connected point is marked as a vertex and construction of a new straight segment begins. This general approach has many variations, some of which are described in [Pavlidis, 1977].

Boundary vertices can be detected as boundary points with a significant change of boundary direction using the curvature (boundary straightness) criterion (see Section 8.2.2). This approach works well for boundaries with rectilinear boundary segments.

Another method for determining the boundary vertices is a **tolerance interval approach** based on setting a maximum allowed difference e. Assume that point $\mathbf{x}_1$ is the end point of a previous segment and so by definition the first point of a new segment. Define points $\mathbf{x}_2$, $\mathbf{x}_3$ positioned a distance e from the point $\mathbf{x}_1$ to be rectilinear—$\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ are positioned on a straight line—see Figure 8.12. The next step is to locate a segment which can fit between parallels directed from points $\mathbf{x}_2$ and $\mathbf{x}_3$. Resulting segments are sub-optimal, although optimality can be achieved with a substantial increase in computational effort [Tomek, 1974].

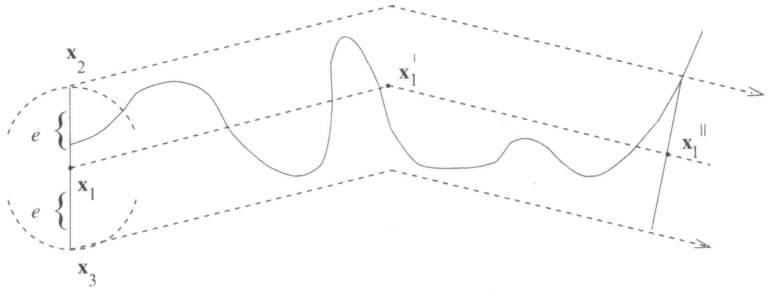


Figure 8.12: Tolerance interval.

The methods introduced above represent single-pass algorithms of boundary segmentation using a segment-growing approach. Often they do not result in the best possible boundary segmentation because the vertex which is located often indicates that the real vertex should have been located a few steps back. The splitting approach of segmenting boundaries into smaller segments can sometimes help, and the best results can be anticipated using a combination of both methods. If the splitting approach is used, segments are usually divided into two new, smaller segments until the new segments meet the final requirements [Duda and Hart, 1973; Pavlidis, 1977]. A simple procedure for splitting begins from end points $\mathbf{x}_1$ and $\mathbf{x}_2$ of a curve; these end points are connected by a line segment. The next step searches all the curve points for the curve point x_3 with the largest distance from the line segment. If the point located is within a preset distance between itself and the line segment, the segment x_1–x_2 is an end segment and all curve vertices are found, the curve being represented polygonally by vertices x_1 and x_2. Otherwise the point x_3 is set as a new vertex and the process is applied recursively to both resulting segments x_1–x_3 and x_3–x_2 (see Figure 8.13 and Section 6.2.7).

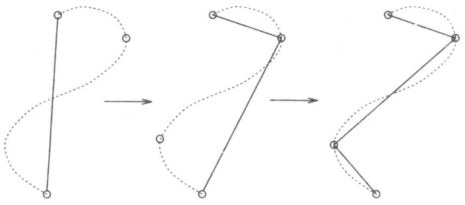


Figure 8.13: Recursive boundary splitting.

Boundary segmentation into segments of **constant curvature** is another possibility for boundary representation. The boundary may also be split into segments which can be represented by polynomials, usually of second order, such as circular, elliptic, or parabolic segments [Costabile et al., 1985; Wuescher and Boyer, 1991]. Curve segmentation into circular arcs and straight lines is presented in [Rosin and West, 1989]. presented in [Rosin and West, 1989]. Segments are considered as primitives for syntactic shape recognition procedures—a typical example is the syntactic description and recognition of chromosomes [Fu, 1974], where boundary segments are classified as convex segments of large curvature, concave segments of large curvature, straight segments, etc., as illustrated in Figure 8.14.

Other syntactic object recognition methods based on a contour partitioning into primitives from a specified set are described in [Jakubowski, 1990]. Partitioning of the contour using location of points with high positive curvatures (corners) is described in [Chien and Aggarwal, 1989], together with applications to occluded contours. A discrete curvature function based on a chain code representation of a boundary is used with a morphological approach to obtain segments of constant curvature in [Leymarie and Levine, 1989]. Contour partitioning using segments of constant intensity is suggested in [Marshall, 1989], and polygonal representation used in a *hypothesize and verify* approach to recognition of occluded objects may be found in [Koch and Kashyap, 1987].

Sensitivity of shape descriptors to scale (image resolution) has already been mentioned as an undesirable feature of a majority of descriptors. In other words, shape description varies with scale, and different results

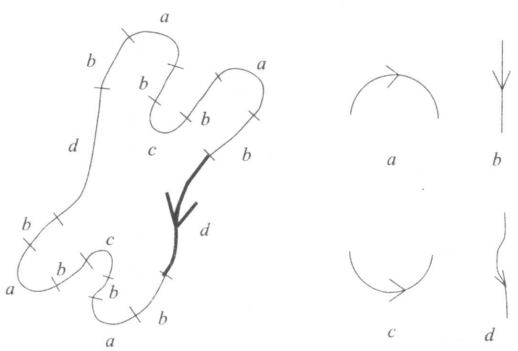


Figure 8.14: Structural description of chromosomes by a chain of boundary segments, code word: d, b, a, b, c, b, a, b, d, b, a, b, c, b, a, b. *Adapted from [Fu, 1974].*