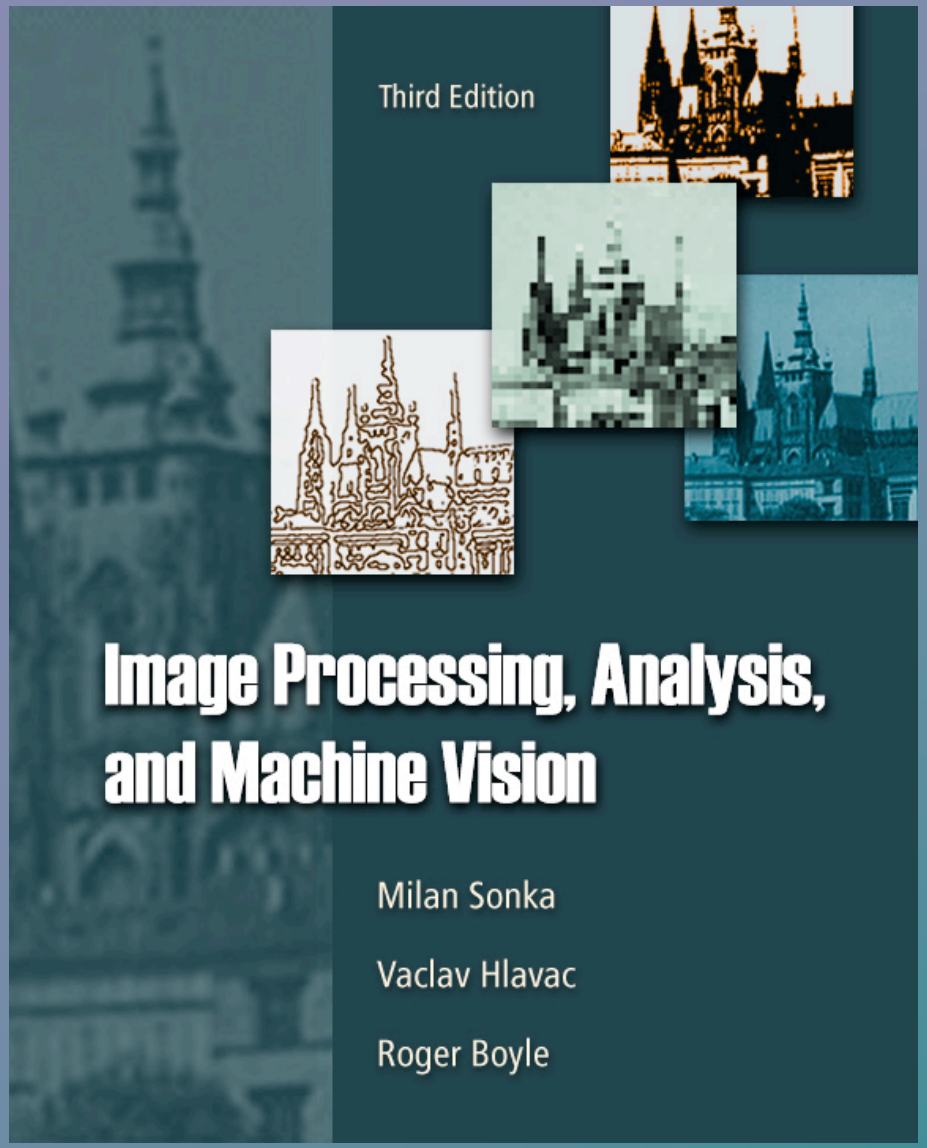


Chapter 6 – Segmentation I

6.1 Thresholding

6.2 Edge-based segmentation

6.3 Region-based segmentation

6.4 Matching

6.5 Evaluation issues in segmentation

Image segmentation

- The main goal of image segmentation is to divide an image into parts that have a strong correlation with objects or areas of the real world depicted in the image.
- Segmentation methods can be divided into three groups: **thresholding**, **edgebased** segmentation and **region-based** segmentation.
- Each region can be represented by its closed boundary, and each closed boundary describes a region.
- Image data ambiguity is one of the main segmentation problems, often accompanied by information noise.
- The more a priori information is available to the segmentation process, the better the segmentation results that can be obtained.

Algorithm 6.1: Basic thresholding

1. Search all the pixels $f(i,j)$ of the image f. An image element $g(i,j)$ of the segmented image is an object pixel if $f(i,j) \geq T$, and is a background pixel otherwise.

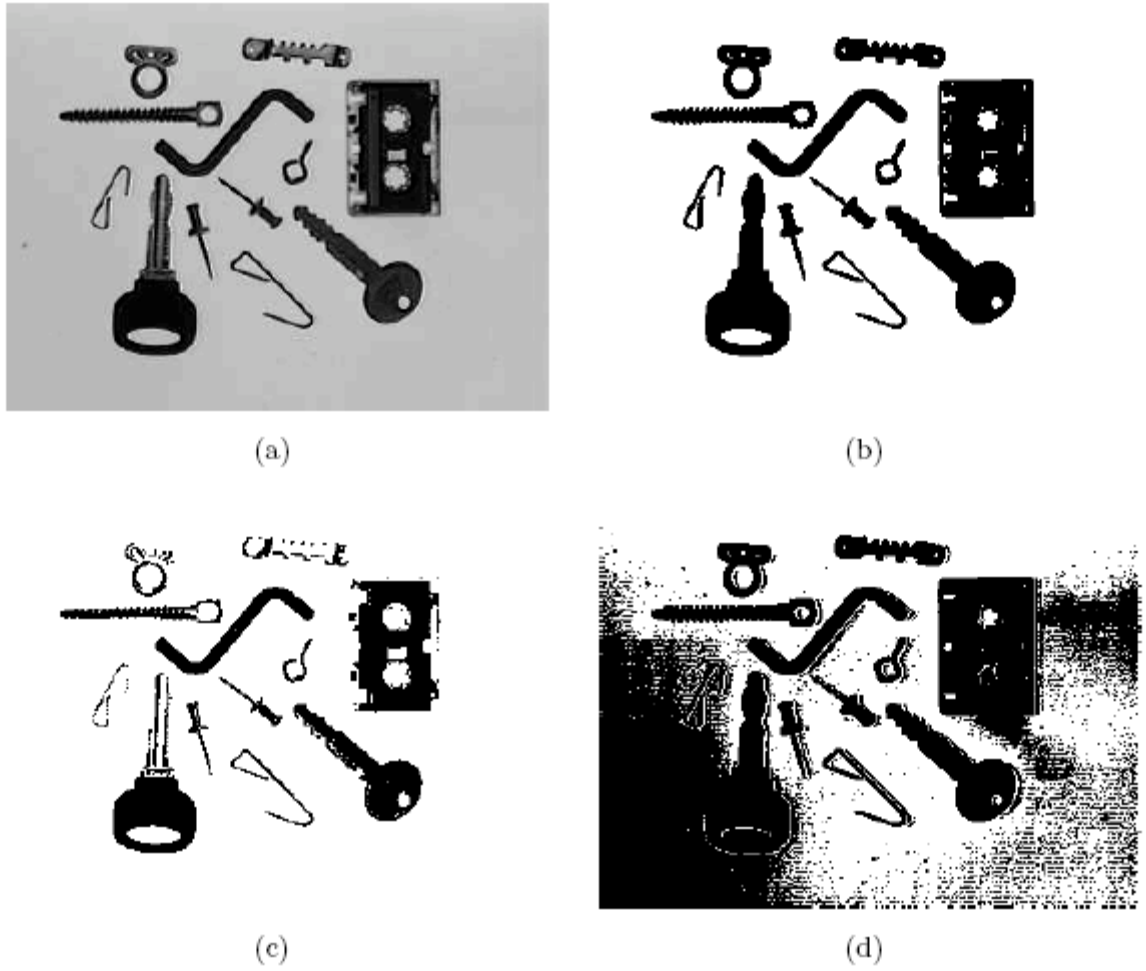
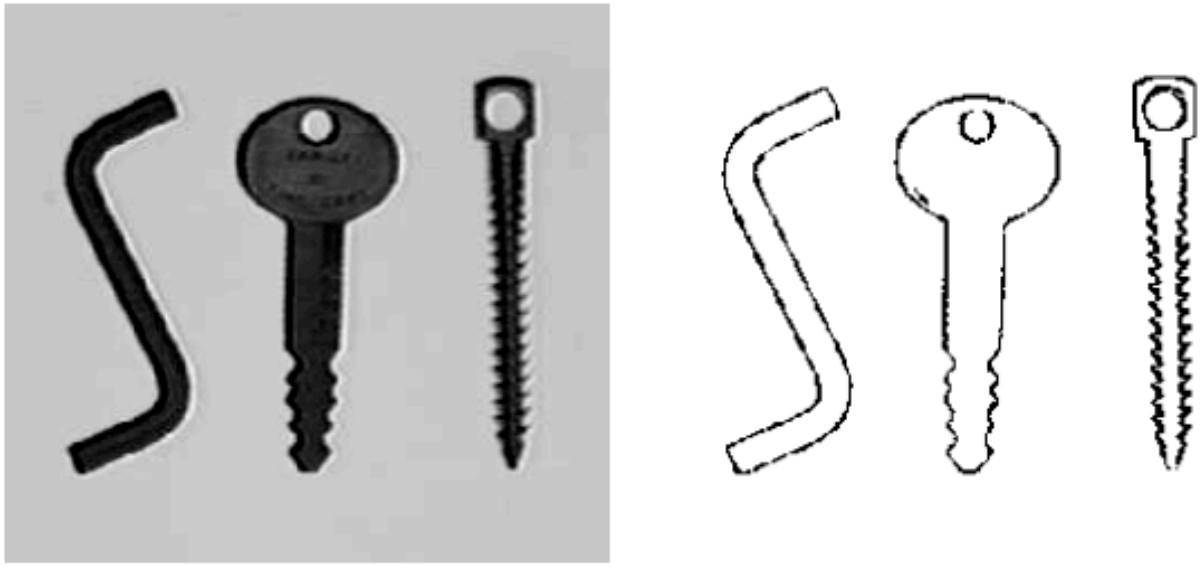


Figure 6.1: Image thresholding. (a) Original image. (b) Threshold segmentation. (c) Threshold too low. (d) Threshold too high.



(a) (b)

Figure 6.2: Image thresholding modification. (a) Original image. (b) Border detection using band-thresholding.

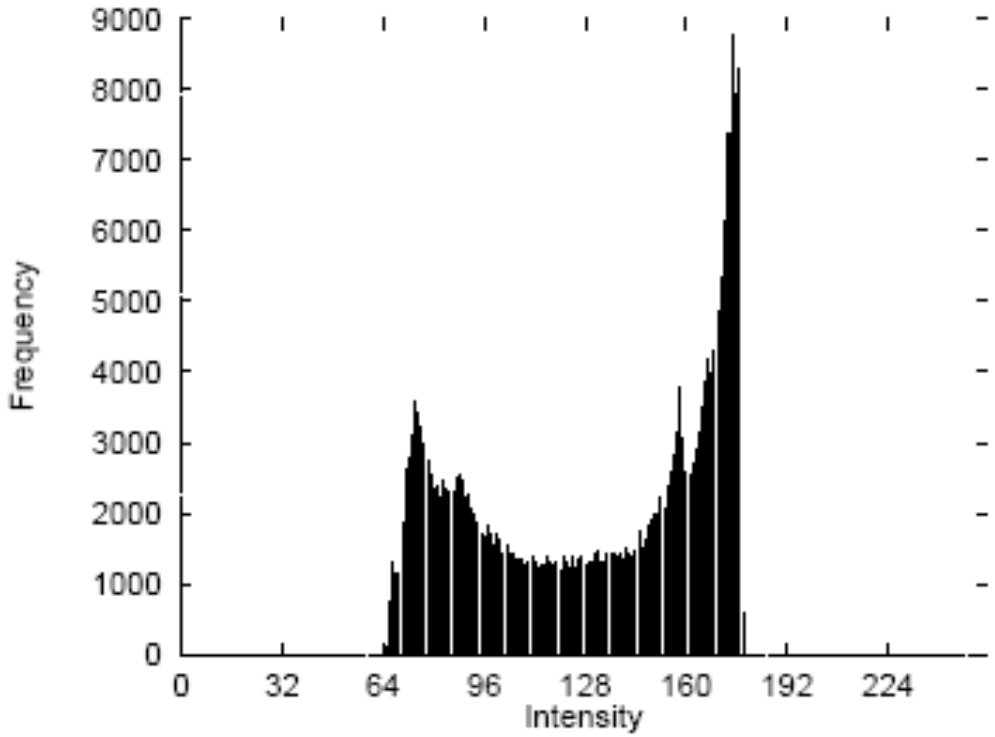


Figure 6.3: A bi-modal histogram.

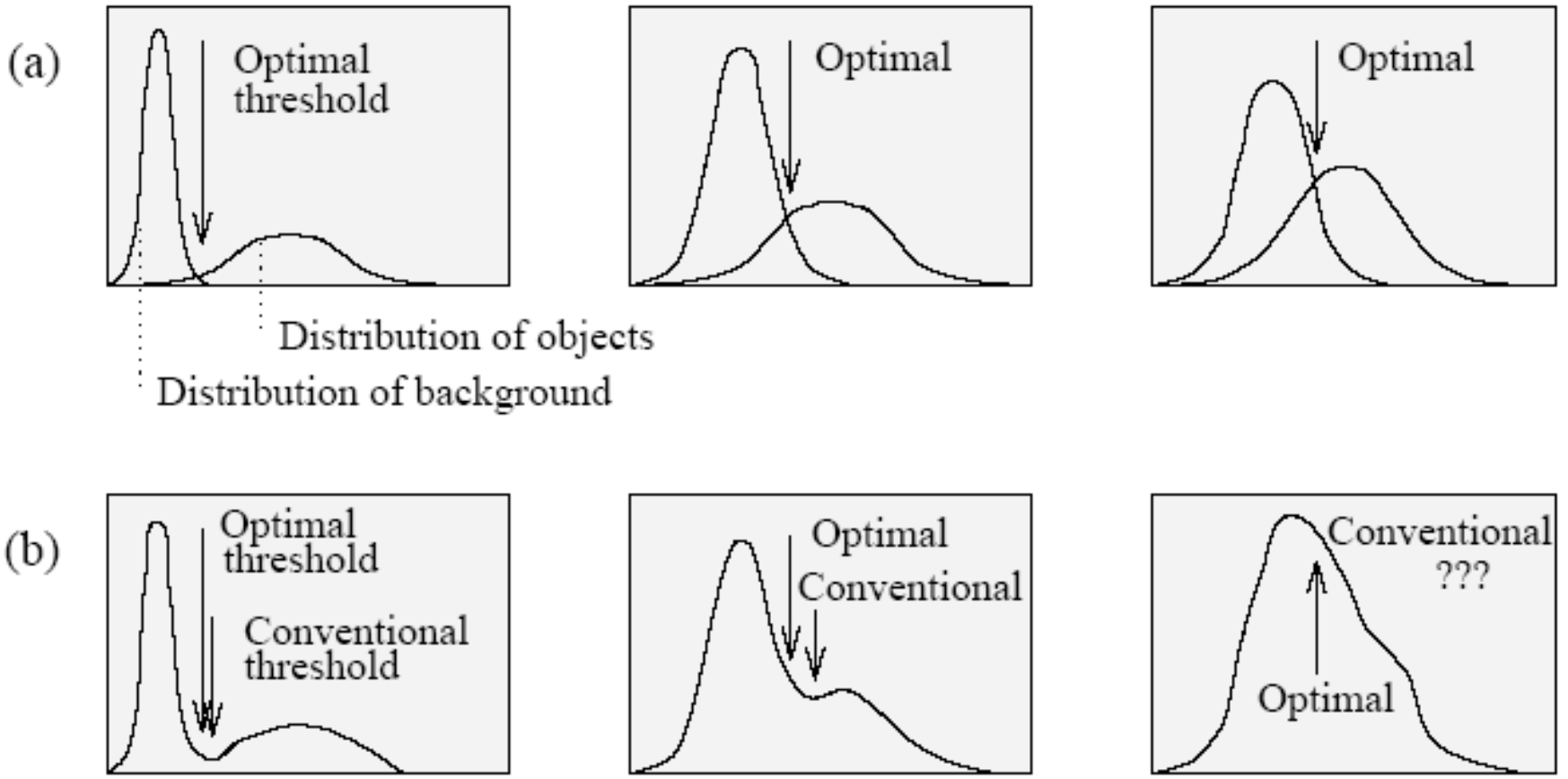


Figure 6.4: Gray-level histograms approximated by two normal distributions—the threshold is set to give minimum probability of segmentation error. (a) Probability distributions of background and objects. (b) Corresponding histograms and optimal threshold.

Algorithm 6.2: Iterative (optimal) threshold selection

1. Assuming no knowledge about the exact location of objects, consider as a first approximation that the four corners of the image contain background pixels only and the remainder contains object pixels.
2. At step t, compute μ_B^t and μ_O^t as the mean background and object gray-level, respectively, where segmentation into background and objects at step t is defined by the threshold value T^t determined in the previous step [equation 6.9]

$$\mu_B^t = \frac{\sum_{(i,j)\in \text{background}} f(i,j)}{\#\text{background\_pixels}}, \qquad \mu_O^t = \frac{\sum_{(i,j)\in \text{objects}} f(i,j)}{\#\text{object\_pixels}}. \tag{6.8}$$

3. Set

$$T^{(t+1)} = \frac{\mu_B^t + \mu_O^t}{2}, \tag{6.9}$$

   $T^{(t+1)}$ now provides an updated background—object distinction.
4. If $T^{(t+1)} = T^{(t)}$, halt; otherwise return to step 2.

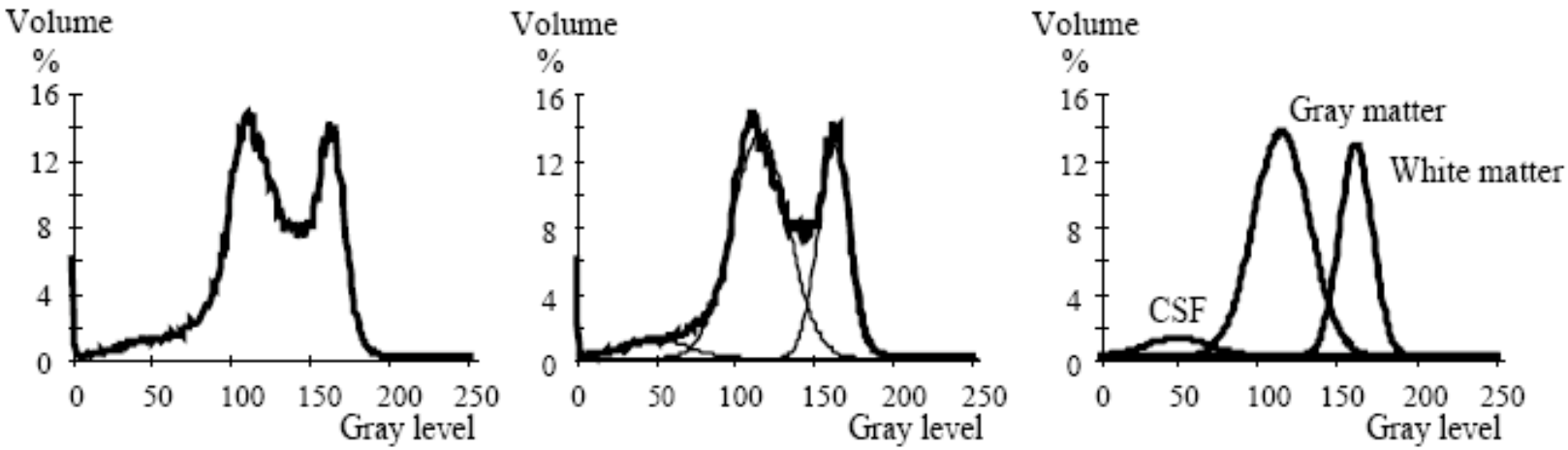


Figure 6.5: Segmentation of 3D T1-weighted MR brain image data using optimal thresholding. (a) Local gray-level histogram. (b) Fitted Gaussian distributions, global 3D image fit. (c) Gaussian distributions corresponding to WM, GM, and CSF. *Courtesy of R. J. Frank, T. J. Grabowski, The University of Iowa.*

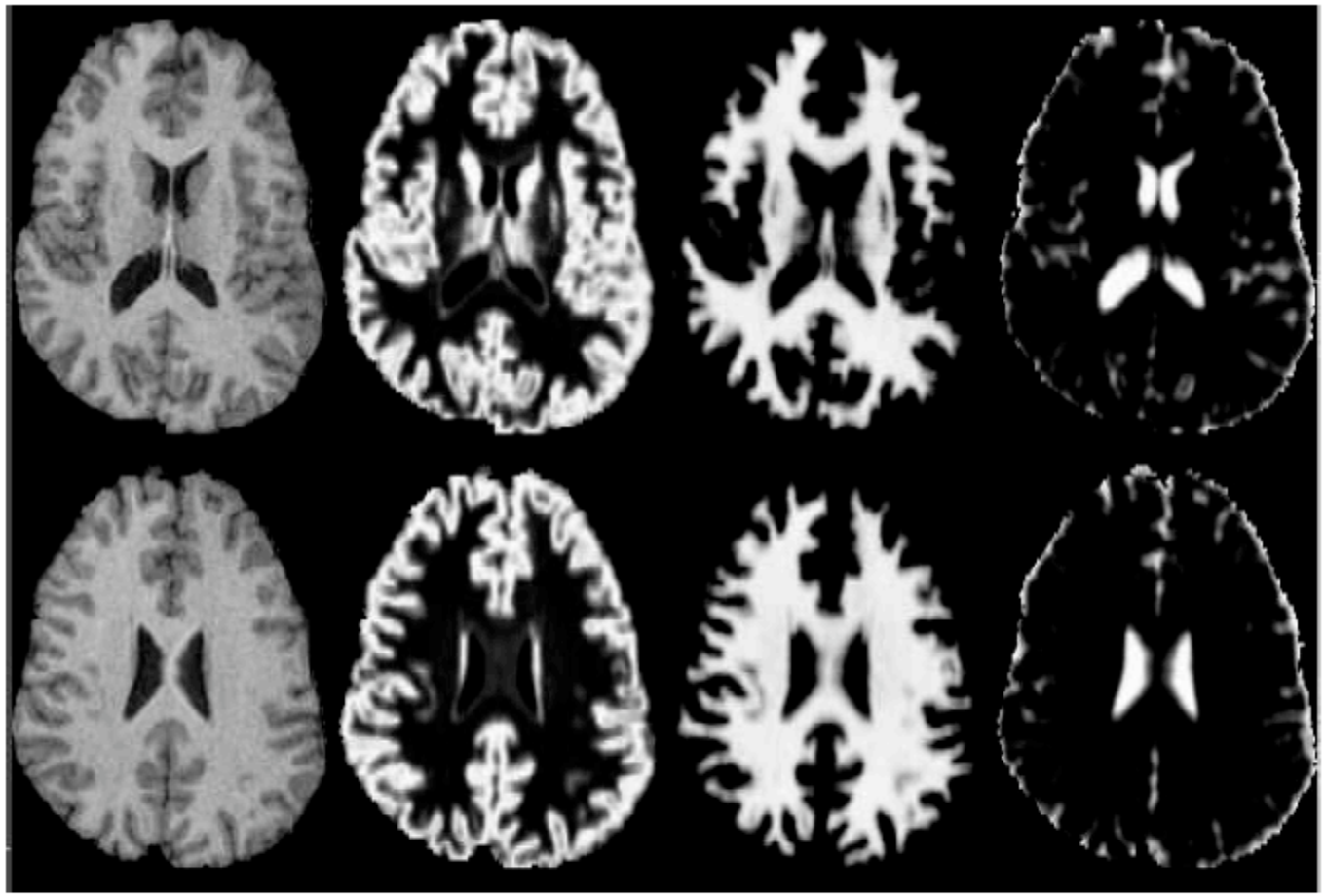


Figure 6.6: Optimal MR brain image segmentation. Left column: original T1-weighted MR images, two of 120 slices of the 3D volume. Middle left: Partial-volume maps of gray matter. The brighter the voxel, the higher is the partial volume percentage of gray matter in the voxel. Middle right: Partial-volume maps of white matter. Right column: Partial-volume maps of cerebro-spinal fluid. *Courtesy of R. J. Frank, T. J. Grabowski, The University of Iowa.*

Algorithm 6.3: Recursive multi-spectral thresholding

1. Initialize the whole image as a single region.
2. Compute a smoothed histogram (see Section 2.3.2) for each spectral band. Find the most significant peak in each histogram and determine two thresholds as local minima on either side of this maximum. Segment each region in each spectral band into sub-regions according to these thresholds. Each segmentation in each spectral band is projected into a multi-spectral segmentation—see Figure 6.7. Regions for the next processing steps are those in the multi-spectral image.
3. Repeat step 2 for each region of the image until each region's histogram contains only one significant peak.

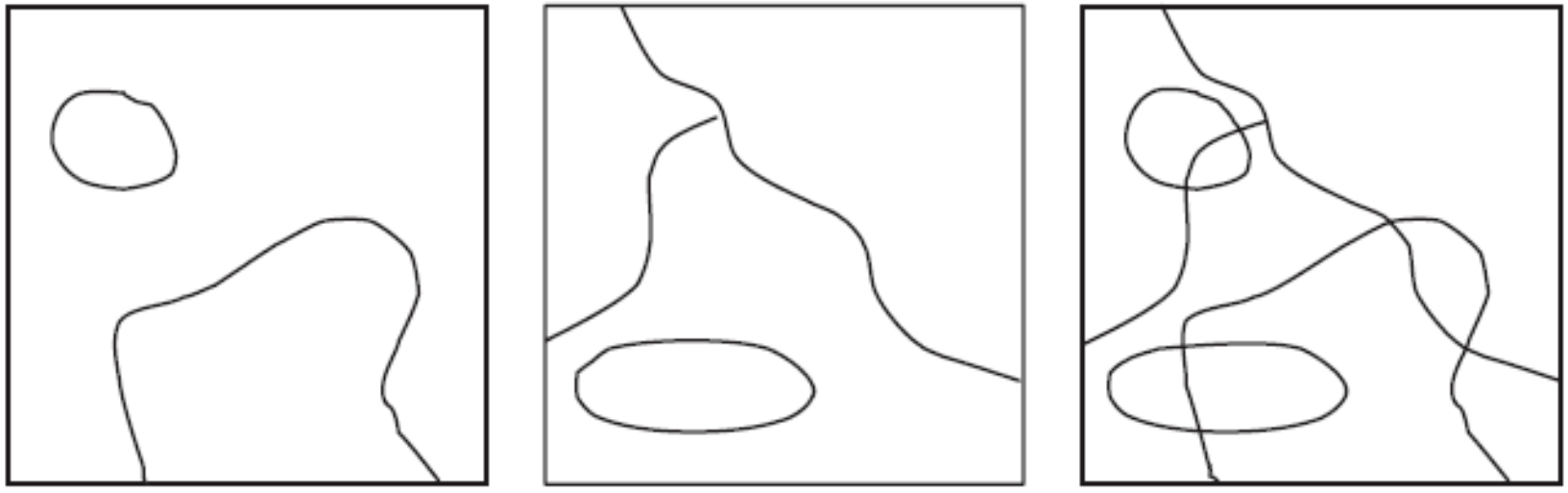


Figure 6.7: Recursive multi-spectral thresholding. (a) Band 1 thresholding. (b) Band 2 thresholding. (c) Multi-spectral segmentation.

Summary

Thresholding

- Thresholding represents the simplest image segmentation process, and it is computationally inexpensive and fast. A brightness constant called a **threshold** is used to segment objects and background.
- Single thresholds can either be applied to the entire image (**global threshold**) or can vary in image parts (**local threshold**). Only under very unusual circumstances can thresholding be successful using a single threshold for the whole image.
- Many modifications of thresholding exist: **local thresholding**, **band thresholding**, **semi-thresholding**, **multi-thresholding**, etc.

Summary

Thresholding cont.

- **Threshold detection** methods are used to determine the threshold automatically. If some property of an image after segmentation is known a priori, the task of threshold detection is simplified, since the threshold can be selected to ensure that this property is satisfied. Threshold detection can use p-**tile thresholding**, **histogram shape analysis**, **optimal thresholding**, etc.
- In **bi-modal histograms**, the threshold can be determined as a minimum between the two highest local maxima.
- **Optimal thresholding** determines the threshold as the closest gray-level corresponding to the minimum probability between the maxima of two or more normal distributions. Such thresholding results in minimum error segmentation.
- **Multi-spectral thresholding** is appropriate for color or multi-band images.

Non-maximum suppression

2. For each pixel with non-zero edge magnitude, inspect the two adjacent pixels indicated by the direction of its edge (see Figure 6.10).
3. If the edge magnitude of either of these two exceeds that of the pixel under inspection, mark it for deletion.
4. When all pixels have been inspected, re-scan the image and erase to zero all edge data marked for deletion.

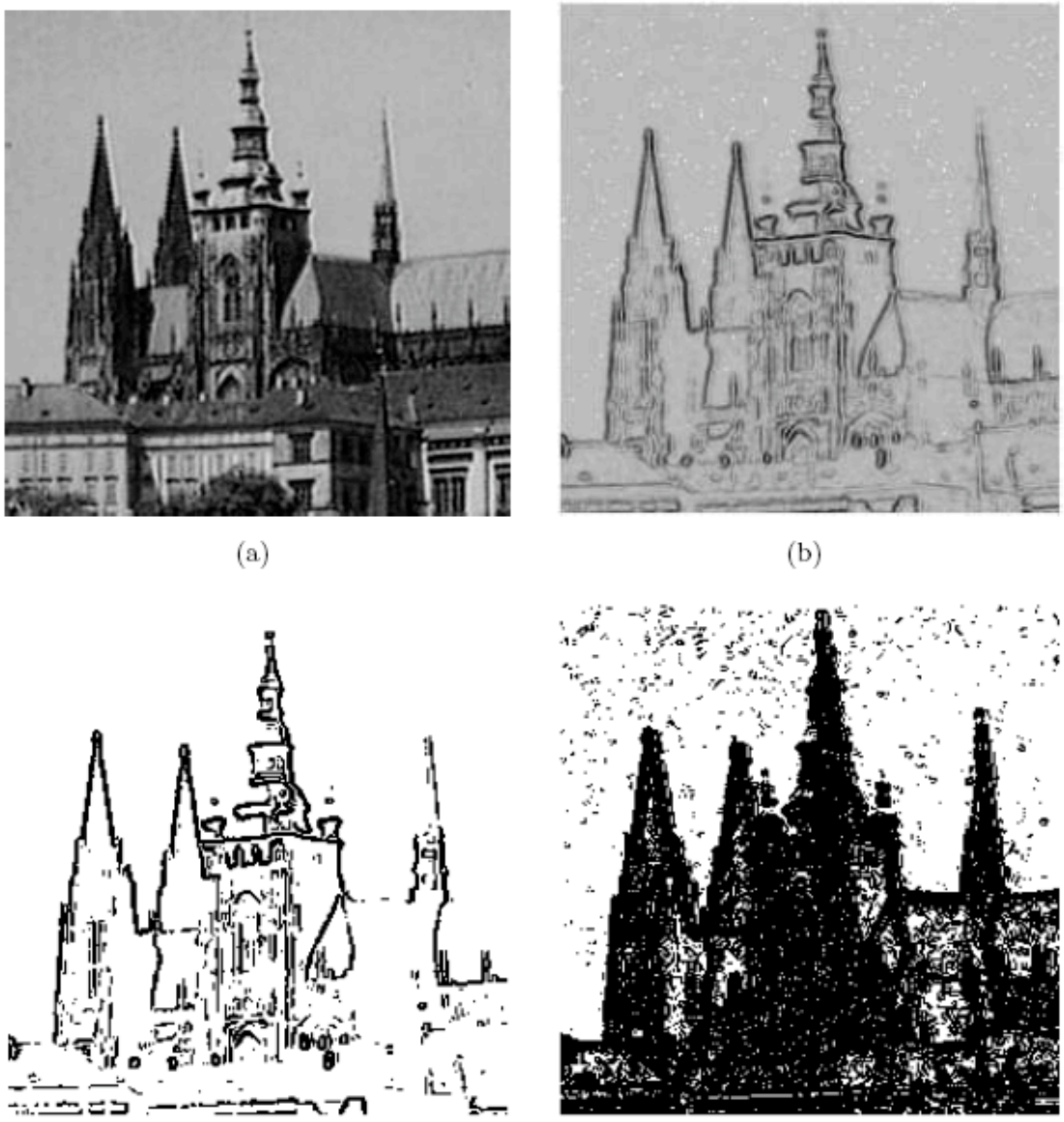
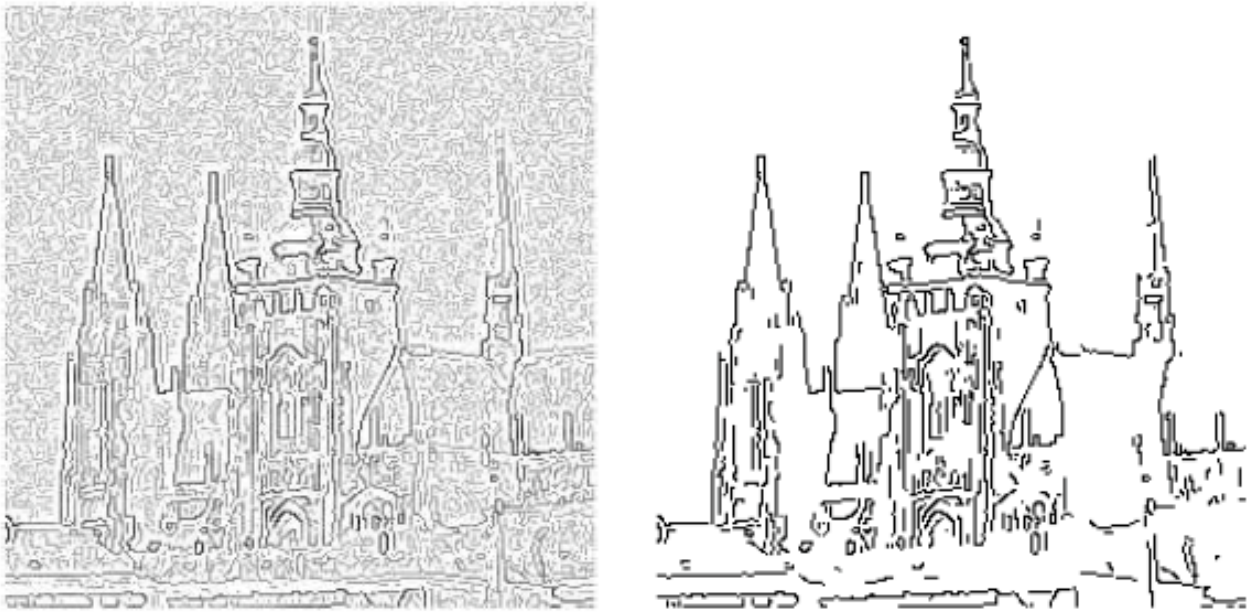


Figure 6.8: Edge image thresholding. (a) Original image. (b) Edge image (low contrast edges enhanced for display). (c) Edge image thresholded at 30. (d) Edge image thresholded at 10.



(a) (b)

Figure 6.9: (a) Non-maximal suppression of the data in Figure 6.8a. (b) hysteresis applied to (a); high threshold 70, low threshold 10.

Algorithm 6.5: Hysteresis to filter output of an edge detector

1. Mark all edges with magnitude greater than t_1 as correct.
2. Scan all pixels with edge magnitude in the range $[t_0, t_1]$.
3. If such a pixel borders another already marked as an edge, then mark it too. 'Bordering' may be defined by 4- or 8-connectivity.
4. Repeat from step 2 until stability.

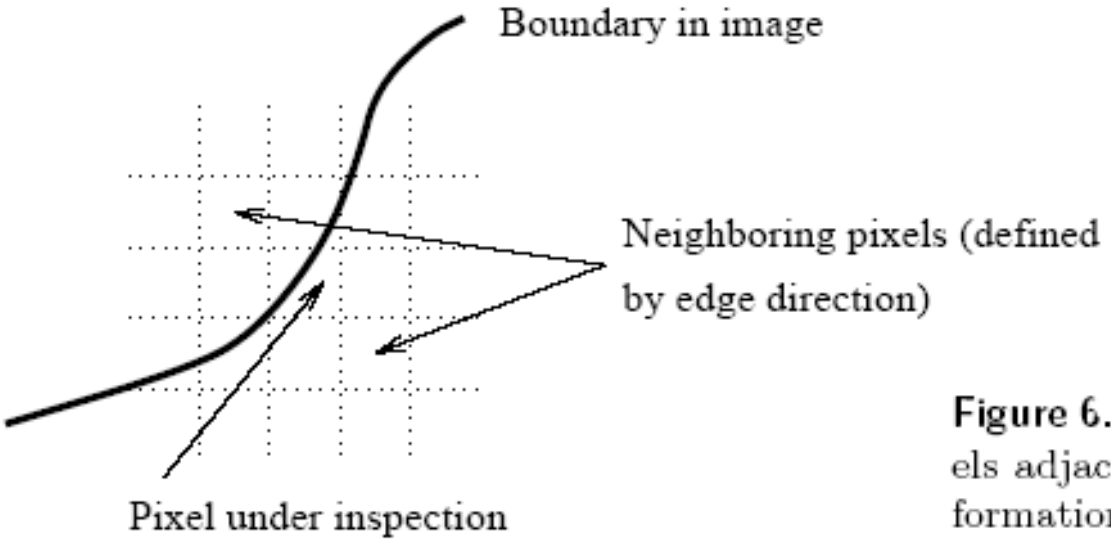


Figure 6.10: Non-maximal suppression; pixels adjacent with respect to local edge information are inspected.

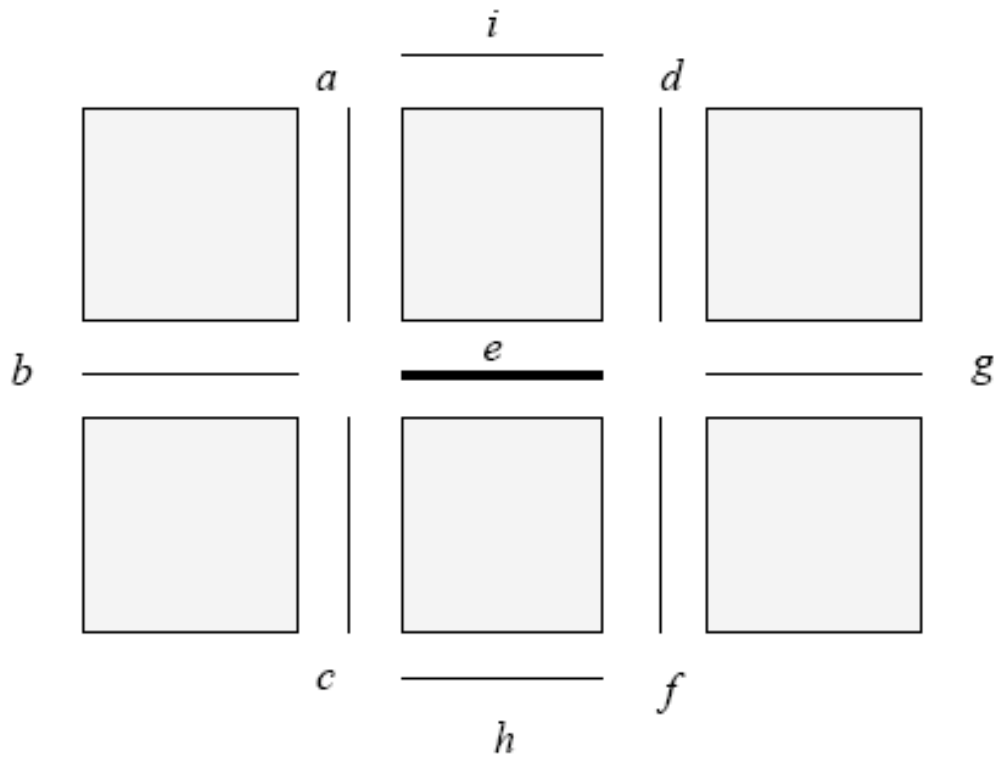


Figure 6.11: Crack edges surrounding central edge e.

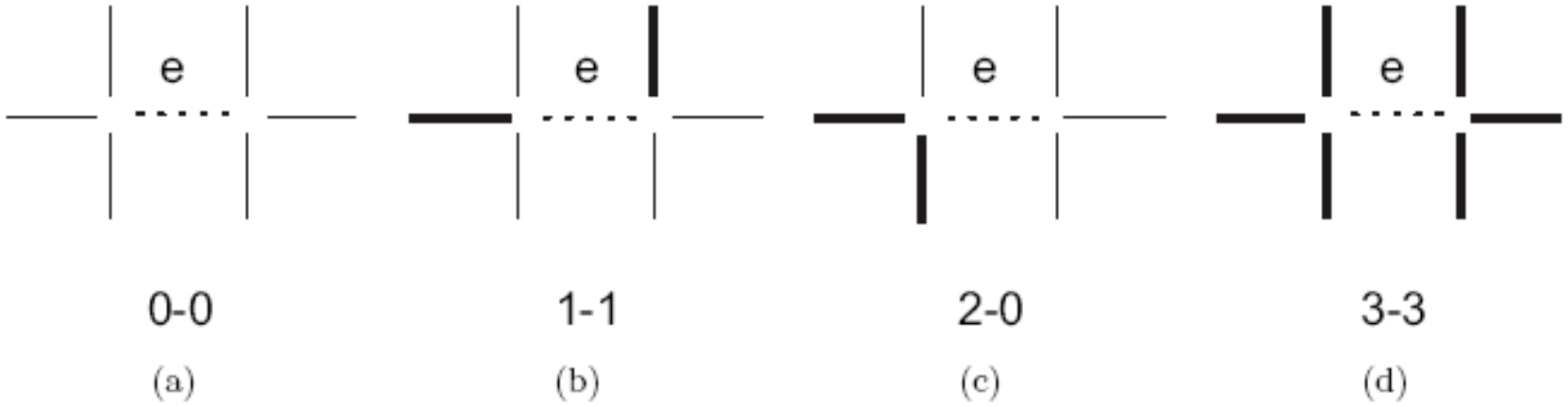


Figure 6.12: Edge patterns and corresponding edge types.

Algorithm 6.6: Edge relaxation

1. Evaluate a confidence $c^{(1)}(e)$ for all crack edges e in the image.
2. Find the edge type of each edge based on edge confidences $c^{(k)}(e)$ in its neighborhood.
3. Update the confidence $c^{(k+1)}(e)$ of each edge e according to its type and its previous confidence $c^{(k)}(e)$.
4. Stop if all edge confidences have converged either to 0 or 1. Repeat steps 2 and 3 otherwise.

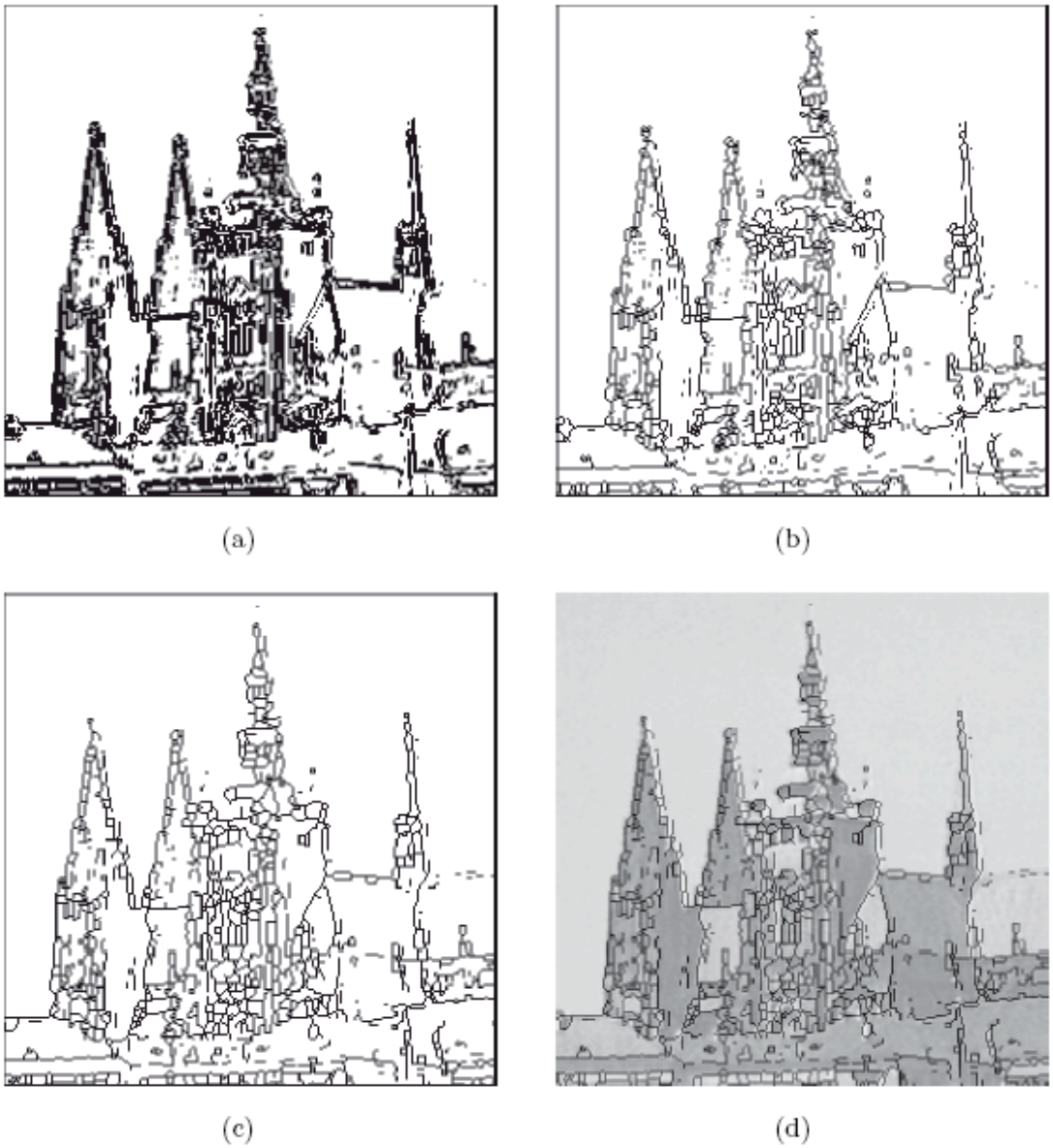


Figure 6.13: Edge relaxation, see Figure 3.11a for original. (a) Resulting borders after 10 iterations. (b) Borders after thinning. (c) Borders after 100 iterations, thinned. (d) Borders after 100 iterations overlaid over original.

Algorithm 6.7: Inner boundary tracing

1. Search the image from top left until a pixel of a new region is found; this pixel P_0 then has the minimum column value of all pixels of that region having the minimum row value. Pixel P_0 is a starting pixel of the region border. Define a variable *dir* which stores the direction of the previous move along the border from the previous border element to the current border element. Assign
   (a) $dir = 3$ if the border is detected in 4-connectivity (Figure 6.14a),
   (b) $dir = 7$ if the border is detected in 8-connectivity (Figure 6.14b).
2. Search the 3×3 neighborhood of the current pixel in an anti-clockwise direction, beginning the neighborhood search in the pixel positioned in the direction
   (a) $(dir + 3) \bmod 4$ (Figure 6.14c),
   (b) $(dir + 7) \bmod 8$ if *dir* is *even* (Figure 6.14d),
   $(dir + 6) \bmod 8$ if *dir* is *odd* (Figure 6.14e).

   The first pixel found with the same value as the current pixel is a new boundary element P_n. Update the *dir* value.
3. If the current boundary element P_n is equal to the second border element P_1, and if the previous border element P_{n-1} is equal to P_0, stop. Otherwise repeat step 2.
4. The detected inner border is represented by pixels $P_0 \ldots P_{n-2}$.

Algorithm 6.8: Outer boundary tracing

1. Trace the inner region boundary in 4-connectivity until done.
2. The outer boundary consists of all non-region pixels that were tested during the search process; if some pixels were tested more than once, they are listed more than once in the outer boundary list.

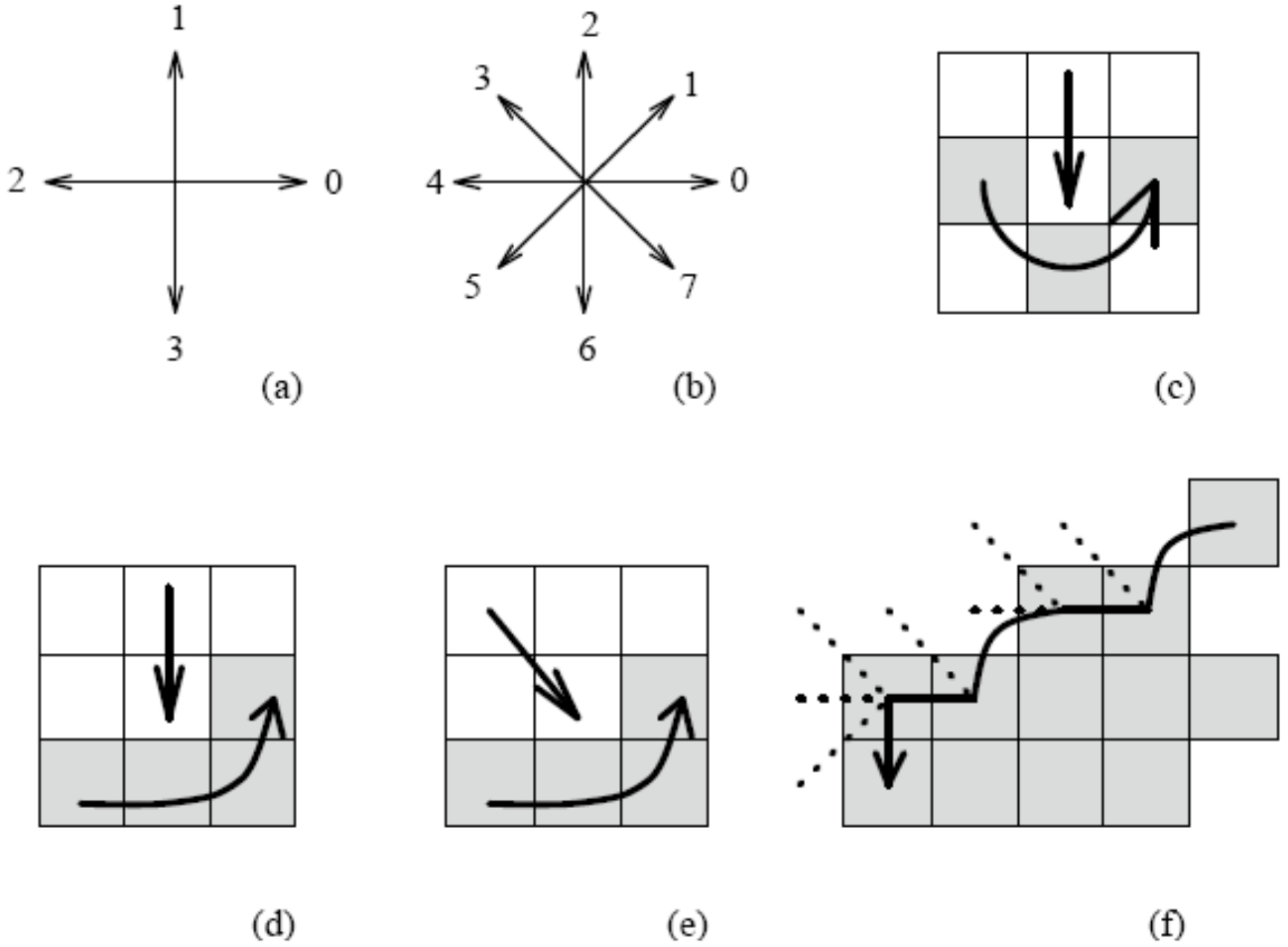


Figure 6.14: Inner boundary tracing. (a) Direction notation, 4-connectivity. (b) 8-connectivity. (c) Pixel neighborhood search sequence in 4-connectivity. (d), (e) Search sequence in 8-connectivity. (f) Boundary tracing in 8-connectivity (dotted lines show pixels tested during the border tracing).

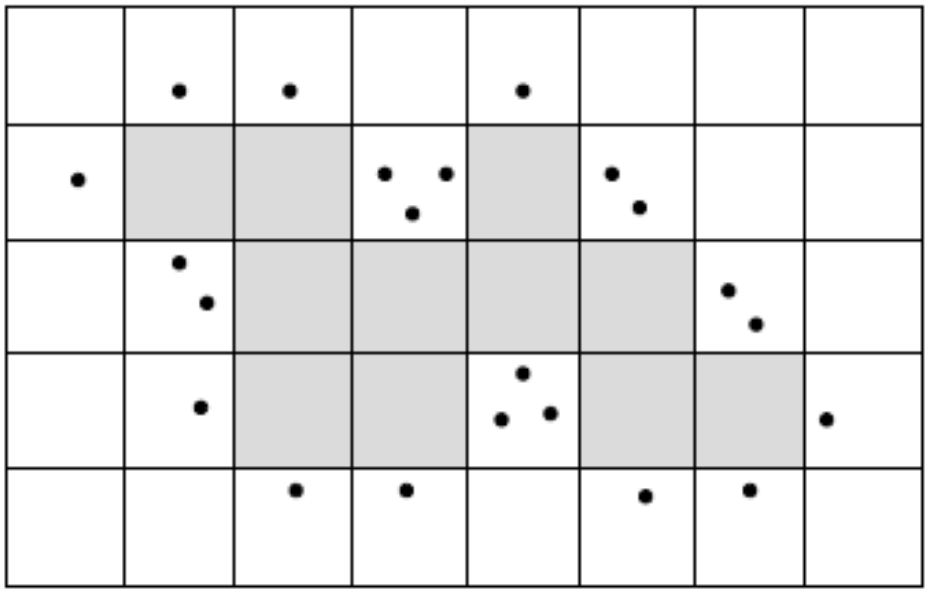
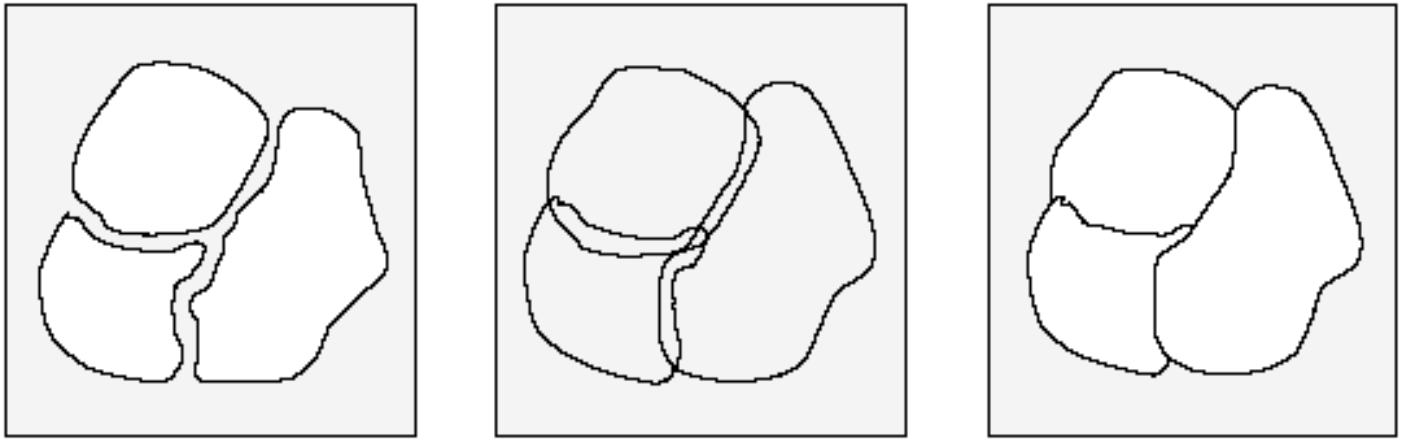


Figure 6.15: Outer boundary tracing; • denotes outer border elements. Note that some pixels may be listed several times.



(a) (b) (c)

Figure 6.16: Boundary locations for inner, outer, and extended boundary definition. (a) Inner. (b) Outer. (c) Extended.

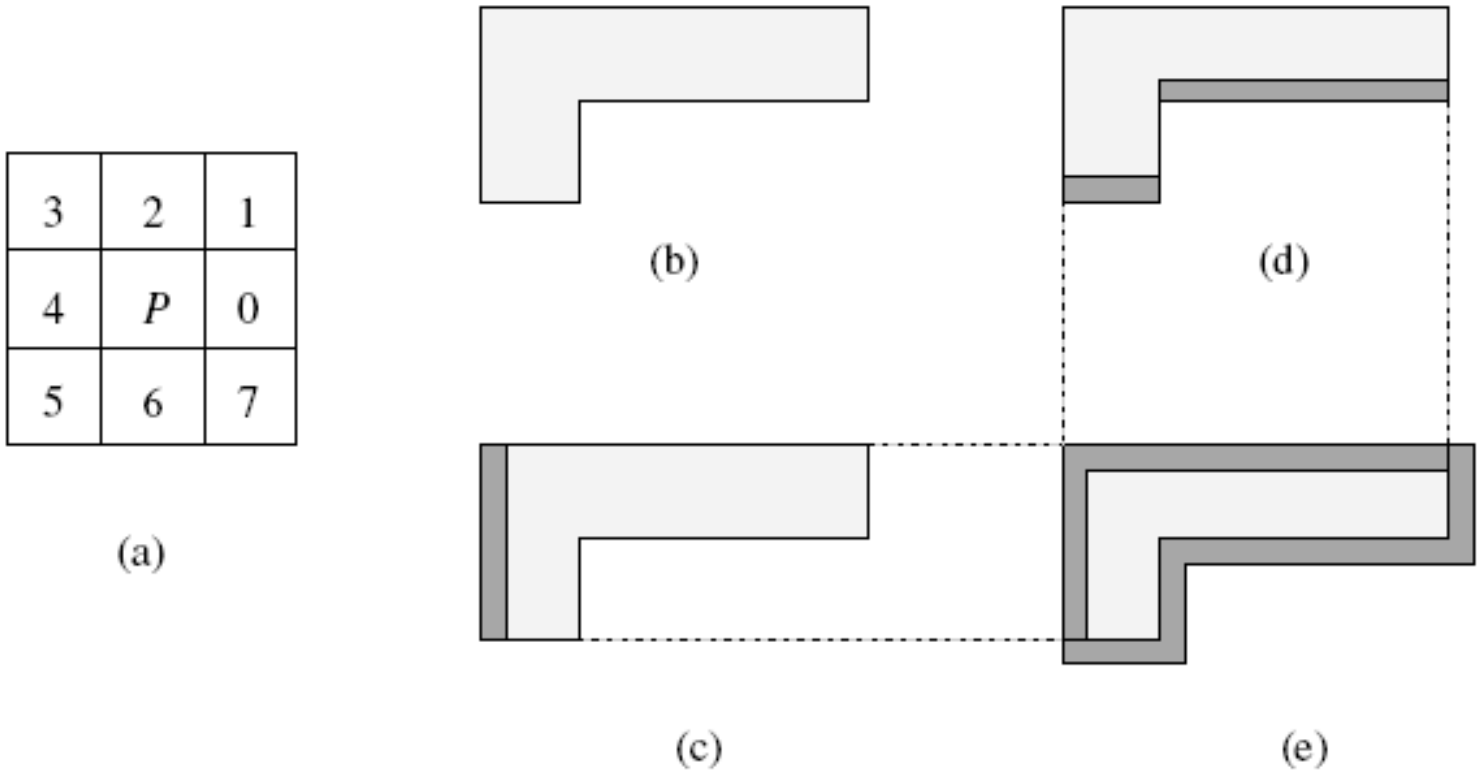


Figure 6.17: Extended boundary definition. (a) Pixel coding scheme. (b) Region R. (c) LEFT(R). (d) LOWER(R). (e) Extended boundary.

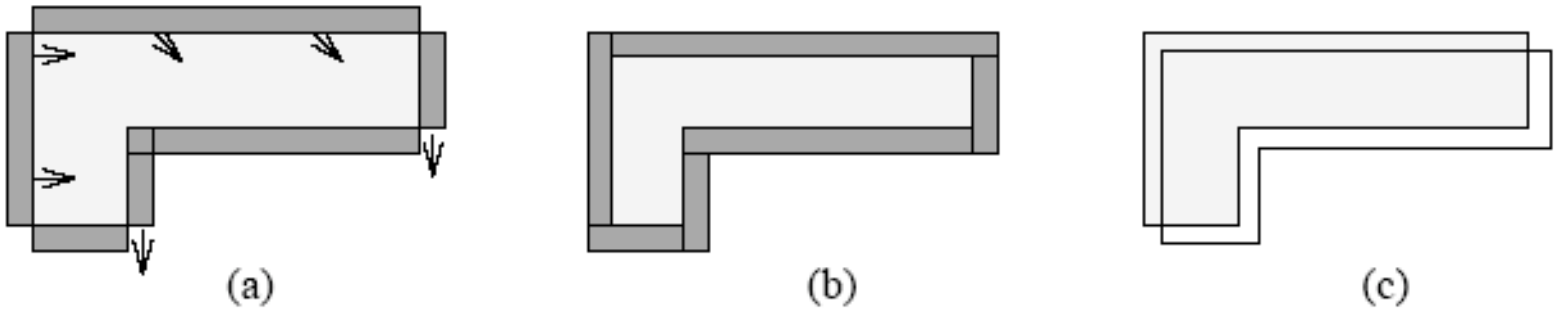


Figure 6.18: Constructing the extended boundary from outer boundary. (a) Outer boundary. (b) Extended boundary construction. (c) Extended boundary has the same shape and size as the natural object boundary.

Algorithm 6.9: Extended boundary tracing

1. Define a starting pixel of an extended boundary in a standard way (the first region pixel found in a left-to-right and top-to-bottom line-by-line image search).
2. The first move along the traced boundary from the starting pixel is in direction $dir = 6$ (down), corresponding to the situation (i) in Figure 6.19.
3. Trace the extended boundary using the look-up table in Figure 6.19 until a closed extended border results.

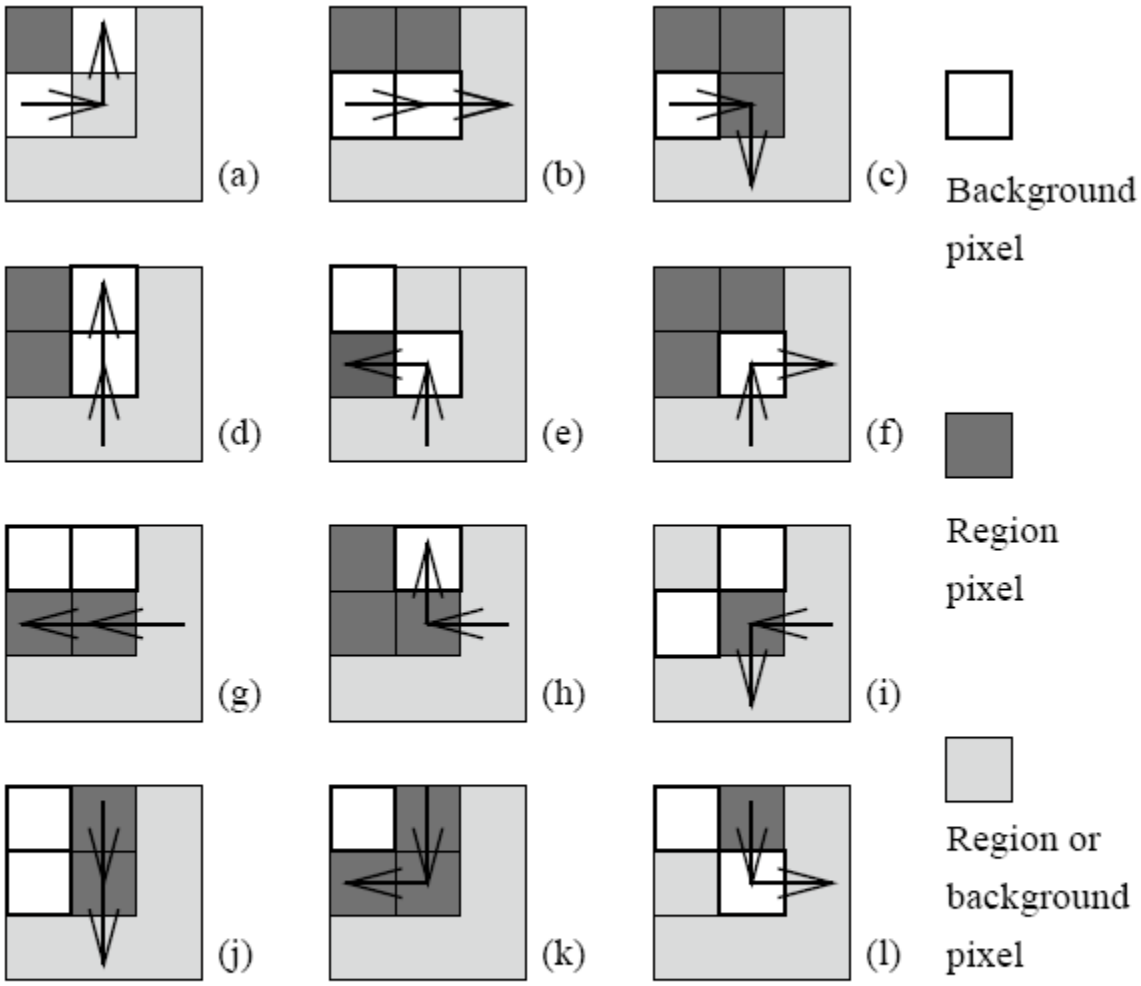
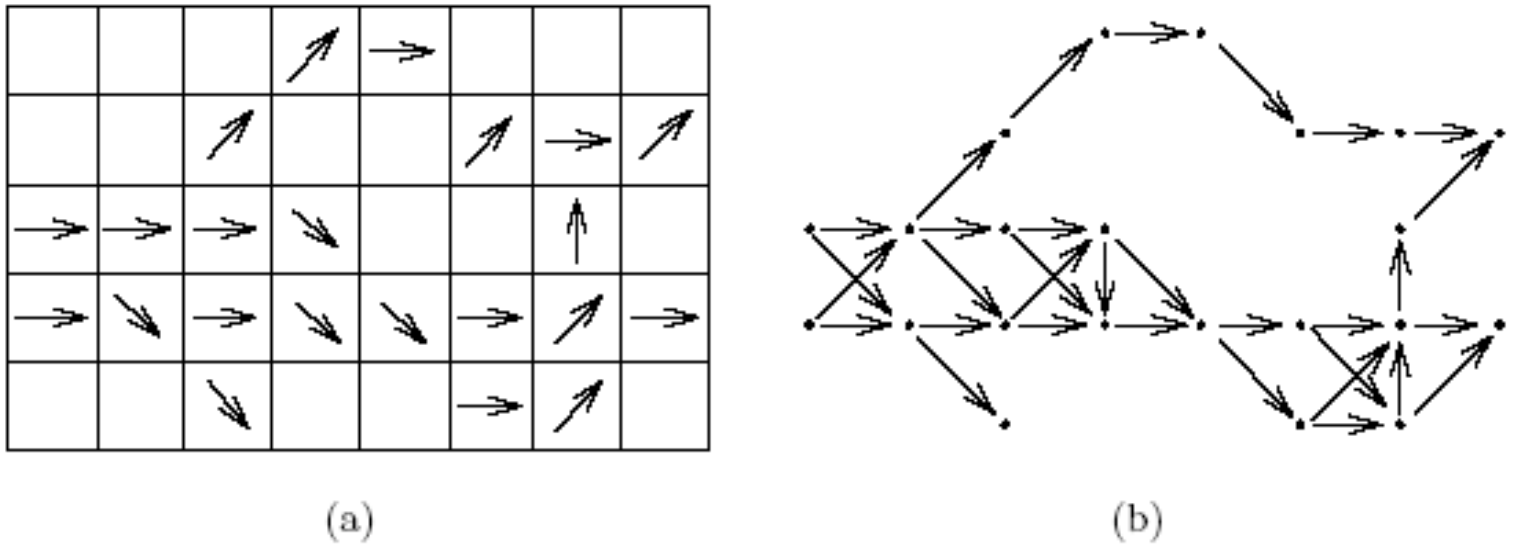


Figure 6.19: Look-up table defining all 12 possible situations that can appear during extended border tracing. Current position is in the central pixel. The direction of the next move depends on the local configuration of background and region points, and on the direction of approach to the current pixel. *Adapted from [Liow, 1991].*

Border searching in gray level images

Algorithm 6.10: Border tracing in gray-level images

1. Suppose the border has been determined up to the border element $\mathbf{x}_i$.
2. Define an element $\mathbf{x}_j$ as a pixel adjacent to $\mathbf{x}_i$ in the direction $\phi(\mathbf{x}_i)$. If the gradient magnitude in $\mathbf{x}_j$ is larger than the preset threshold, $\mathbf{x}_j$ is considered a border element; return to step 1. Otherwise proceed to step 3.
3. Compute the average gray-level value in the 3×3 neighborhood of the pixel $\mathbf{x}_j$. Compare the result with some preset gray-level value and decide whether $\mathbf{x}_j$ is positioned inside or outside the region. Proceed to step 4.
4. Try to continue the border tracing in pixel $\mathbf{x}_k$ which is adjacent to $\mathbf{x}_i$ in direction $[\phi(\mathbf{x}_i) \pm \pi/4]$, the sign being determined according to the result of step 3. If a border continuation is found, $\mathbf{x}_k$ is a new border element, and return to step 1. If $\mathbf{x}_k$ is not a border element, start the border tracing at another promising pixel.



(a) (b)

Figure 6.20: Graph representation of an edge image. (a) Edge directions of pixels with above-threshold edge magnitudes. (b) Corresponding graph.

Algorithm 6.11: A-algorithm graph search

1. Expand the starting node n_A and put all its successors into an OPEN list with pointers back to the starting node n_A. Evaluate the cost function f for each expanded node.
2. If the OPEN list is empty, fail. Determine the node n_i from the OPEN list with the lowest associated cost $f(n_i)$ and remove it. If $n_i = n_B$, then trace back through the pointers to find the optimum path and stop.
3. If the option to stop was not taken in step 2, expand the specified node n_i, and put its successors on the OPEN list with pointers back to n_i. Compute their costs f. Go to step 2.

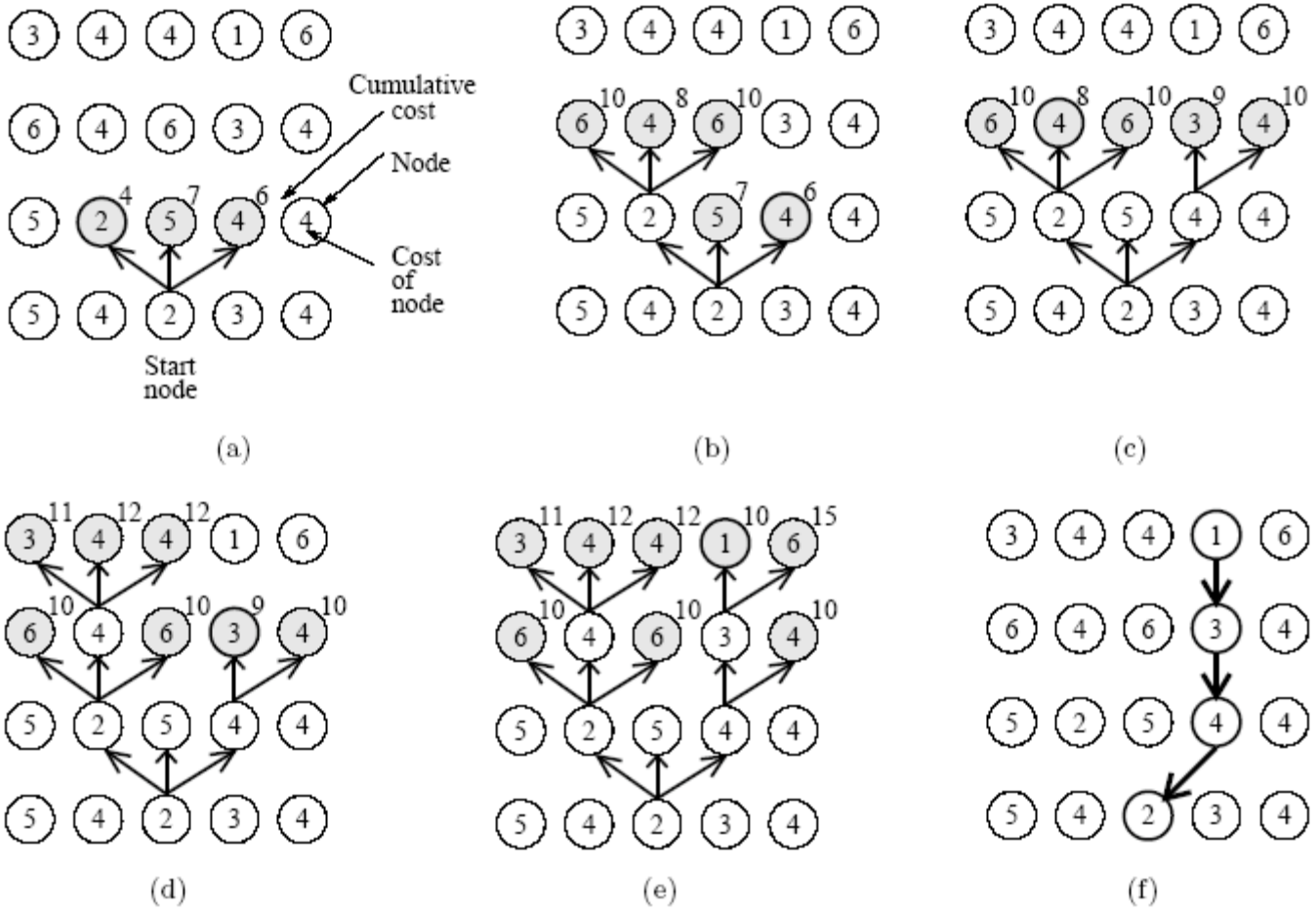


Figure 6.21: Example of a graph searching sequence using the A-algorithm (see text for description of progress of algorithm). (a) Step 1, expansion of the start node. (b) Step 2. (c) Steps 3 and 4. (d) Step 5. (e) Step 6. (f) The optimal path is defined by back-tracking.

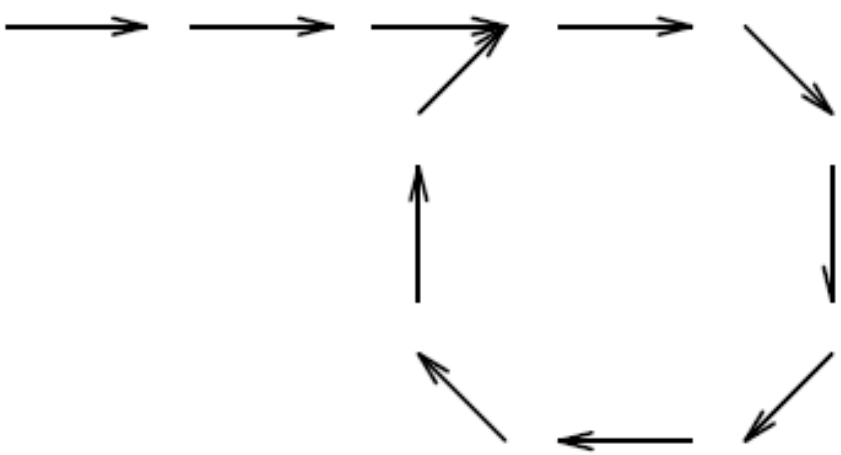


Figure 6.22: Example of following a closed loop in image data.

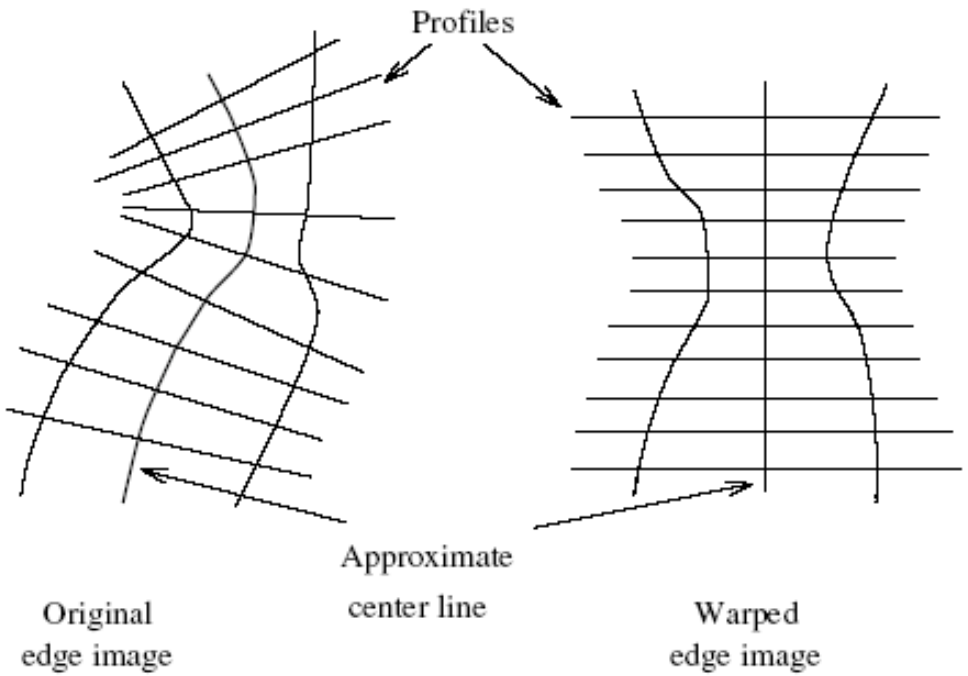
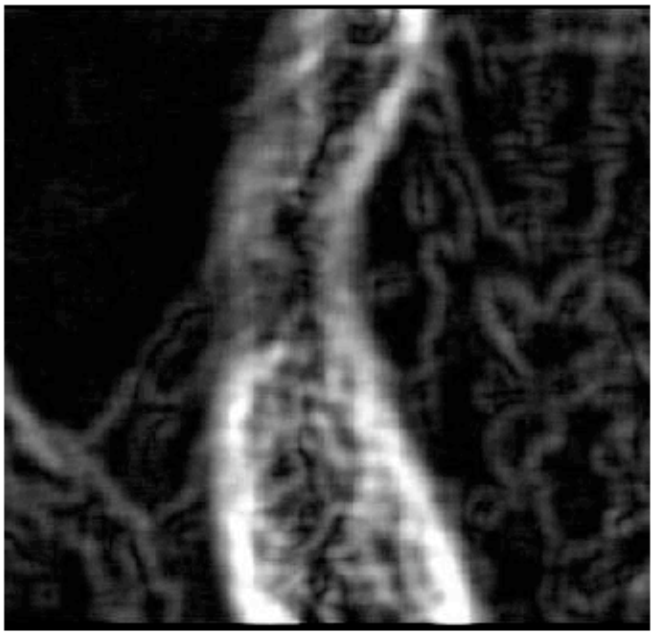
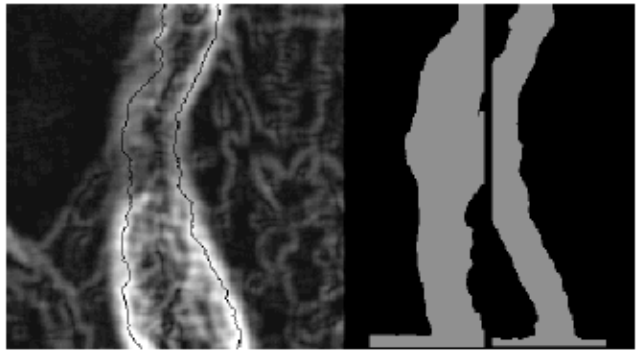


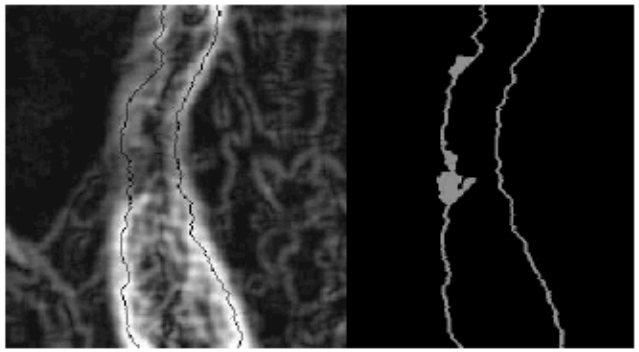
Figure 6.23: Geometric warping produces a straightened image: The graph constructed requires (and allows) searches in one main direction only (e.g., top-down). *Adapted from [Fleagle et al., 1989].*



(a)



(b)



(c)

Figure 6.24: Comparison of optimal and heuristic graph search performance. (a) Raw cost function (inverted edge image of a vessel). (b) Optimal graph search, resulting vessel borders are shown adjacent to the cost function, expanded nodes shown (38%). (c) Heuristic graph search, resulting borders and expanded nodes (2%).

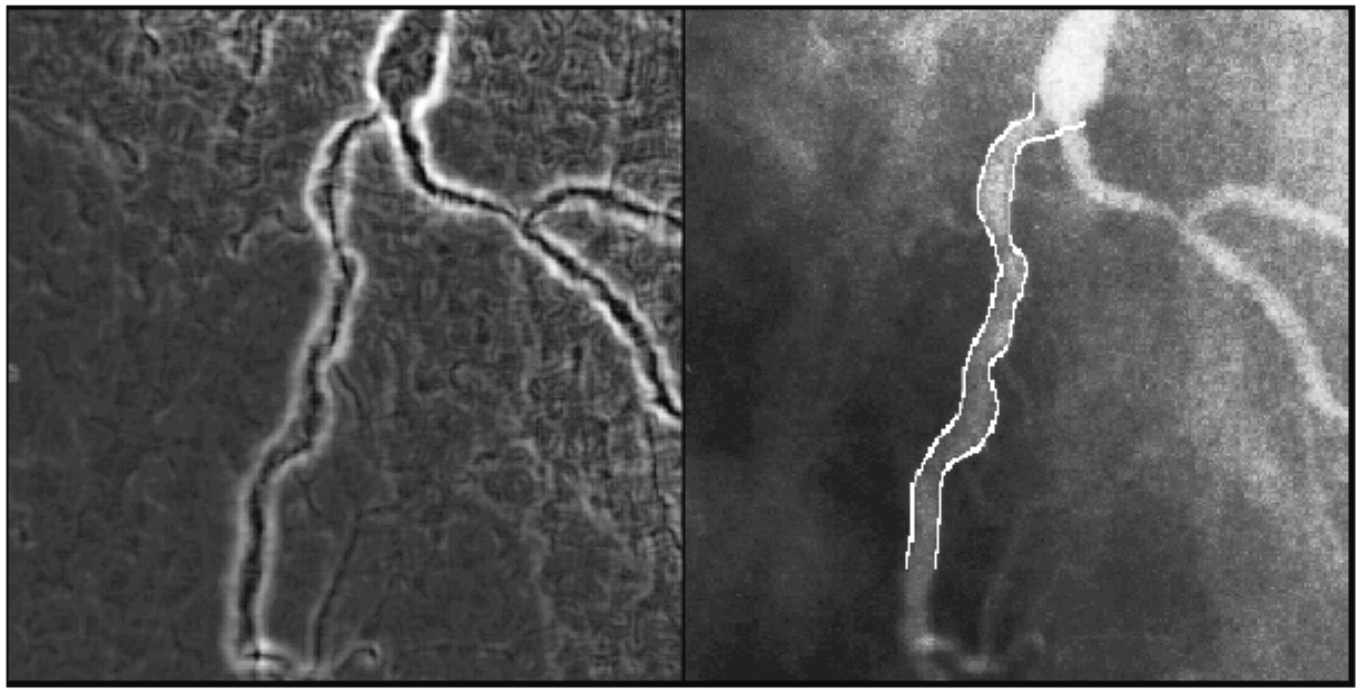


Figure 6.25: Graph search applied to coronary vessel border detection. (a) Edge image. (b) Determined vessel borders.

Bidirectional search can be used to create partial borders.
Search proceeds in both directions: forward search links successors;
backward search links predecessors

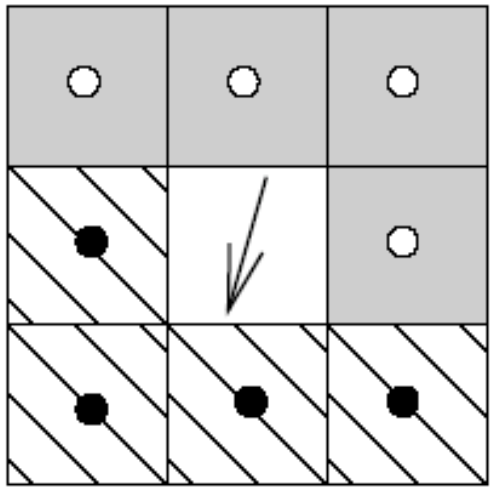


Figure 6.26: Bidirectional heuristic search: Edge predecessors (marked ○) and successors (marked •).

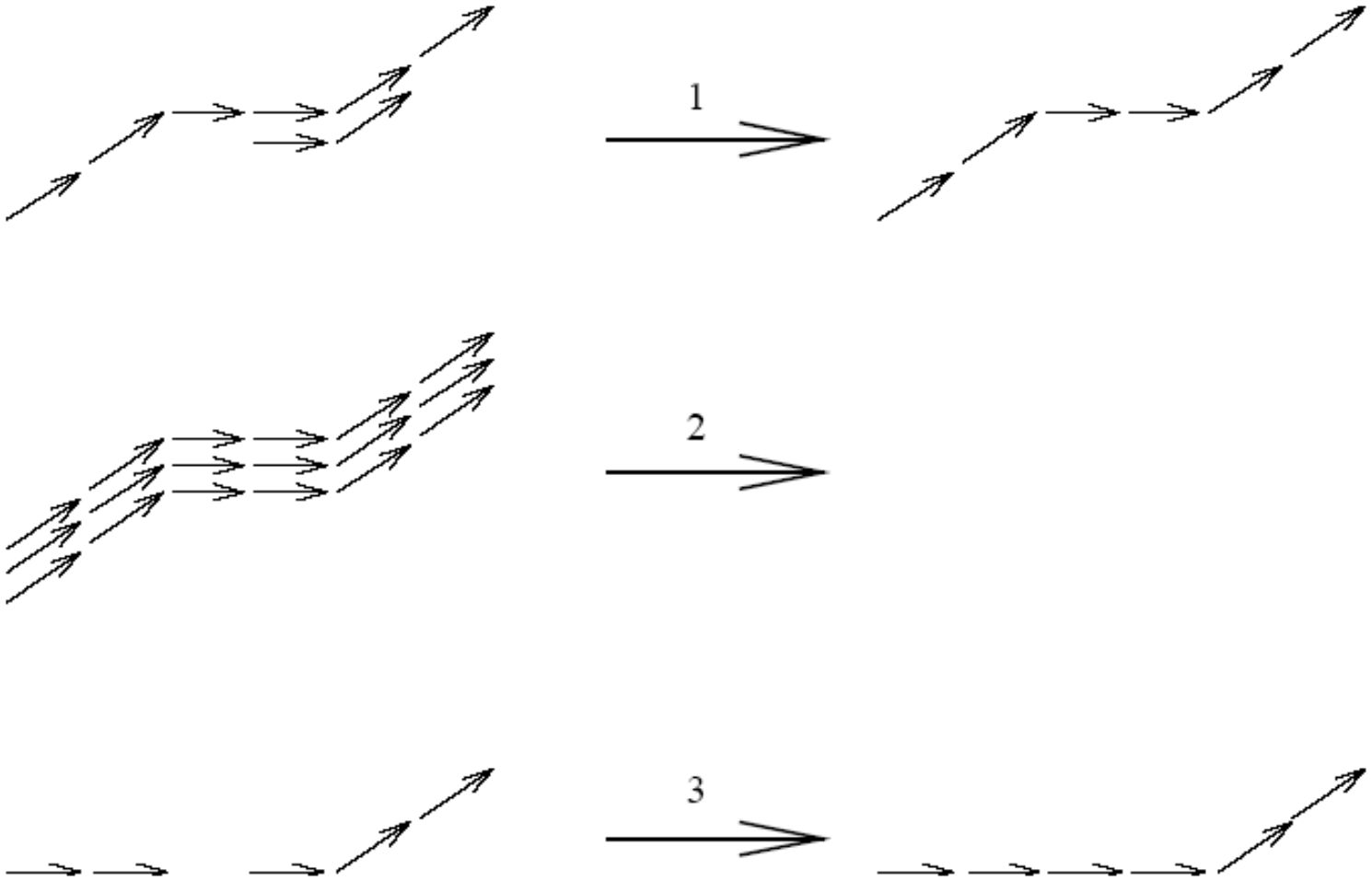


Figure 6.27: Rules for edge chain modification; note that edge responses resulting from continuous changes of illumination are removed (case 2).

Algorithm 6.12: Heuristic search for image borders

1. Search for the strongest edge in the image, not considering edges previously marked or edges that already are part of located borders. Mark the located edge. If the edge magnitude is less than the preset threshold or if no image edge was found, go to step 5.
2. Expand all the image edges positioned in front of the specified starting edge until no new successors can be found.
3. Expand all the image edges positioned behind the specified starting edge until no new predecessors can be found. In steps 2 and 3 do not include edges that are already part of any existing edge chain.
4. If the resulting edge chain consists of at least three edges, it is stored in the chain list, otherwise it is deleted. Proceed to step 1.
5. Modify edge chains according to the rules given in Figure 6.27.
6. Repeat step 5 until the resulting borders do not change substantially from step to step.

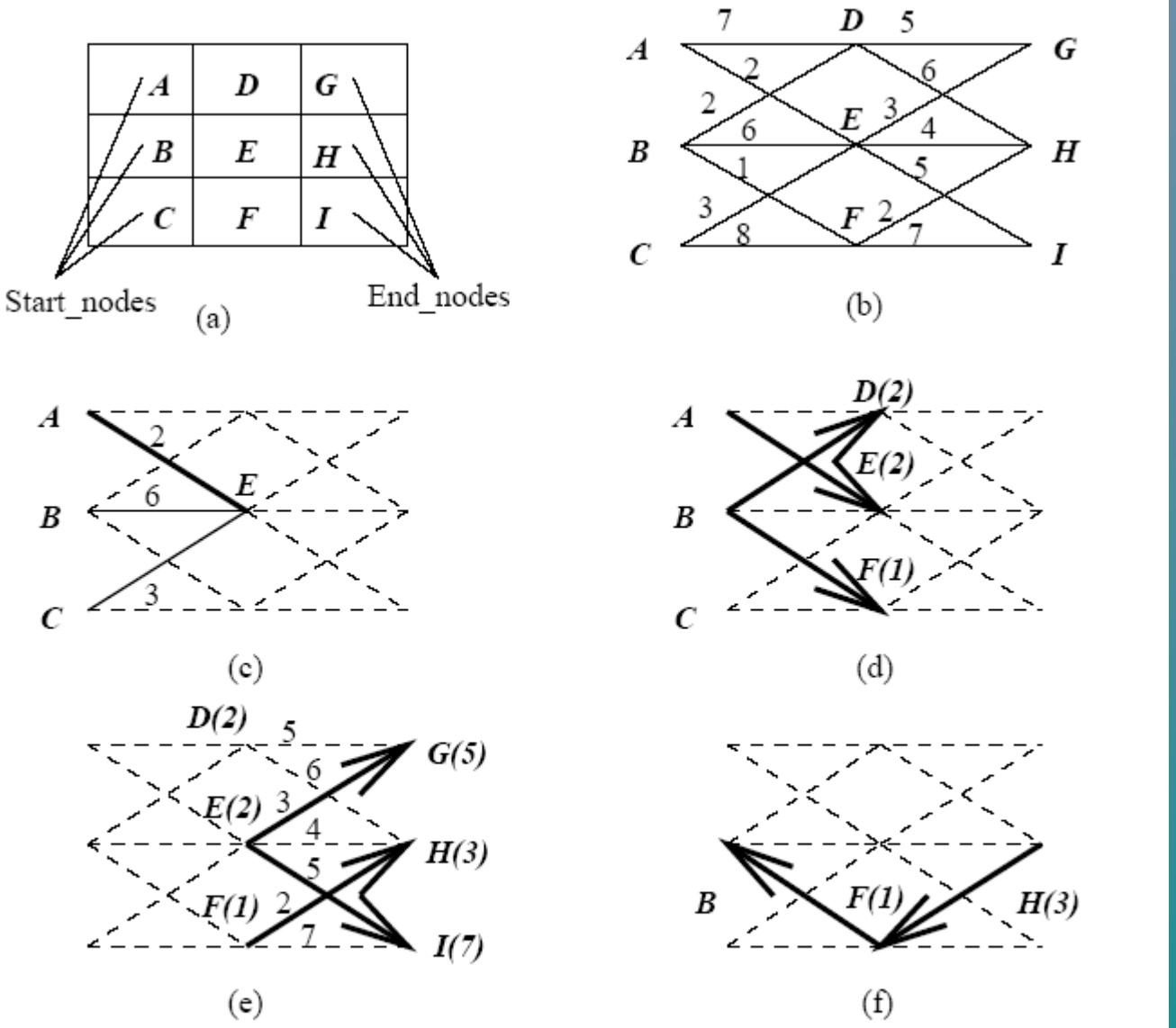


Figure 6.28: Edge following as dynamic programming. (a) Edge image. (b) Corresponding graph, partial costs assigned. (c) Possible paths from any start point to E, $A-E$ is optimal. (d) Optimal partial paths to nodes D, E, F. (e) Optimal partial paths to nodes G, H, I. (f) Back-tracking from H defines the optimal boundary.

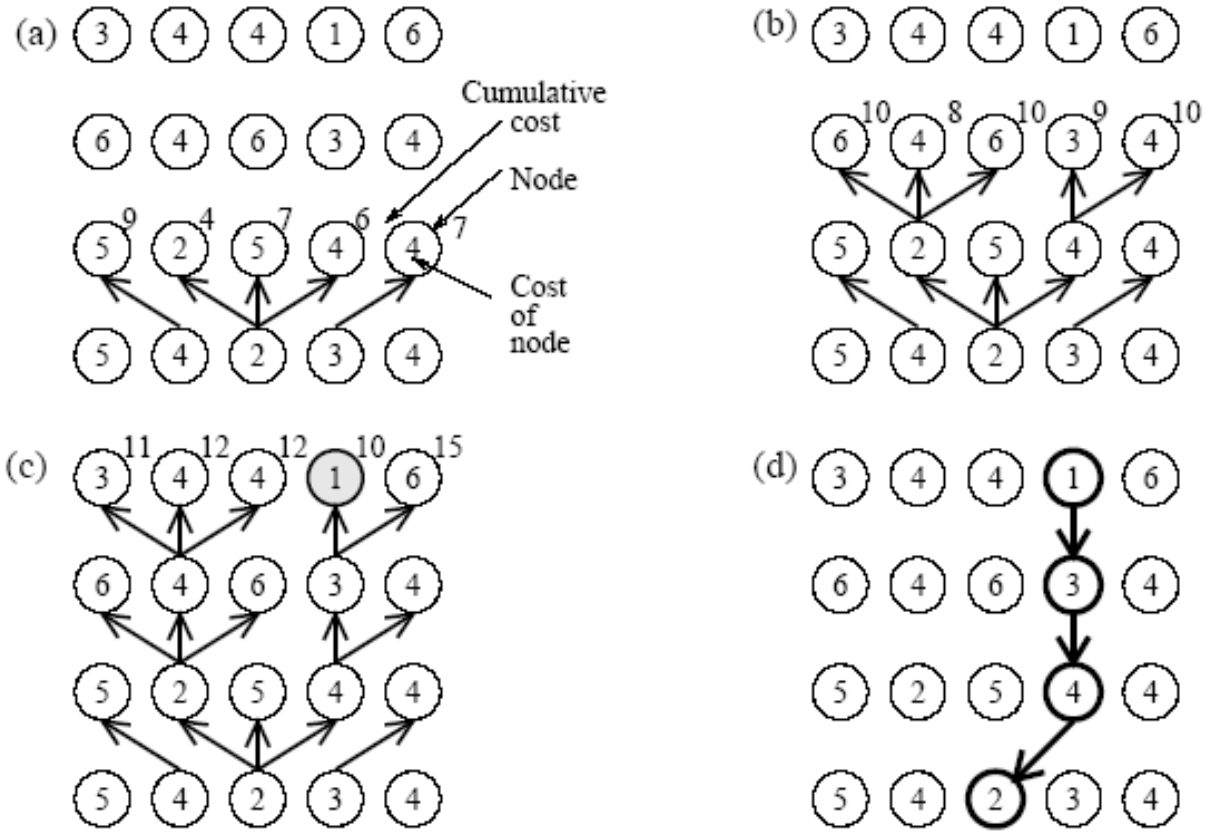


Figure 6.29: Example of a graph searching sequence using dynamic programming: (a) step 1, expansion of the first graph layer; (b) step 2; (c) step 3—the minimum-cost node in the last layer marked; (d) the optimal path is defined by back-tracking.

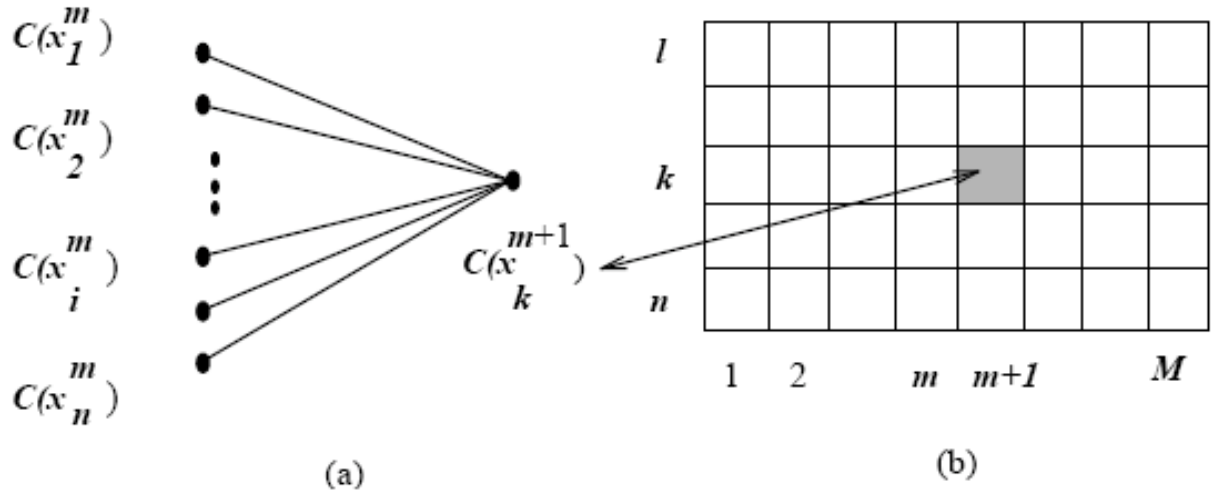


Figure 6.30: Dynamic programming: (a) one step of the cost calculation; (b) graph layers, node notation.

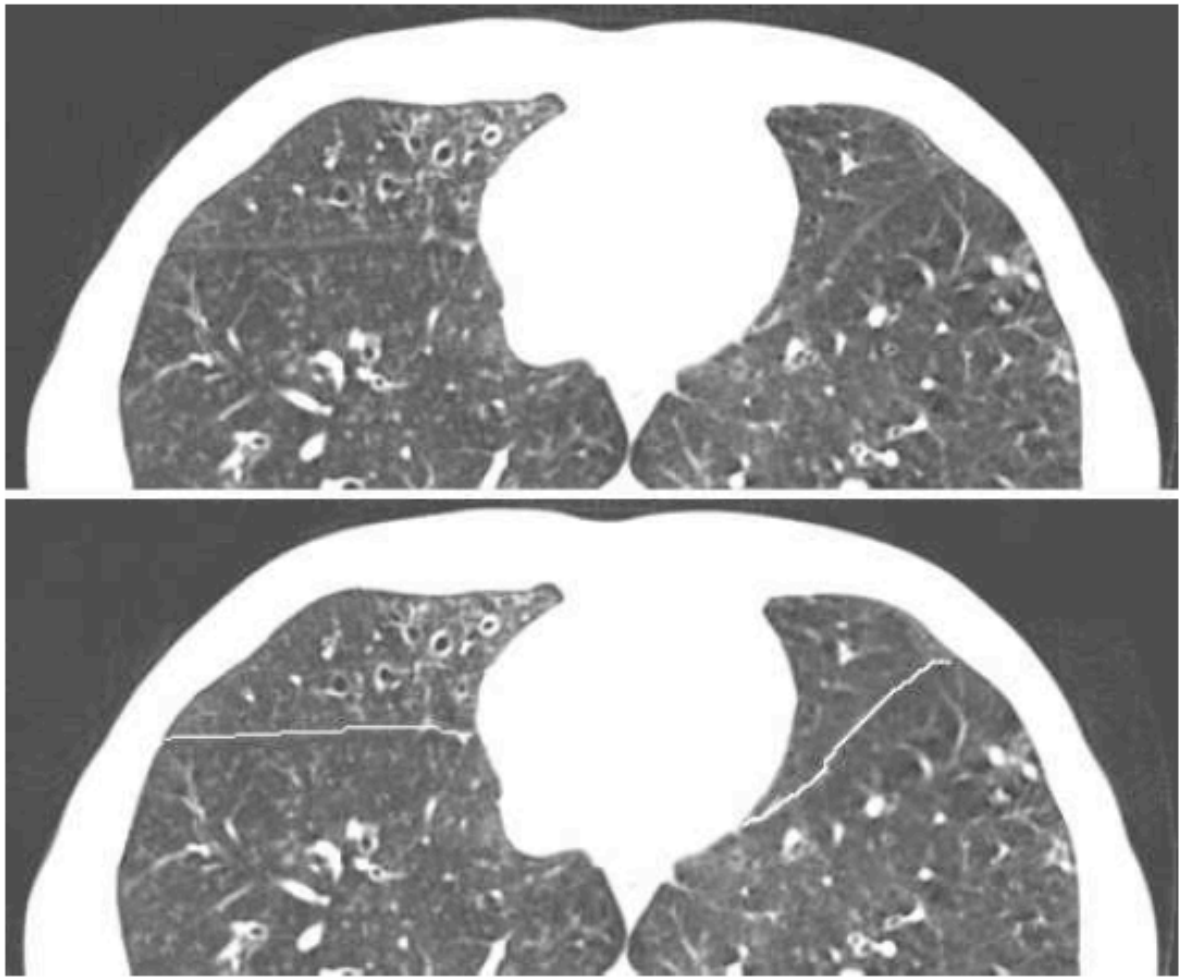


Figure 6.31: Detection of pulmonary fissures using dynamic programming. Top: Sub-region of an original cross-sectional X-ray CT image of a human lung. Bottom: Detected fissures shown in white.

Algorithm 6.13: Boundary tracing as dynamic programming

1. Specify initial costs $C(x_i^1)$ of all nodes in the first graph layer, $i = 1, \ldots, n$ and partial path costs $g^m(i,k)$, $m = 1, \ldots, M-1$.
2. Repeat step 3 for all $m = 1, \ldots, M-1$.
3. Repeat step 4 for all nodes $k = 1, \ldots, n$ in the graph layer m.
4. Let
$$C\left(x_k^{m+1}\right) = \min_{i=-1,0,1} \left(C(x_{k+i}^m) + g^m(i,k) \right). \qquad (6.24)$$
Set pointer from node x_k^{m+1} back to node x_i^{m*}; where $*$ denotes the optimal predecessor.
5. Find an optimal node x_k^{M*} in the last graph layer M and obtain an optimal path by back-tracking through the pointers from x_k^{M*} to x_i^{1*}.

Dpboundary_demo; cd 06Segm1

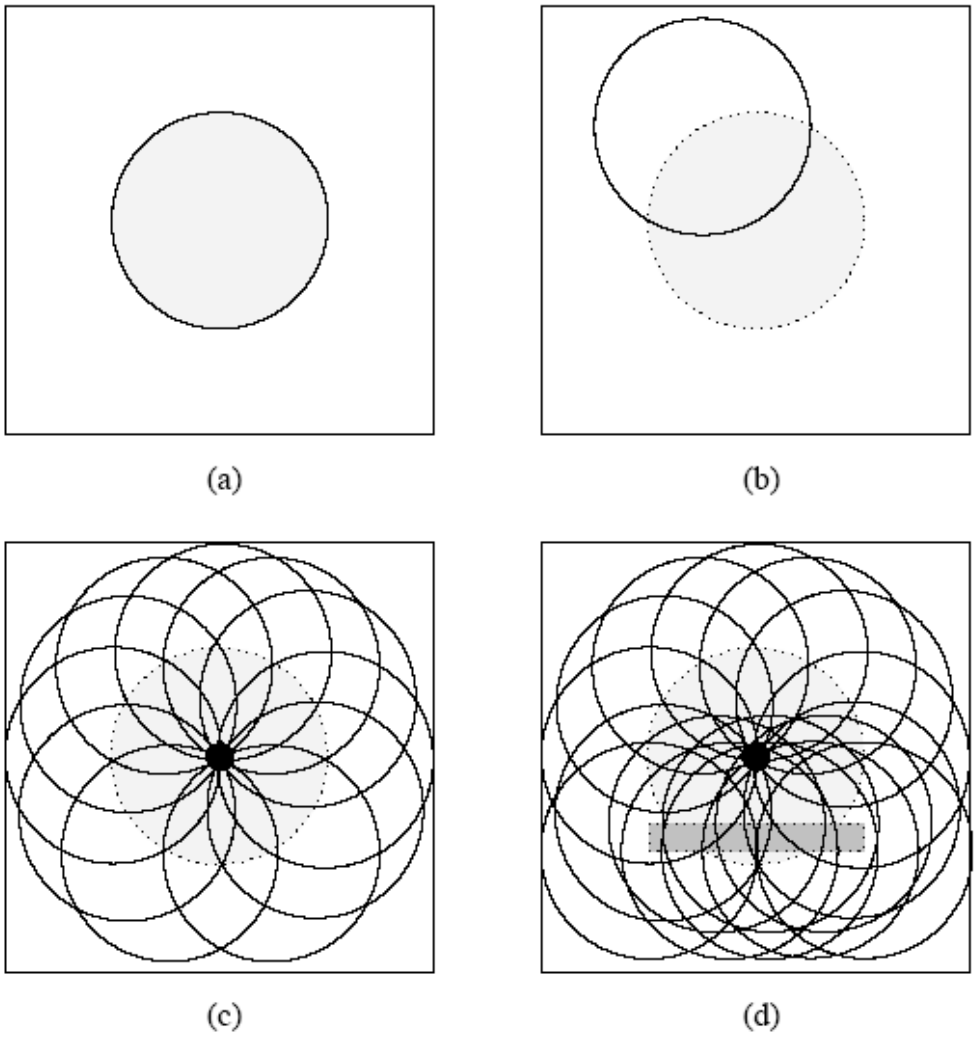


Figure 6.32: Hough transform—example of circle detection. (a) Original image of a dark circle (known radius r) on a bright background. (b) For each dark pixel, a potential circle-center locus is defined by a circle with radius r and center at that pixel. (c) The frequency with which image pixels occur in the circle-center loci is determined—the highest-frequency pixel represents the center of the circle (marked by •). (d) The Hough transform correctly detects the circle (marked by •) in the presence of incomplete circle information and overlapping structures. (See Figure 6.37 for a real-life example.)

Hough transform: line detection

Each image point votes for multiple lines in the *k-q* parameter space.
Votes are accumulated in array *A(k,q); k-q* space is quantized.

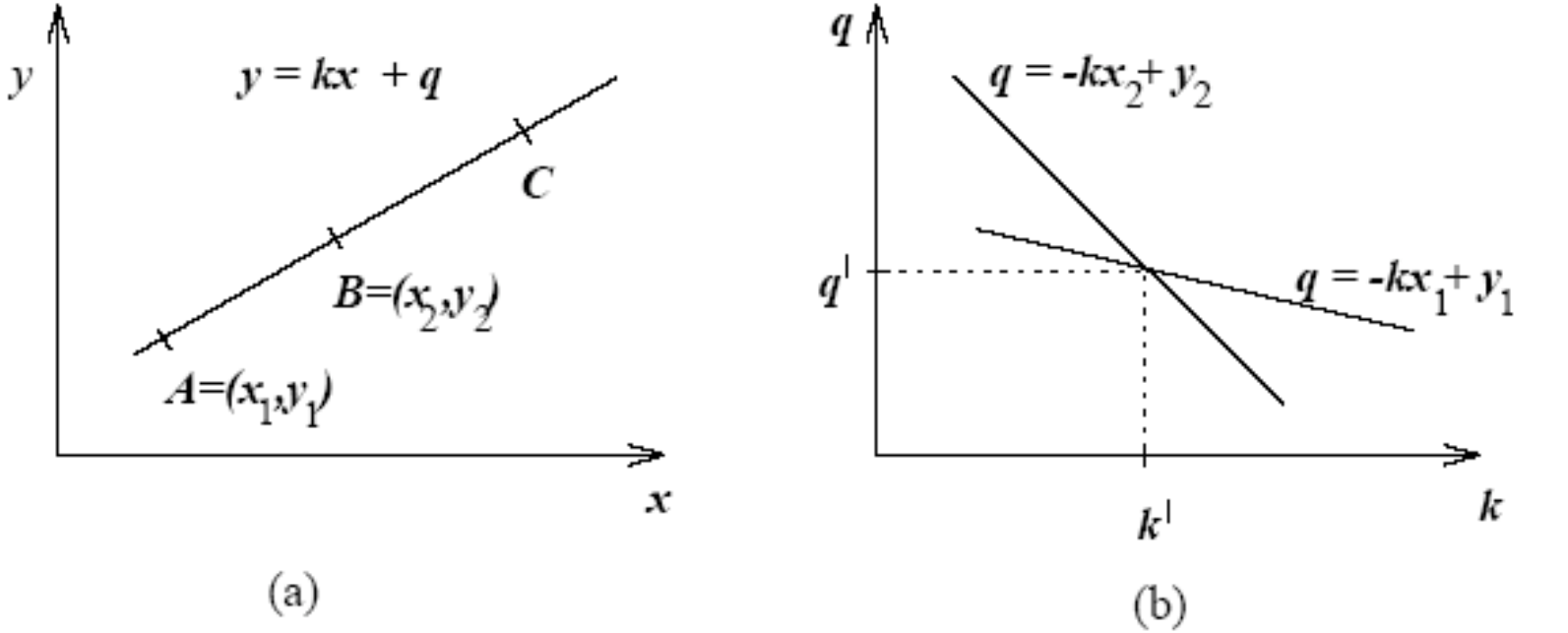


Figure 6.33: Hough transform principles. (a) Image space. (b) k, q parameter space.

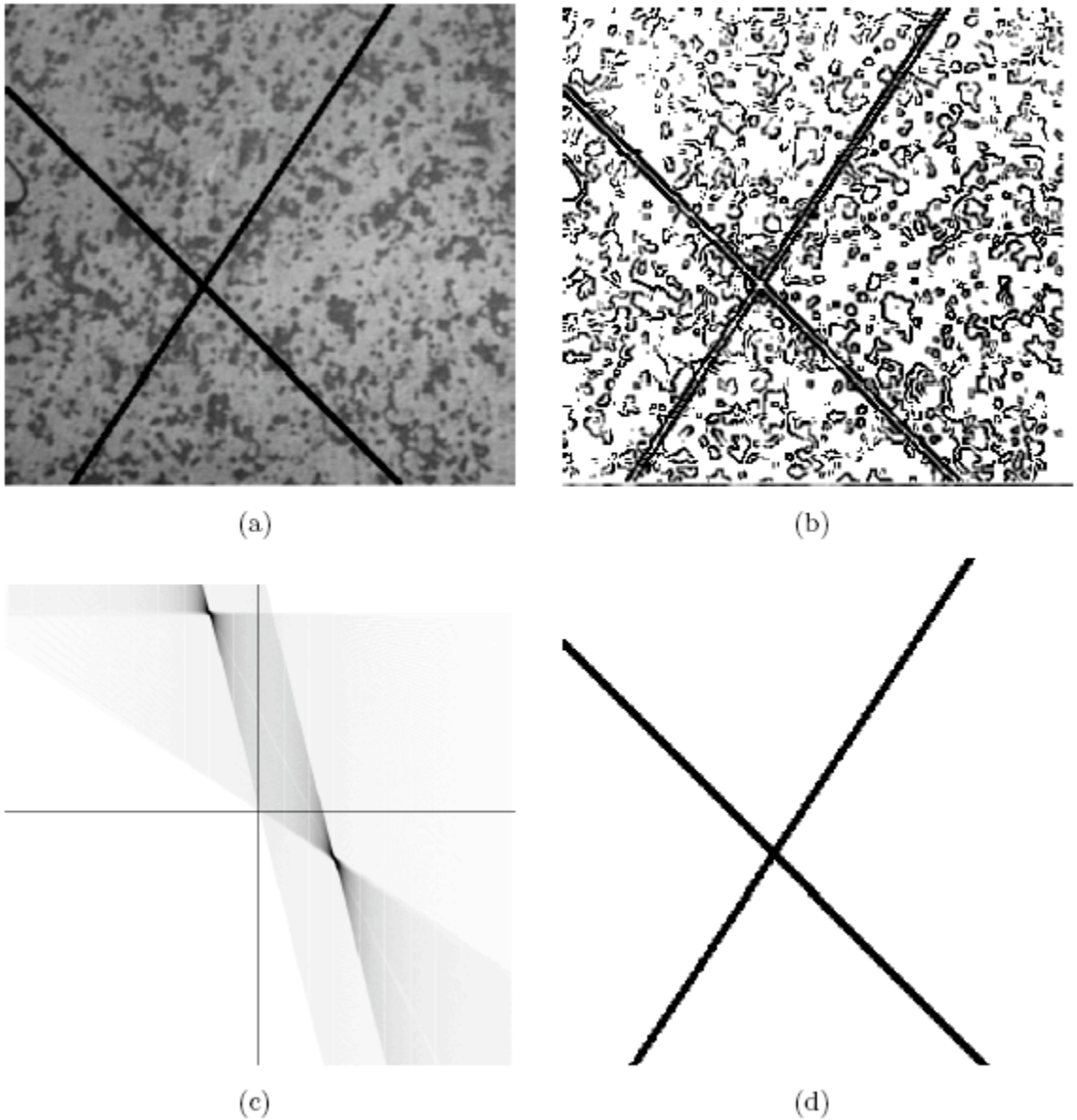


Figure 6.34: Hough transform—line detection. (a) Original image. (b) Edge image (note many edges, which do not belong to the line). (c) Parameter space. (d) Detected lines.

Hough transform: $s = x\cos\theta + y\sin\theta$

Each image point votes for multiple lines in the *s*-θ parameter space.
Votes are accumulated in array *A(s,*θ*); s*-θ is quantized.

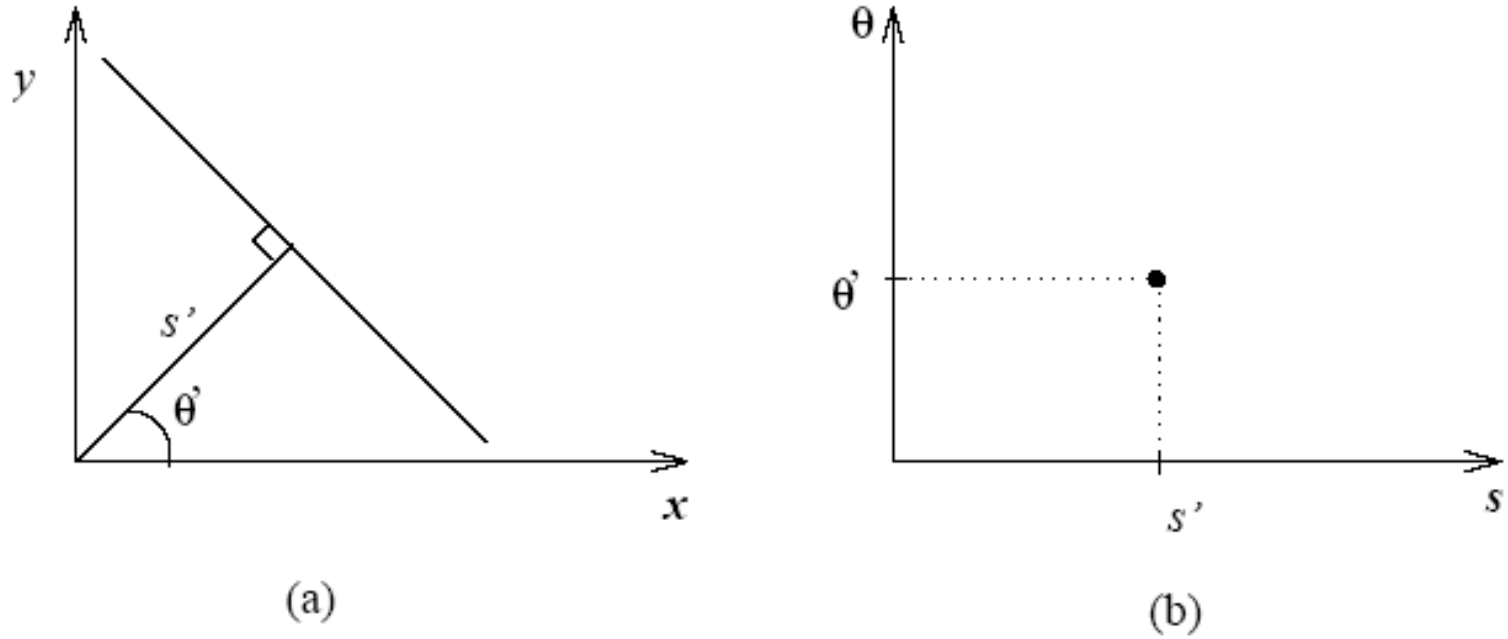
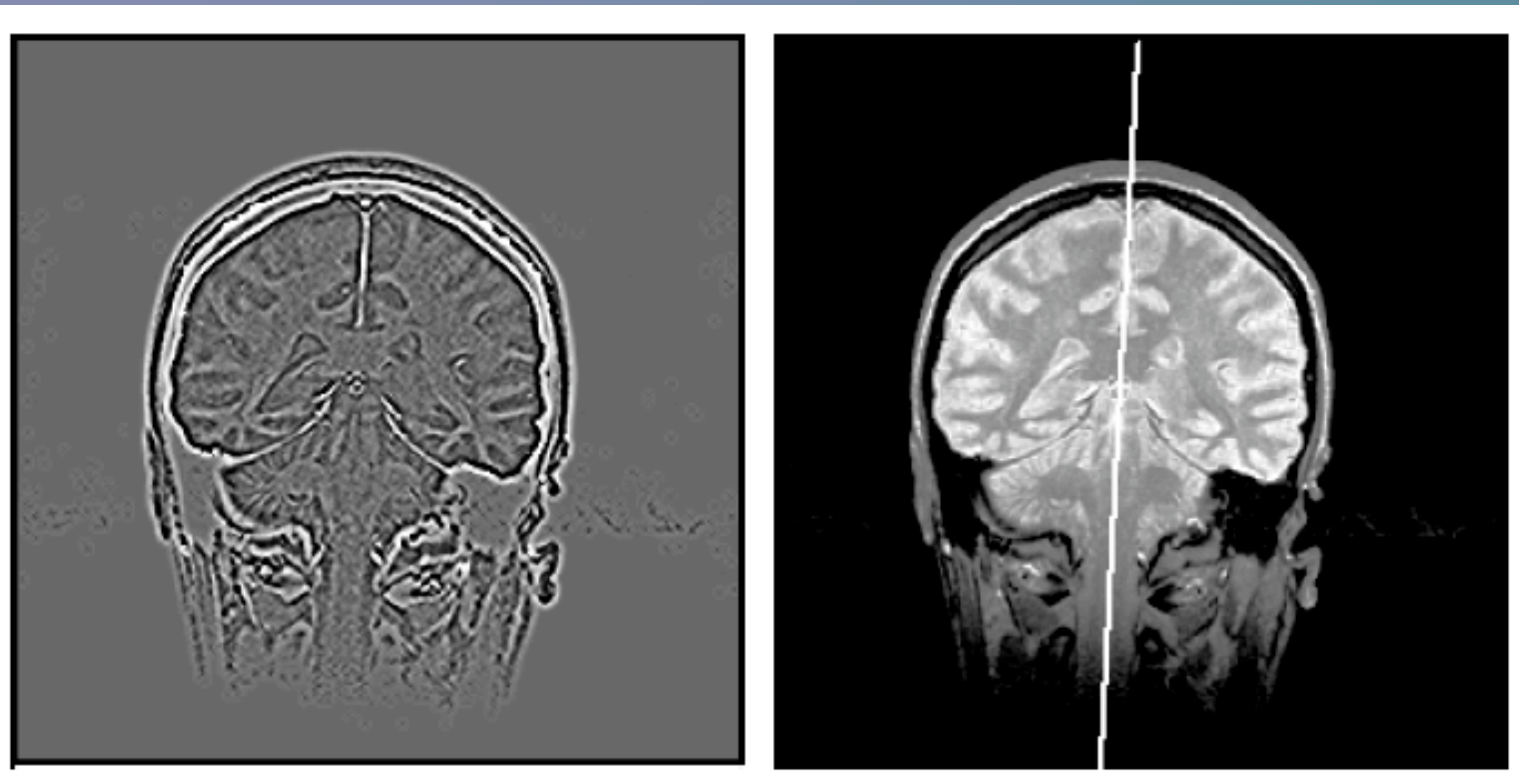


Figure 6.35: Hough transform in s, θ space. (a) Straight line in image space. (b) s, θ parameter space.

Hough_lines_demo: cd 06Segm1, addpath ../01Intro/



(a) (b)

Figure 6.36: Hough transform line detection used for MRI brain segmentation to the left and right hemispheres. (a) Edge image. (b) Segmentation line in original image data.

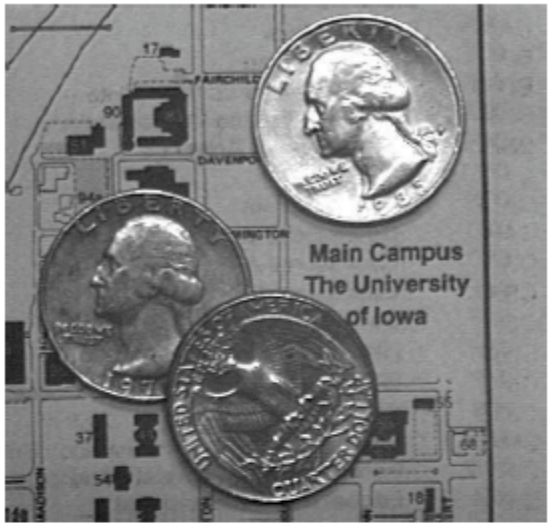
Hough transform: curve detection

Algorithm 6.14: Curve detection using the Hough transform

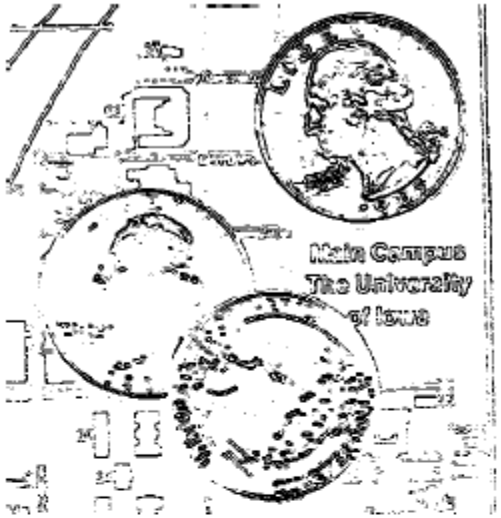
1. Quantize parameter space within the limits of parameters $\mathbf{a}$. The dimensionality n of the parameter space is given by the number of parameters of the vector $\mathbf{a}$.
2. Form an n-dimensional accumulator array $A(\mathbf{a})$ with structure matching the quantization of parameter space; set all elements to zero.
3. For each image point (x_1, x_2) in the appropriately thresholded gradient image, increase all accumulator cells $A(\mathbf{a})$ if $f(\mathbf{x}, \mathbf{a}) = 0$

$$A(\mathbf{a}) = A(\mathbf{a}) + \Delta A$$

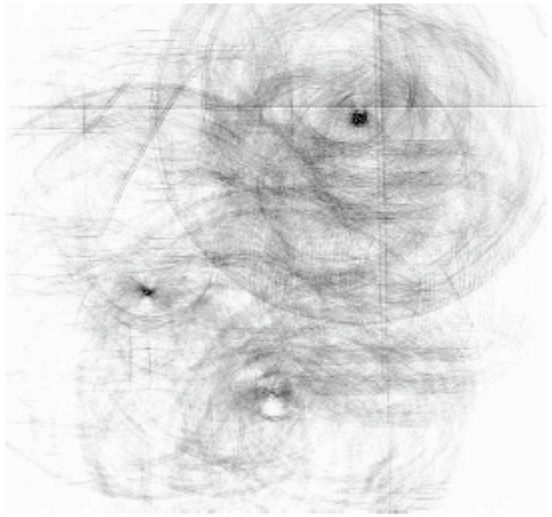
   for all $\mathbf{a}$ inside the limits used in step 1.
4. Local maxima in the accumulator array $A(\mathbf{a})$ correspond to realizations of curves $f(\mathbf{x}, \mathbf{a})$ that are present in the original image.



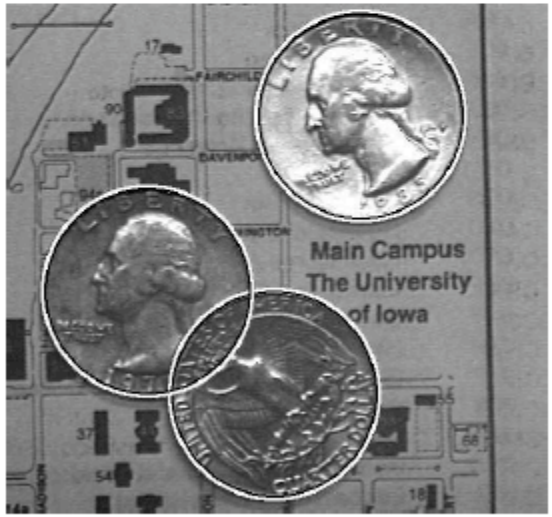
(a)



(b)



(c)



(d)

Figure 6.37: Hough transform—circle detection. (a) Original image. (b) Edge image (note that the edge information is far from perfect). (c) Parameter space. (d) Detected circles.

Generalized hough transform: detecting general shapes

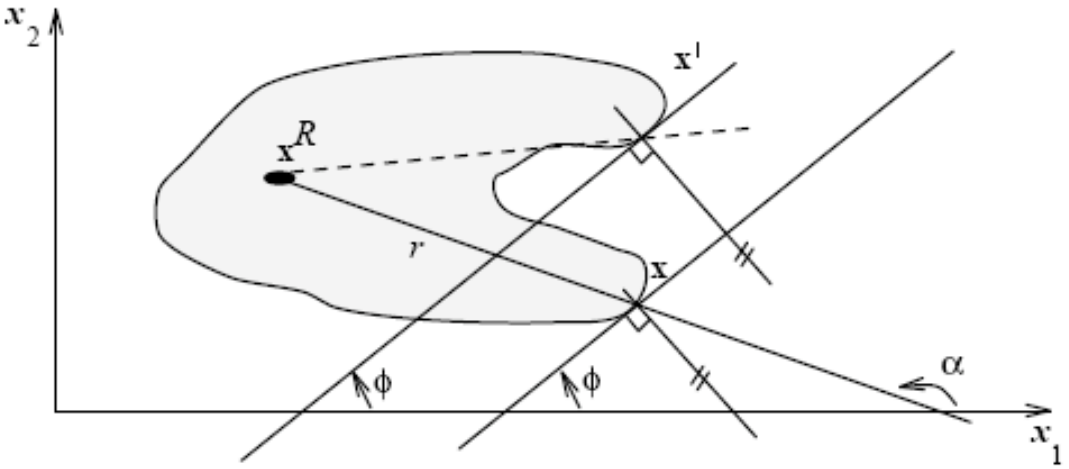


Figure 6.38: Principles of the generalized Hough transform: geometry of R-table construction.

ϕ_1	$(r_1^1, \alpha_1^1), (r_1^2, \alpha_1^2), \ldots, (r_1^{n_1}, \alpha_1^{n_1})$
ϕ_2	$(r_2^1, \alpha_2^1), (r_2^2, \alpha_2^2), \ldots, (r_2^{n_2}, \alpha_2^{n_2})$
ϕ_3	$(r_3^1, \alpha_3^1), (r_3^2, \alpha_3^2), \ldots, (r_3^{n_3}, \alpha_3^{n_3})$
$\ldots$	$\ldots$
ϕ_k	$(r_k^1, \alpha_k^1), (r_k^2, \alpha_k^2), \ldots, (r_k^{n_k}, \alpha_k^{n_k})$

Table 6.1: R-table

Algorithm 6.15: Generalized Hough transform

1. Construct an R-table description of the desired object.
2. Form a data structure A that represents the potential reference points

$$A(x_1, x_2, S, \tau) .$$

   Set all accumulator cell values $A(x_1, x_2, S, \tau)$ to zero.

3. For each pixel (x_1, x_2) in a thresholded gradient image, determine the edge direction $\Phi(\mathbf{x})$; find all potential reference points $\mathbf{x}^R$ and increase all

$$A(\mathbf{x}^R, S, \tau) = A(\mathbf{x}^R, S, \tau) + \Delta A$$

   for all possible values of rotation and size change

$$x_1^R = x_1 + r(\phi + \tau)\, S \cos\big(\alpha(\phi + \tau)\big) ,$$
$$x_2^R = x_2 + r(\phi + \tau)\, S \sin\big(\alpha(\phi + \tau)\big) .$$

4. The location of suitable regions is given by local maxima in the A data structure.

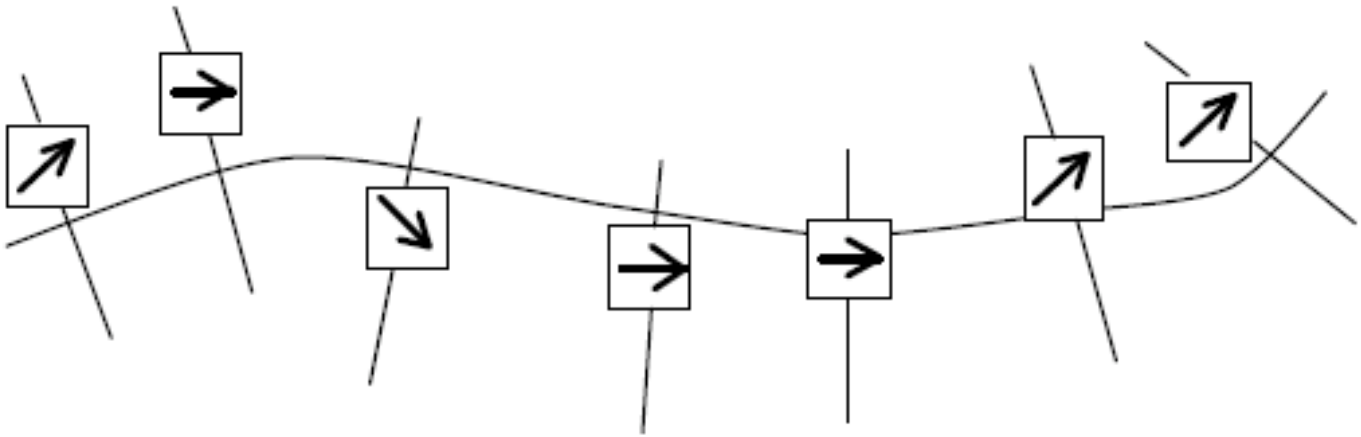


Figure 6.39: A priori information about boundary location.

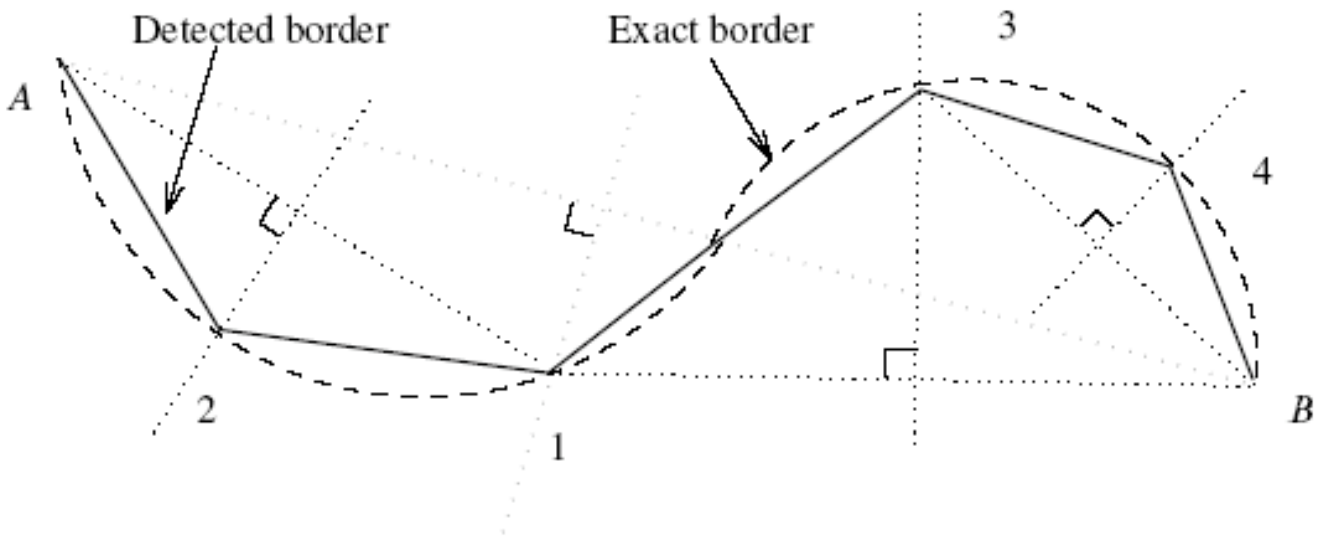


Figure 6.40: Divide-and-conquer iterative border detection; numbers show the sequence of division steps.

Summary

Edge-based image segmentation

- Edge-based segmentation relies on edges found in an image by edge detecting operators—these edges mark image locations of discontinuities in gray-level, color, texture, etc.
- The most common problems of edge-based segmentation, caused by image noise or unsuitable information in an image, are an edge presence in locations where there is no border, and no edge presence where a real border exists.
- **Edge image thresholding** is based on construction of an edge image that is processed by an appropriate threshold.
- In **edge relaxation**, edge properties are considered in the context of neighboring edges. If sufficient evidence of the border presence exists, local edge strength increases and vice versa. Using a global relaxation (optimization) process, continuous borders are constructed.

Summary

Edge-based image segmentation cont.

- Three types of region borders may be formed: **inner**, **outer**, and **extended**. The inner border is always part of a region, but the outer border never is. Therefore, using inner or outer border definition, two adjacent regions never have a common border. Extended borders are defined as single common borders between adjacent regions still being specified by standard pixel co-ordinates.

- If the criterion of optimality is defined, globally optimal borders can be determined using **(heuristic) graph searching** or **dynamic programming**. Graph-searchbased border detection represents an extremely powerful segmentation approach. The border detection process is transformed into a search for the optimal path in the weighted graph. Costs are associated with each graph node that reflect the likelihood that the border passes through the particular node (pixel). The aim is to find the optimal path (optimal border, with respect to some objective function) that connects two specified nodes or sets of nodes that represent the border's beginning and end.

Summary

Edge-based image segmentation cont.

- **Cost definition** (evaluation functions) is the key to successful border detection. Cost calculation complexity may range from simple inverted edge strength to complex representation of a priori knowledge about the sought borders, segmentation task, image data, etc.
- Graph searching uses Nilsson's **A-algorithm** and guarantees optimality. **Heuristic graph search** may substantially increase search speed, although the heuristics must satisfy additional constraints to guarantee optimality.
- **Dynamic programming** is based on the principle of optimality and presents an efficient way of simultaneously searching for optimal paths from multiple starting and ending points.

Summary

Edge-based image segmentation cont.

- Using the A-algorithm to search a graph, it is not necessary to construct the entire graph since the costs associated with expanded nodes are calculated only if needed. In dynamic programming, a complete graph must be constructed.
- If calculation of the local cost functions is computationally inexpensive, dynamic programming may represent a computationally less demanding choice. However, which of the two graph searching approaches (A-algorithm, dynamic programming) is more efficient for a particular problem depends on the evaluation functions and on the quality of heuristics for an A-algorithm.

Summary

Edge-based image segmentation cont.

- **Hough transform** segmentation is applicable if objects of known shape are to be detected within an image. The Hough transform can detect straight lines and curves (object borders) if their analytic equations are known. It is robust in recognition of occluded and noisy objects.
- The generalized Hough transform can be used if the analytic equations of the searched shapes are not available; the parametric curve (region border) description is based on sample situations and is determined in the learning stage.
- While forming the regions from complete borders is trivial, **region determination from partial borders** may be a very complex task. Region construction may be based on probabilities that pixels are located inside a region closed by the partial borders. Such methods do not always find acceptable regions but they are useful in many practical situations.

Algorithm 6.16: Region forming from partial borders

1. For each border pixel $\mathbf{x}$, search for an opposite edge pixel within a distance not exceeding a given maximum M. If an opposite edge pixel is not found, process the next border pixel in the image. If an opposite edge pixel is found, mark each pixel on the connecting straight line as a potential region member.

2. Compute the number of markers for each pixel in the image (the number of markers tells how often a pixel was on a connecting line between opposite edge pixels). Let $b(\mathbf{x})$ be the number of markers for the pixel $\mathbf{x}$.

3. The weighted number of markers $B(\mathbf{x})$ is then determined as follows:

$$\begin{aligned} B(\mathbf{x}) &= 0.0 \quad \text{for } b(\mathbf{x}) = 0\,, \\ &= 0.1 \quad \text{for } b(\mathbf{x}) = 1\,, \\ &= 0.2 \quad \text{for } b(\mathbf{x}) = 2\,, \\ &= 0.5 \quad \text{for } b(\mathbf{x}) = 3\,, \\ &= 1.0 \quad \text{for } b(\mathbf{x}) > 3\,. \end{aligned} \tag{6.28}$$

The confidence that a pixel $\mathbf{x}$ is a member of a region is given as the sum $\sum_i B(\mathbf{x_i})$ in a 3×3 neighborhood of the pixel $\mathbf{x}$. If the confidence that a pixel $\mathbf{x}$ is a region member is one or larger, then pixel $\mathbf{x}$ is marked as a region pixel, otherwise it is marked as a background pixel.

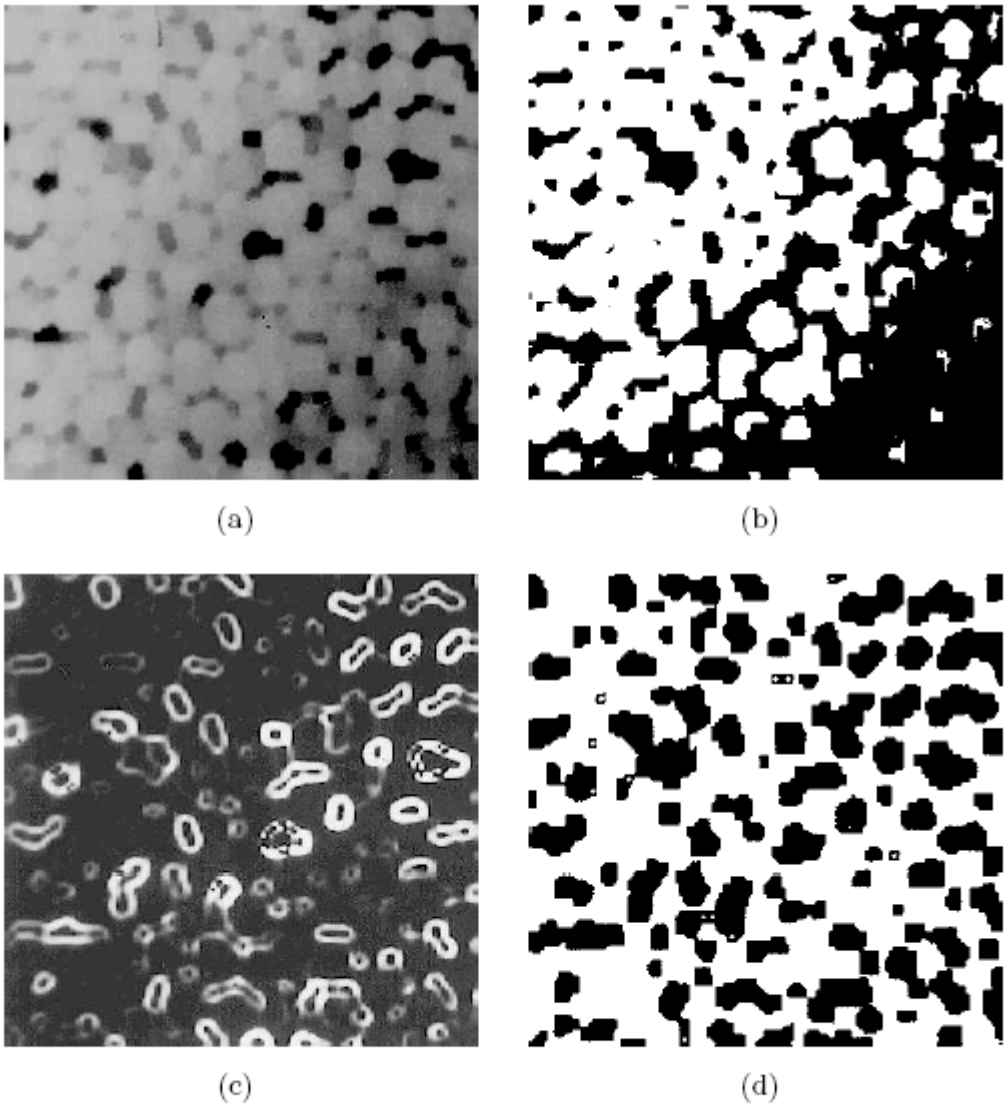


Figure 6.41: Region forming from partial borders. (a) Original image. (b) Thresholding. (c) Edge image. (d) Regions formed from partial borders.

Region-based segmentation

Algorithm 6.17: Region merging (outline)

1. Define some starting method to segment the image into many small regions satisfying condition (6.30).
2. Define a criterion for merging two adjacent regions.
3. Merge all adjacent regions satisfying the merging criterion. If no two regions can be merged maintaining condition (6.30), stop.

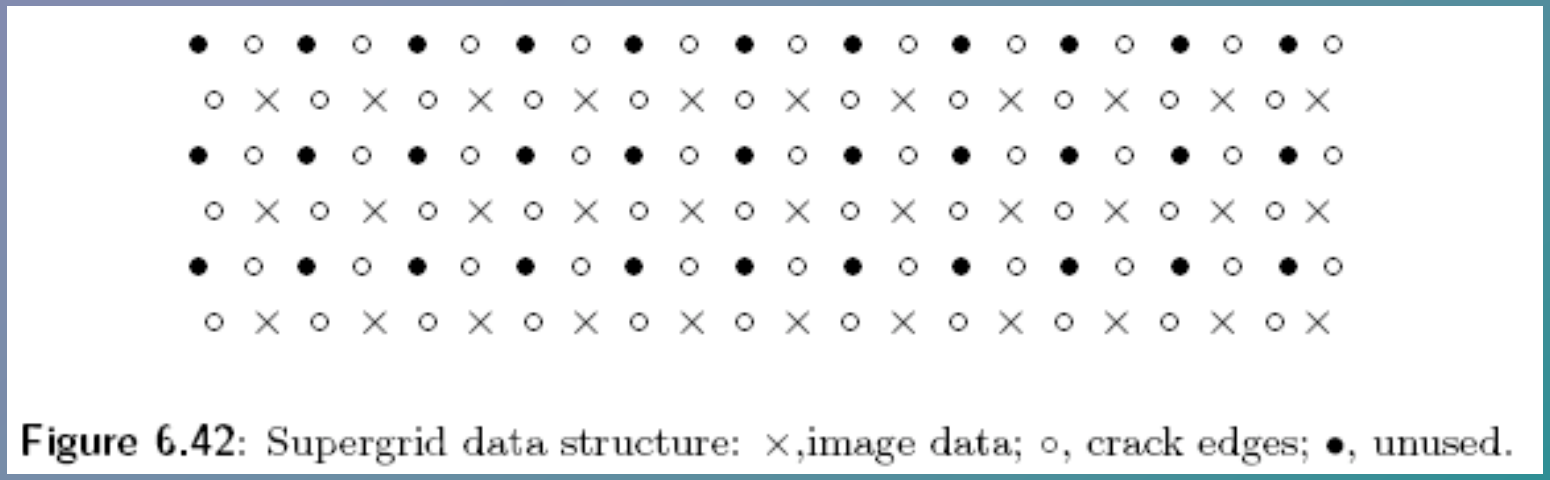


Figure 6.42: Supergrid data structure: ×,image data; ∘, crack edges; •, unused.

Algorithm 6.18: Region merging via boundary melting

1. Define a starting image segmentation into regions of constant gray-level. Construct a supergrid edge data structure in which to store the crack edge information.
2. Remove all weak crack edges from the edge data structure (using equation (6.32) and threshold T_1).
3. Recursively remove common boundaries of adjacent regions R_i, R_j, if

$$\frac{W}{\min(l_i, l_j)} \geq T_2 \,,$$

   where W is the number of weak edges on the common boundary, l_i, l_j are the perimeter lengths of regions R_i, R_j, and T_2 is another preset threshold.
4. Recursively remove common boundaries of adjacent regions R_i, R_j if

$$\frac{W}{l} \geq T_3 \tag{6.33}$$

   or, using a weaker criterion [Ballard and Brown, 1982]

$$W \geq T_3 \,, \tag{6.34}$$

   where l is the length of the common boundary and T_3 is a third threshold.

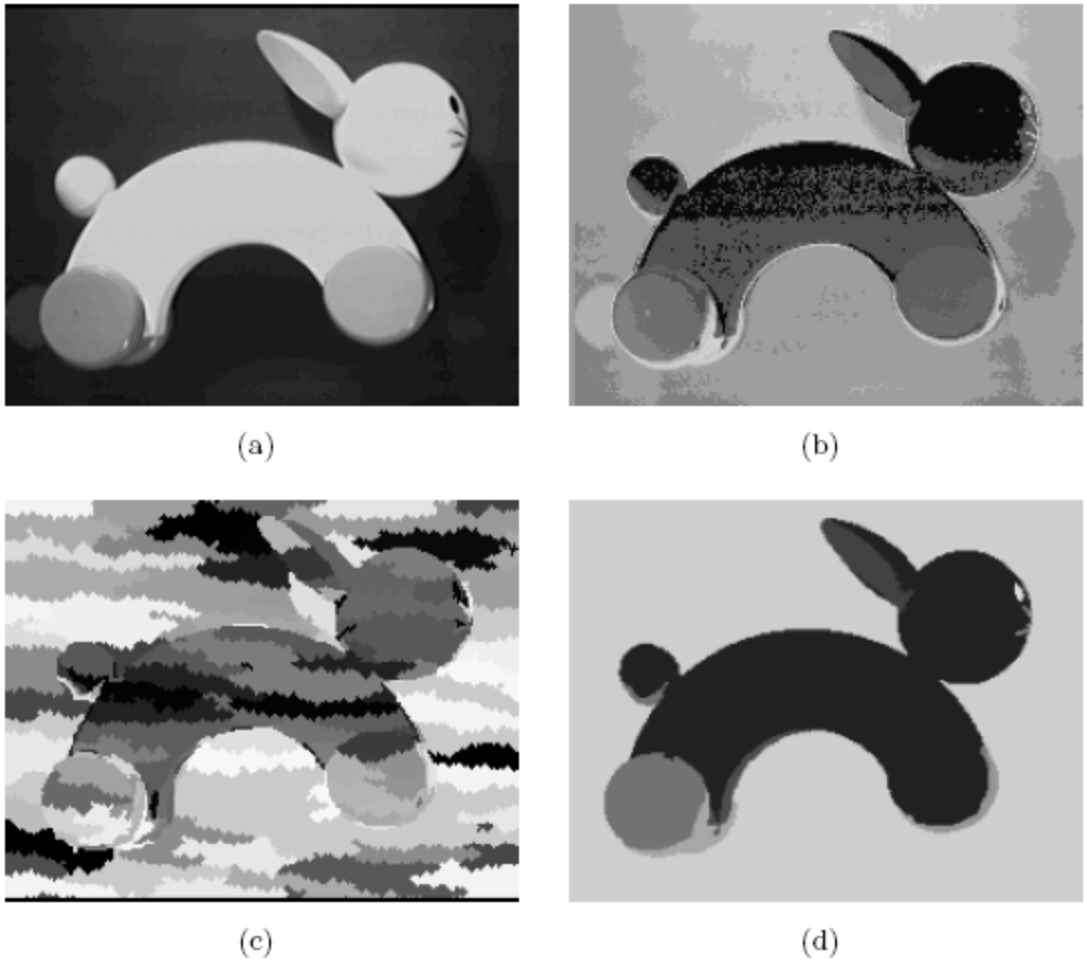


Figure 6.43: Region merging segmentation. (a) Original image. (b) Pseudo-color representation of the original image (in grayscale). (c) Recursive region merging. (d) Region merging via boundary melting. *Courtesy of R. Marik, Czech Technical University.*

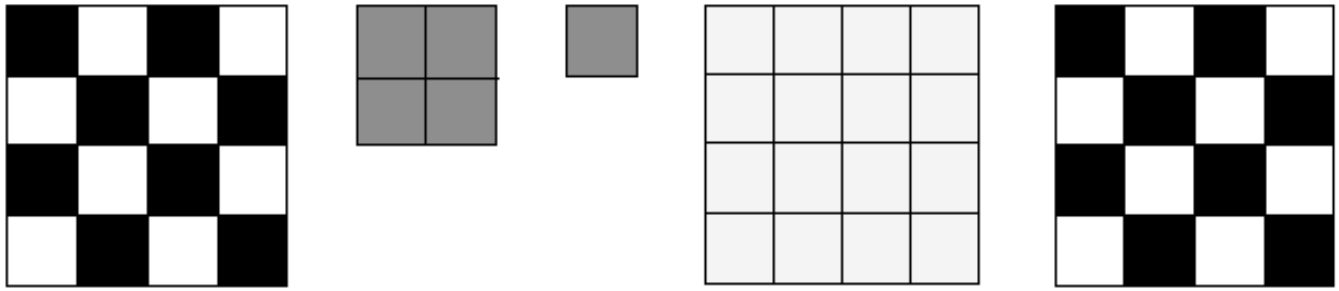


Figure 6.44: Different segmentations may result from region splitting and region merging approaches. (a) Chessboard image, corresponding pyramid. (b) Region splitting segmentation (upper pyramid level is homogeneous, no splitting possible). (c) Region merging segmentation (lowest pyramid level consists of regions that cannot be merged).

Figure 6.45: Split-and-merge in a hierarchical data structure.

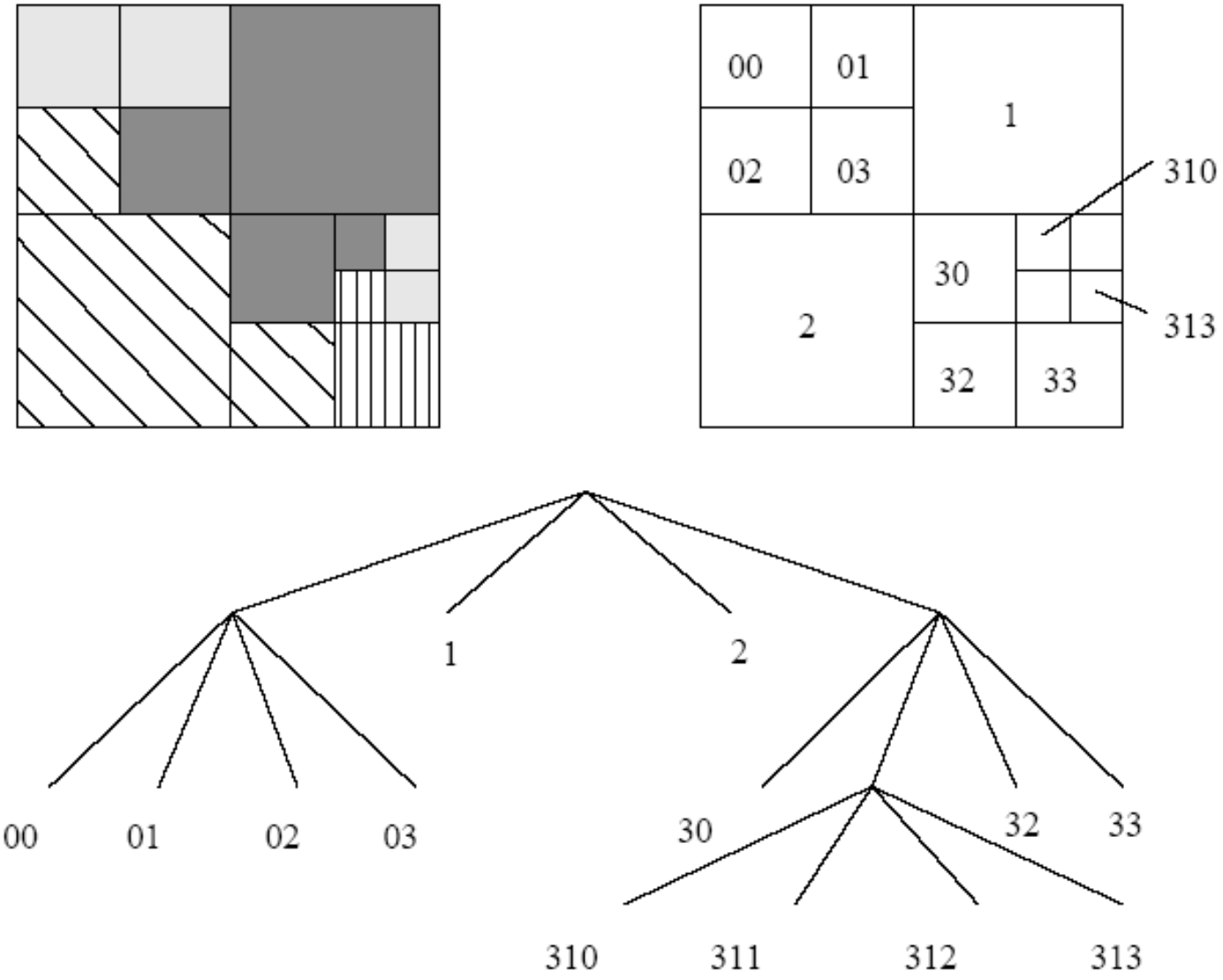


Figure 6.46: Segmentation quadtree.

Algorithm 6.19: Split and merge

1. Define an initial segmentation into regions, a homogeneity criterion, and a pyramid data structure.
2. If any region R in the pyramid data structure is not homogeneous [$H(R) = \text{FALSE}$], split it into four child-regions; if any four regions with the same parent can be merged into a single homogeneous region, merge them. If no region can be split or merged, go to step 3.
3. If any two adjacent regions R_i, R_j (even if they are in different pyramid levels or do not have the same parent) can be merged into a homogeneous region, merge them.
4. Merge small regions with the most similar adjacent region if it is necessary to remove small-size regions.

Algorithm 6.20: Split and link to the segmentation tree

1. Define a pyramid data structure with overlapping regions. Evaluate the starting region description.
2. Build a segmentation tree starting with leaves. Link each node of the tree to that one of the four possible parents to which it has the most similar region properties. Build the whole segmentation tree. If there is no link to an element in the higher pyramid level, assign the value zero to this element.
3. Update the pyramid data structure; each element must be assigned the average of the values of all its existing children.
4. Repeat steps 2 and 3 until no significant segmentation changes appear between iterations (a small number of iterations is usually sufficient).

Algorithm 6.21: Single-pass split-and-merge

1. Search an entire image line by line except the last column and last line. Perform the following steps for each pixel.
2. Find a splitting pattern for a 2×2 pixel block.
3. If a mismatch between assigned labels and splitting patterns in overlapping blocks is found, try to change the assigned labels of these blocks to remove the mismatch (discussed below).
4. Assign labels to unassigned pixels to match a splitting pattern of the block.
5. Remove small regions if necessary.

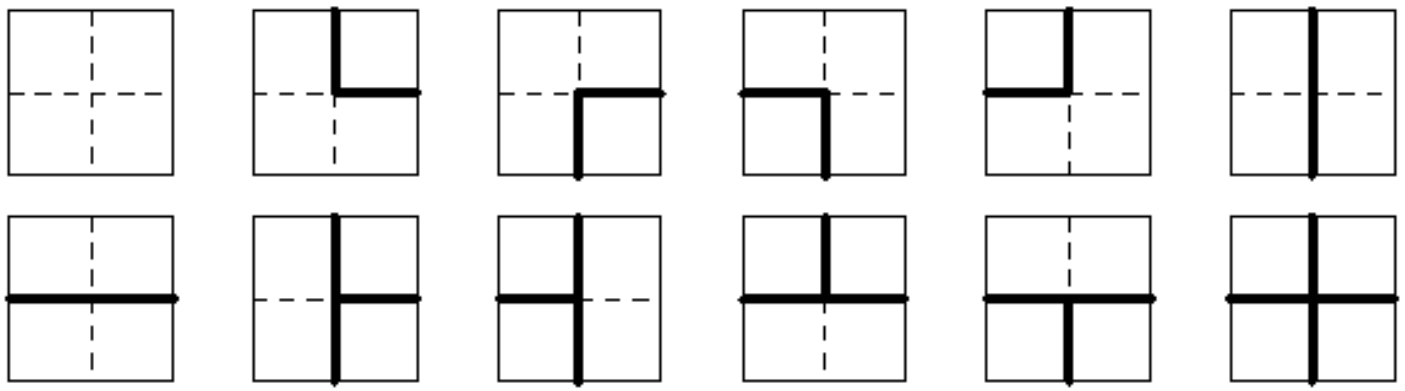


Figure 6.47: Splitting of 2×2 image blocks, all 12 possible cases.

Watershed segmentation

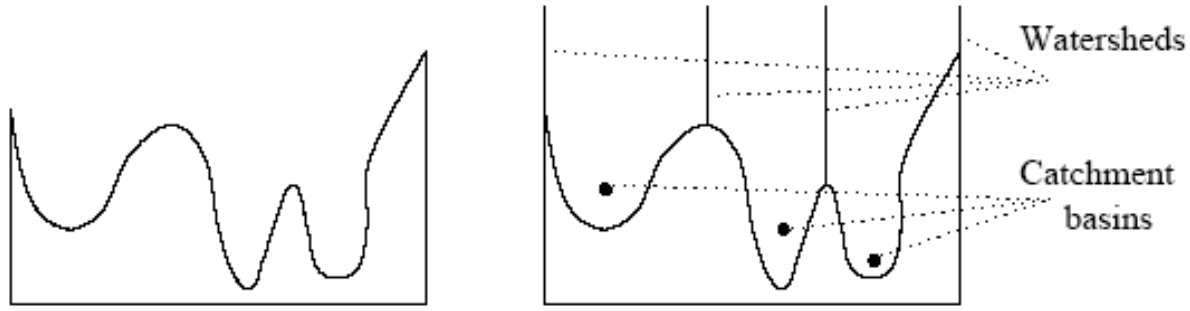


Figure 6.48: One-dimensional example of watershed segmentation. (a) Gray-level profile of image data. (b) Watershed segmentation—local minima of gray-level (altitude) yield catchment basins, local maxima define the watershed lines.

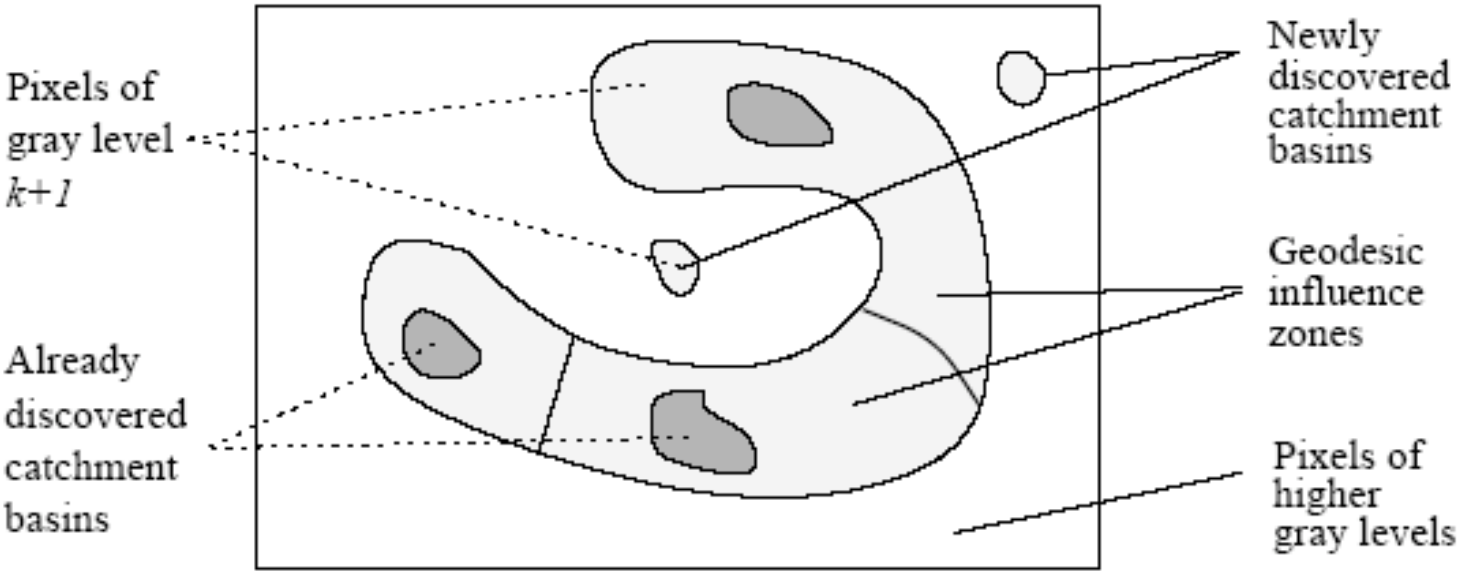


Figure 6.49: Geodesic influence zones of catchment basins.

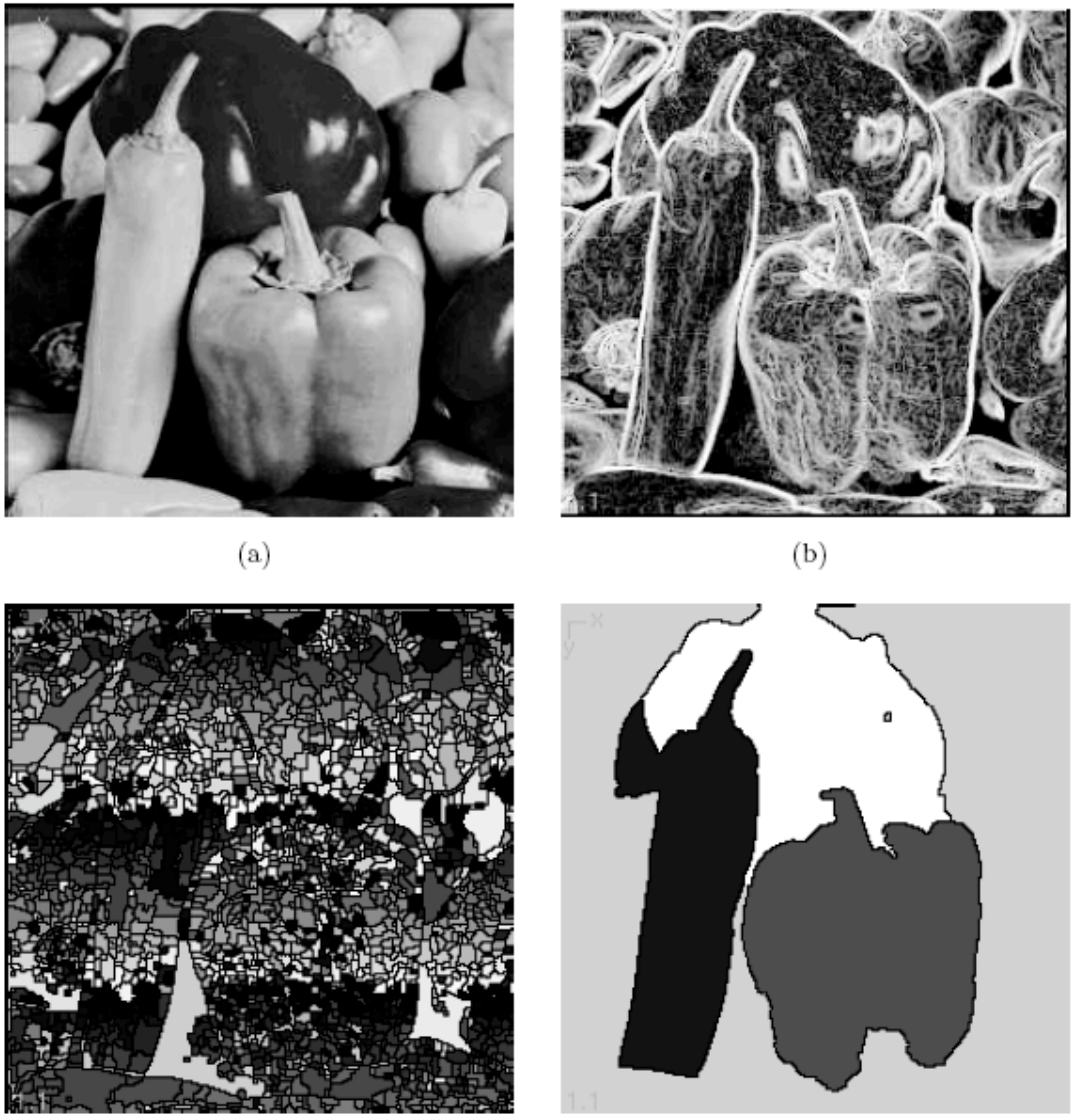


Figure 6.50: Watershed segmentation. (a) Original;. (b) Gradient image, 3×3 Sobel edge detection, histogram equalized. (c) Raw watershed segmentation. (d) Watershed segmentation using region markers to control oversegmentation. *Courtesy of W. Higgins, Penn State University.*

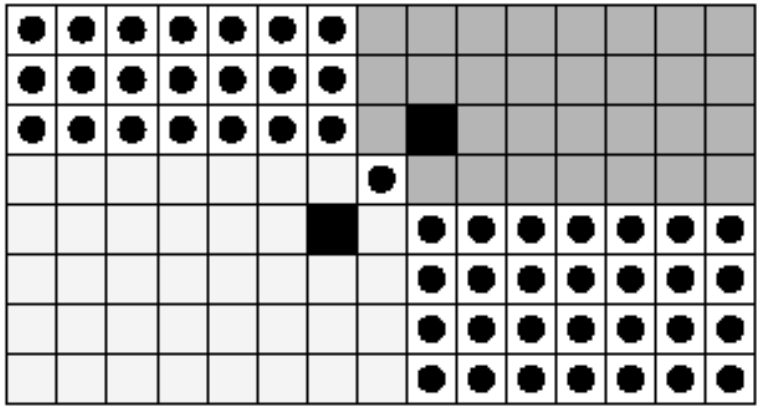


Figure 6.51: Thick watershed lines may result in gray-level plateaus. Earlier identified catchment basins are marked as black pixels, and new catchment basin additions resulting from this processing step are shown in the two levels of gray. The thick watersheds are marked with •. To avoid thick watersheds, specialized rules must be developed.

Region growing post-processing

Algorithm 6.22: Removal of small image regions

1. Search for the smallest image region R_{min}.
2. Find the adjacent region R most similar to R_{min}, according to the homogeneity criteria used. Merge R and R_{min}.
3. Repeat steps 1 and 2 until all regions smaller than a pre-selected size are removed from the image.

Summary

Image segmentation

- The main goal of image segmentation is to divide an image into parts that have a strong correlation with objects or areas of the real world depicted in the image.
- Segmentation methods can be divided into three groups: **thresholding**, **edgebased** segmentation and **region-based** segmentation.
- Each region can be represented by its closed boundary, and each closed boundary describes a region.
- Image data ambiguity is one of the main segmentation problems, often accompanied by information noise.
- The more a priori information is available to the segmentation process, the better the segmentation results that can be obtained.

Summary

Region-based image segmentation

- **Region growing** segmentation should satisfy the following condition of complete segmentation (6.1)

$$R = \bigcup_{i=1}^{S} R_i\,, \qquad R_i \cap R_j = \emptyset\,, \qquad i \neq j\,,$$

and the maximum region homogeneity conditions (6.30), (6.31)

$$H(R_i) = \text{TRUE}, \quad i = 1, 2, \ldots, S\,,$$
$$H(R_i \cup R_j) = \text{FALSE}, \quad i \neq j\,, \quad R_i \text{ adjacent to } R_j\,.$$

- Three basic approaches to region growing exist: **region merging**, **region splitting**, and **split-and-merge** region growing.
- **Region merging** starts with an oversegmented image in which regions satisfy equation (6.30). Regions are merged to satisfy condition (6.31) as long as equation (6.30) remains satisfied.

Summary

Region-based image segmentation cont.

- **Region splitting** is the opposite of region merging. Region splitting begins with an undersegmented image which does not satisfy condition (6.30). Therefore, the existing image regions are sequentially split to satisfy conditions (6.1), (6.30), and (6.31).
- A combination of **splitting and merging** may result in a method with the advantages of both other approaches. Split-and-merge approaches typically use pyramid image representations. Because both split-and-merge processing options are available, the starting segmentation does not have to satisfy either condition (6.30) or (6.31).

Summary

Region-based image segmentation cont.

- In **watershed** segmentation, catchment basins represent the regions of the segmented image. The first watershed segmentation approach starts with finding a downstream path from each pixel of the image to local minima of image surface altitude. A catchment basin is then defined as the set of pixels for which their respective downstream paths all end up in the same altitude minimum. In the second approach, each gray-level minimum represents one catchment basin and the strategy is to start filling the catchment basins from the bottom.

Summary

Region-based image segmentation cont.

- Images segmented by region growing methods often contain either too many regions (under-growing) or too few regions (over-growing) as a result of nonoptimal parameter setting. To improve classification results, a variety of **postprocessors** has been developed. Simpler post-processors decrease the number of small regions in the segmented image. More complex post-processing may combine segmentation information obtained from region growing and edge-based segmentation.

Matching

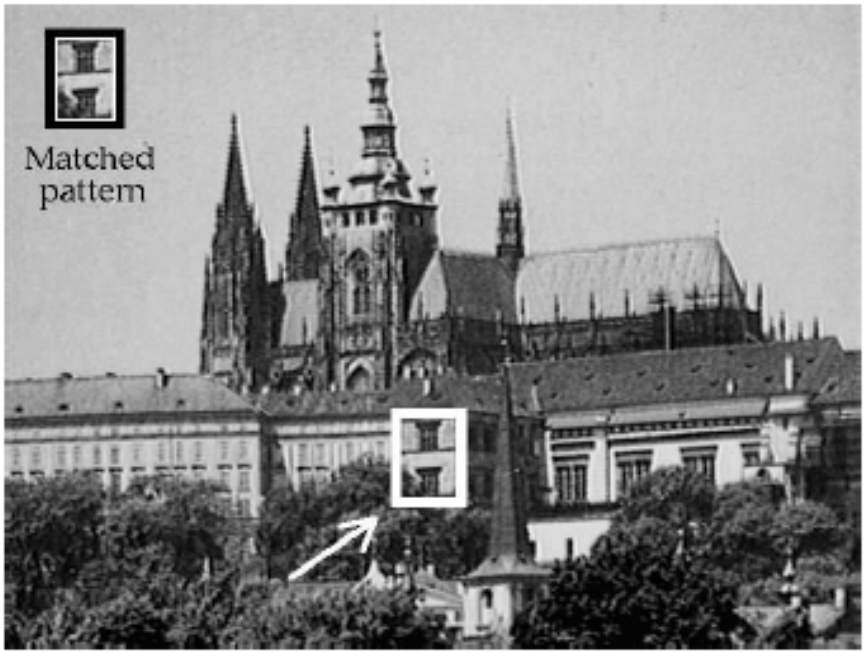


Figure 6.52: Segmentation by matching; matched pattern and location of the best match.

Algorithm 6.23: Match-based segmentation

1. Evaluate a match criterion for each location and rotation of the pattern in the image.
2. Local maxima of this criterion exceeding a preset threshold represent pattern locations in the image.

$$\begin{vmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 \end{vmatrix}$$

(a)

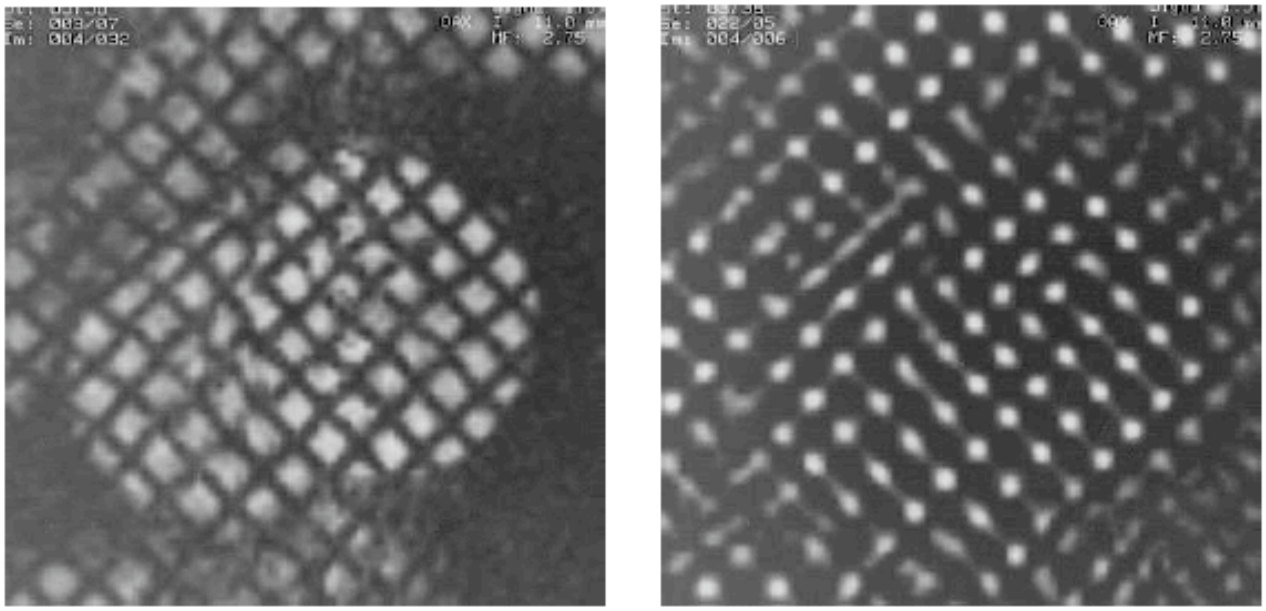
$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

(b)

$$\begin{vmatrix} \underline{\mathbf{1/3}} & 1/6 & 1/8 & \times & \times \\ 1/5 & 1/7 & 1/8 & \times & \times \\ 1/8 & 1/9 & 1/57 & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{vmatrix}$$

(c)

Figure 6.53: Optimality matching criterion evaluation: (a) image data; (b) matched pattern; (c) values of the optimality criterion C_3 (the best match underlined).



(a) (b)

Figure 6.54: X-shaped mask matching. (a) Original image (see also Figure 16.21). (b) Correlation image; the better the local correlation with the X-shaped mask, the brighter the correlation image. *Courtesy of D. Fisher, S. Collins, The University of Iowa.*

Summary

Matching

- Matching can be used to locate objects of known appearance in an image, to search for specific patterns, etc. The best match is based on some criterion of optimality which depends on object properties and object relations.
- Matching criteria can be defined in many ways; in particular, correlation between a pattern and the searched image data is often used as a general matching criterion.
- Chamfer matching may be used to locate one-dimensional features that might otherwise defeat cost-based optimality approaches.

Evaluation issues

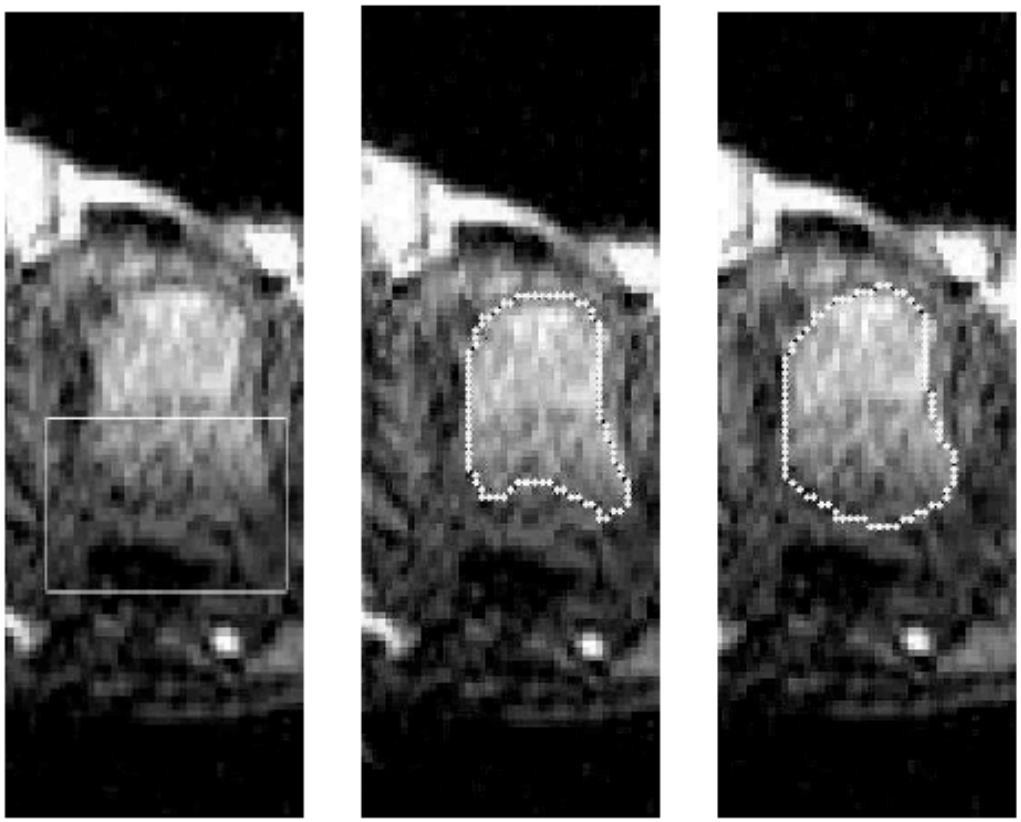


Figure 6.55: A region from a dynamically enhanced MRI study with partly ambiguous boundary. Two different experts have overlaid their judgments. *Courtesy of O. Kubassova, S. Tanner, University of Leeds.*

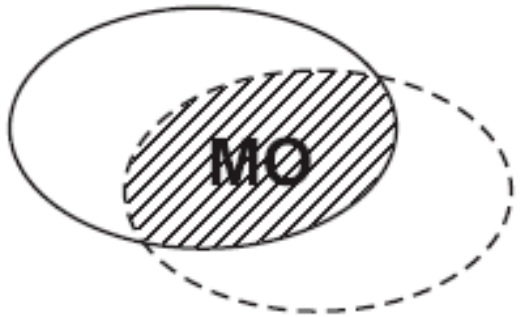


Figure 6.56: Mutual overlap: machine segmented region in solid, ground truth in dashed.

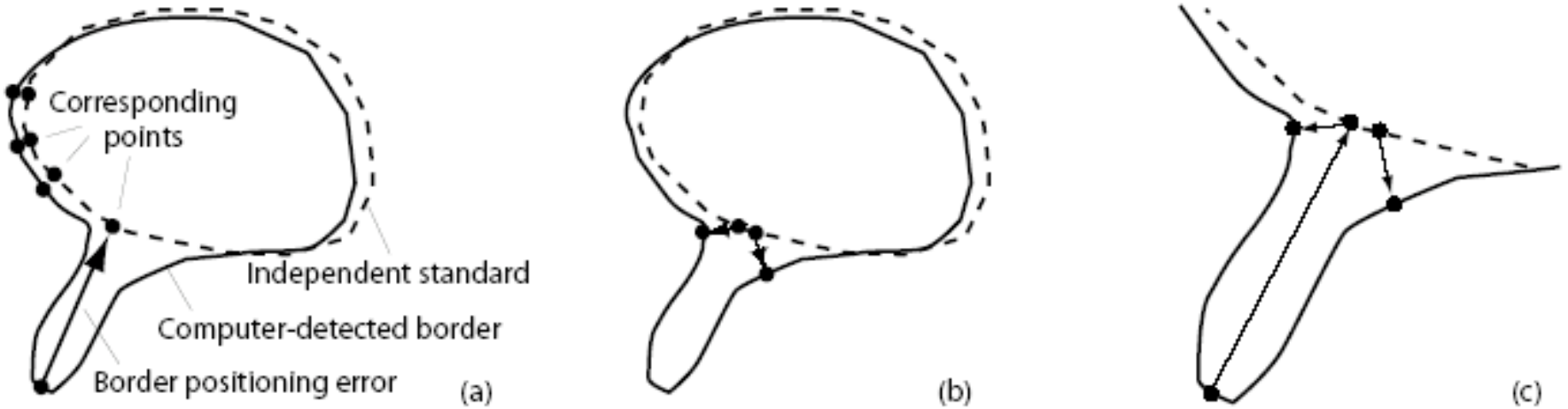
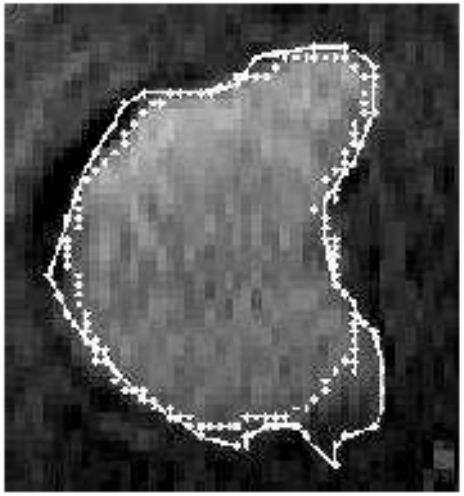


Figure 6.57: Border positioning errors. (a) Assuming that ground truth is known, border positioning errors are computed as directed distances between the computer-determined and correct borders. (b) If the errors are calculated in the opposite direction (from ground truth to the computer-determined border), a substantially different answer may result. (c) Zoomed area showing the difference in calculating directional errors.



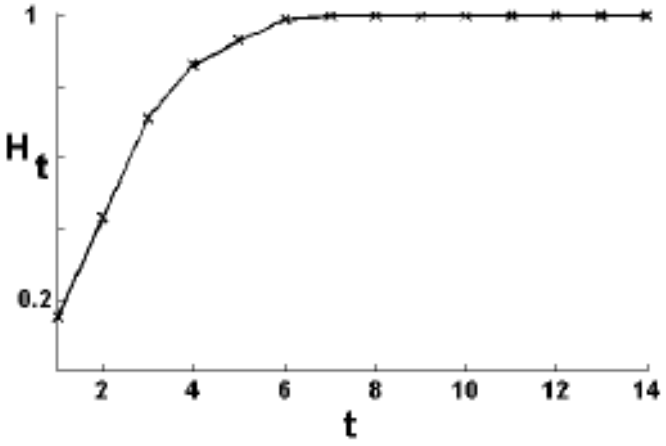


Figure 6.58: Use of the H_t metric on a segmentation from dynamically enhanced MRI; ground truth is dotted and a sample segmentation is solid. *Courtesy of O. Kubassova, University of Leeds.*

Summary

Evaluation

- Evaluation of segmentation is useful in deciding between algorithms, or parameter choice for a given algorithm.
- *Supervised* evaluation compares algorithm output to ground truth.
- Supervised approaches usually compare area overlap, or distance between boundaries—there are several ways of doing this.
- Ground truth is often poorly defined or expensive to extract. *Unsupervised* approaches judge segmentations in ignorance of it.
- Many unsupervised approaches exist but they are usually constrained by assumptions about image regions.