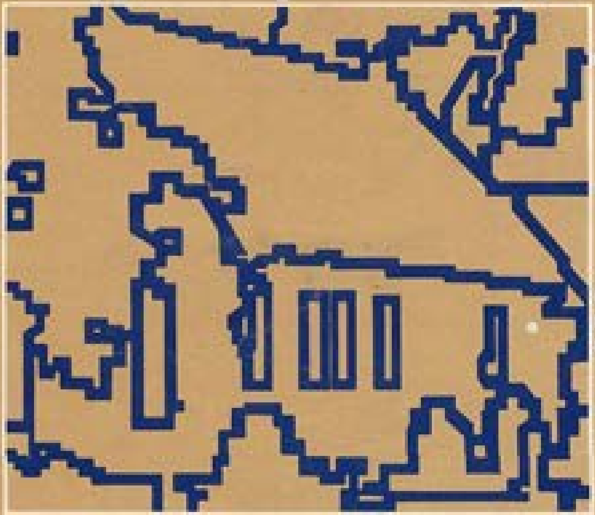
COMPUTER VISION

DANA H. BALLARD • CHRISTOPHER M. BROWN



COMPUTER VISION

Dana H. Ballard
Christopher M. Brown

Department of Computer Science
University of Rochester
Rochester, New York

PRENTICE-HALL, INC., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

BALLARD, DANA HARRY.
Computer vision.

Bibliography: p.
Includes index.
1. Image processing. I. Brown, Christopher M.
II. Title.
TA1632.B34 621.38'0414 81-20974
ISBN 0-13-165316-4 AACR2

Cover design by Robin Breite

Printed in the United States of America

10 9 8 7 6 5 4 3 2

ISBN 0-13-165316-4

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

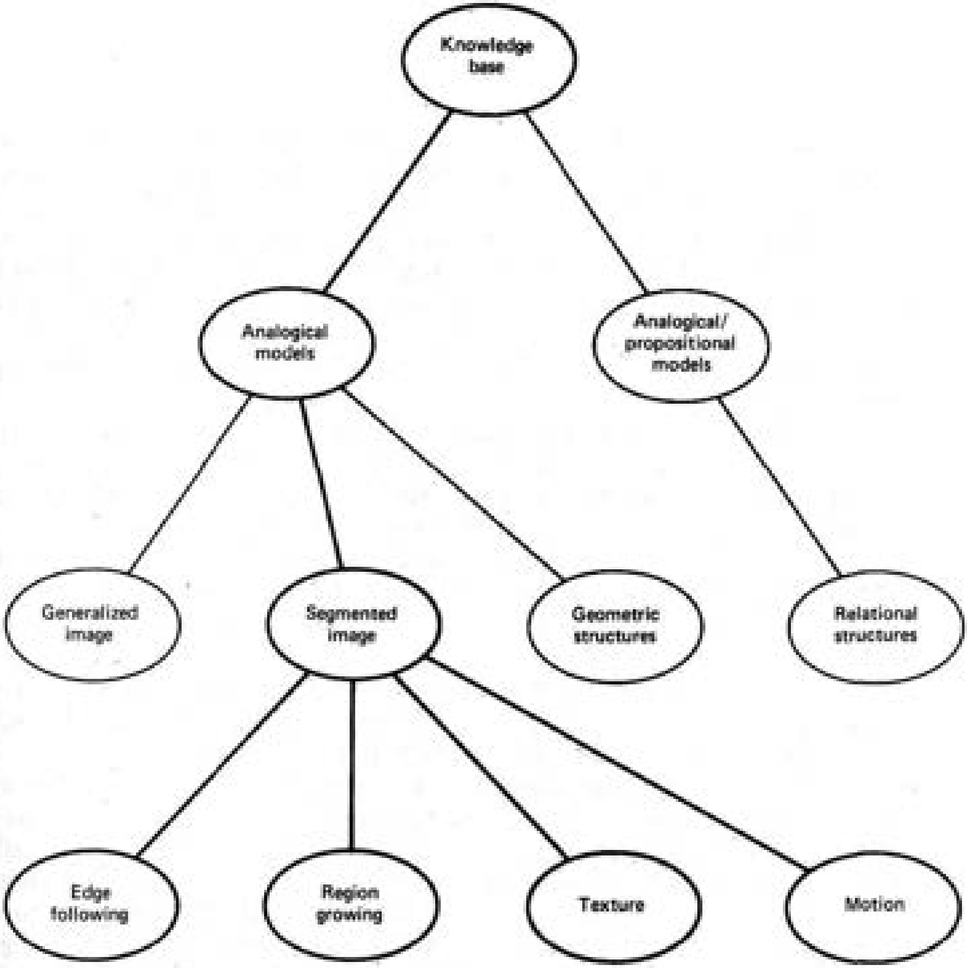
Contents

PART II
SEGMENTED IMAGES
115

Part III
GEOMETRICAL STRUCTURES
227

SEGMENTED IMAGES

II



The idea of segmentation has its roots in work by the Gestalt psychologists (e.g., Kohler), who studied the preferences exhibited by human beings in grouping or organizing sets of shapes arranged in the visual field. Gestalt principles dictate certain grouping preferences based on features such as proximity, similarity, and continuity. Other results had to do with figure/ground discrimination and optical illusions. The latter have provided a fertile ground for vision theories to post-Gestaltists such as Gibson and Gregory, who emphasize that these grouping mechanisms organize the scene into *meaningful units* that are a significant step toward image understanding.

In computer vision, grouping parts of a generalized image into units that are homogeneous with respect to one or more characteristics (or features) results in a *segmented image.* The segmented image extends the generalized image in a crucial respect: it contains the beginnings of domain-dependent interpretation. At this descriptive level the internal domain-dependent models of objects begin to influence the grouping of generalized image structures into units meaningful in the domain. For instance, the model may supply crucial parameters to segmentation procedures.

In the segmentation process there are two important aspects to consider: one is the data structure used to keep track of homogeneous groups of features; the other is the transformation involved in computing the features.

Two basic sorts of segments are natural: boundaries and regions. These can be used combined into a single descriptive structure, a set of nodes (one per region), connected by arcs representing the "adjacency" relation. The "dual" of this structure has arcs corresponding to boundaries connecting nodes representing points where several regions meet. Chapters 4 and 5 describe segmentation with respect to boundaries and regions respectively, emphasizing gray levels and gray-level differences as indicators of segments. Of course, from the standpoint of the algorithms involved, it is irrelevant whether the features are intensity gray levels or intrinsic image values perhaps representing motion, color, or range.

Texture and motion images are addressed in Chapters 6 and 7. Each has several computationally difficult aspects, and neither has received the attention given static, nontextured images. However, each is very important in the segmentation enterprise.

Boundary Detection 4

4.1 ON ASSOCIATING EDGE ELEMENTS

Boundaries of objects are perhaps the most important part of the hierarchy of structures that links raw image data with their interpretation [Marr 1975]. Chapter 3 described how various operators applied to raw image data can yield primitive edge elements. However, an image of only disconnected edge elements is relatively featureless; additional processing must be done to group edge elements into structures better suited to the process of interpretation. The goal of the techniques in this chapter is to perform a level of *segmentation*, that is, to make a coherent one-dimensional (*edge*) feature from many individual local edge elements. The feature could correspond to an object boundary or to any meaningful boundary between scene entities. The problems that edge-based segmentation algorithms have to contend with are shown by Fig. 4.1, which is an image of the local edge elements yielded by one common edge operator applied to a chest radiograph. As can be seen, the edge elements often exist where no meaningful scene boundary does, and conversely often are absent where a boundary is. For example, consider the boundaries of ribs as revealed by the edge elements. Missing edge elements and extra edge elements both tend to frustrate the segmentation process.

The methods in this chapter are ordered according to the amount of knowledge incorporated into the grouping operation that maps edge elements into boundaries. "Knowledge" means implicit or explicit constraints on the likelihood of a given grouping. Such constraints may arise from general physical arguments or (more often) from stronger restrictions placed on the image arising from domain-dependent considerations. If there is much knowledge, this implies that the global form of the boundary and its relation to other image structures is very constrained. Little prior knowledge means that the segmentation must proceed more on the basis of local clues and evidence and general (domain-dependent) assumptions with fewer expectations and constraints on the final resulting boundary.

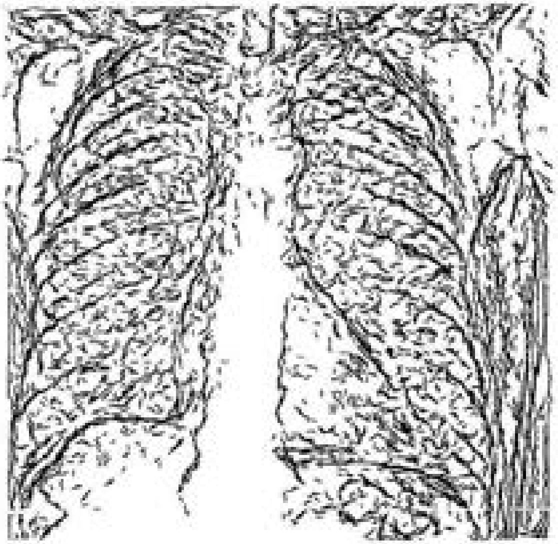


Fig. 4.1 Edge elements in a chest radiograph.

These constraints take many forms. Knowledge of where to expect a boundary allows very restricted searches to verify the edge. In many such cases, the domain knowledge determines the type of curve (its parameterization or functional form) as well as the relevant "noise processes." In images of polyhedra, only straight-edged boundaries are meaningful, and they will come together at various sorts of vertices arising from corners, shadows of corners, and occlusions. Human rib boundaries appear approximately like conic sections in chest radiographs, and radiographs have complex edge structures that can compete with rib edges. All this specific knowledge can and should guide our choice of grouping method.

If less is known about the specific image content, one may have to fall back on general world knowledge or heuristics that are true for most domains. For instance, in the absence of evidence to the contrary, the shorter line between two points might be selected over a longer line. This sort of general principle is easily built into evaluation functions for boundaries, and used in segmentation algorithms that proceed by methodically searching for such groupings. If there are no a priori restrictions on boundary shapes, a general contour-extraction method is called for, such as edge following or linking of edge elements.

The methods we shall examine are the following:

1. *Searching near an approximate location.* These are methods for refining a boundary given an initial estimate.
2. *The Hough transform.* This elegant and versatile technique appears in various guises throughout computer vision. In this chapter it is used to detect boundaries whose shape can be described in an analytical or tabular form.
3. *Graph searching.* This method represents the image of edge elements as a graph. Thus a boundary is a path through a graph. Like the Hough transform, these techniques are quite generally applicable.

4. *Dynamic programming.* This method is also very general. It uses a mathematical formulation of the globally best boundary and can find boundaries in noisy images.
5. *Contour following.* This hill-climbing technique works best with good image data.

4.2 SEARCHING NEAR AN APPROXIMATE LOCATION

If the approximate or a priori likely location of a boundary has been determined somehow, it may be used to guide the effort to refine that boundary [Kelly 1971]. The approximate location may have been found by one of the techniques below applied to a lower resolution image, or it may have been determined using high-level knowledge.

4.2.1 Adjusting A Priori Boundaries

This idea was described by [Bolles 1977] (see Fig. 4.2). Local searches are carried out at regular intervals along directions perpendicular to the approximate (a priori) boundary. An edge operator is applied to each of the discrete points along each of these perpendicular directions. For each such direction, the edge with the highest magnitude is selected from among those whose orientations are nearly parallel to the tangent at the point on the nearby a priori boundary. If sufficiently many elements are found, their locations are fit with an analytic curve such as a low-degree polynomial, and this curve becomes the representation of the boundary.

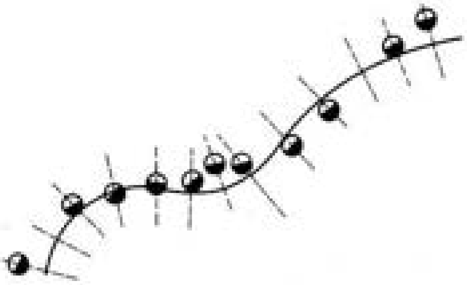


Fig. 4.2 Search orientations from an approximate boundary location.

4.2.2 Non-linear Correlation in Edge Space

In this correlation-like technique, the a priori boundary is treated as a rigid template, or piece of rigid wire along which edge operators are attached like beads. The a priori representation thus also contains relative locations at which the existence of edges will be tested (Fig. 4.3). An edge element returned by the edge-operator application "matches" the a priori boundary if its contour is tangent to the template and its magnitude exceeds some threshold. The template is to be moved around the image, and for each location, the number of matches is computed. If the number of matches exceeds a threshold, the boundary location is declared to

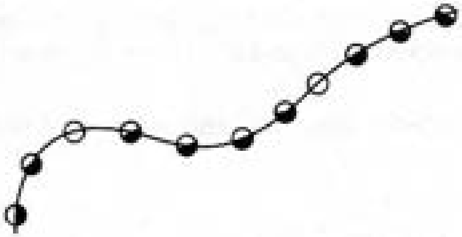


Fig. 4.3 A template for edge-operator application.

be the current template location. If not, the template is moved to a different image point and the process is repeated. Either the boundary will be located or there will eventually be no more image points to try.

4.2.3 Divide-and-Conquer Boundary Detection

This is a technique that is useful in the case that a low-curvature boundary is known to exist between two edge elements and the noise levels in the image are low (Algorithm 8.1). In this case, to find a boundary point in between the two known points, search along the perpendiculars of the line joining the two points. The point of maximum magnitude (if it is over some threshold) becomes a break point on the boundary and the technique is applied recursively to the two line segments formed between the three known boundary points. (Some fix must be applied if the maximum is not unique.) Figure 4.4 shows one step in this process. Divide-and-conquer boundary detection has been used to outline kidney boundaries on computed tomograms (these images were described in Section 2.3.4) [Selfridge et al. 1979].

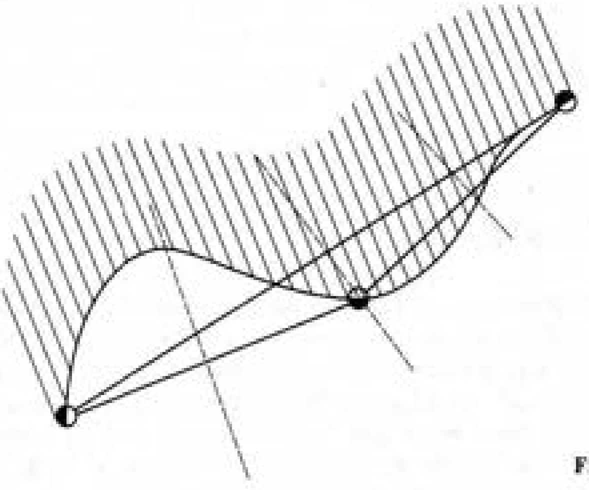


Fig. 4.4 Divide and conquer technique.

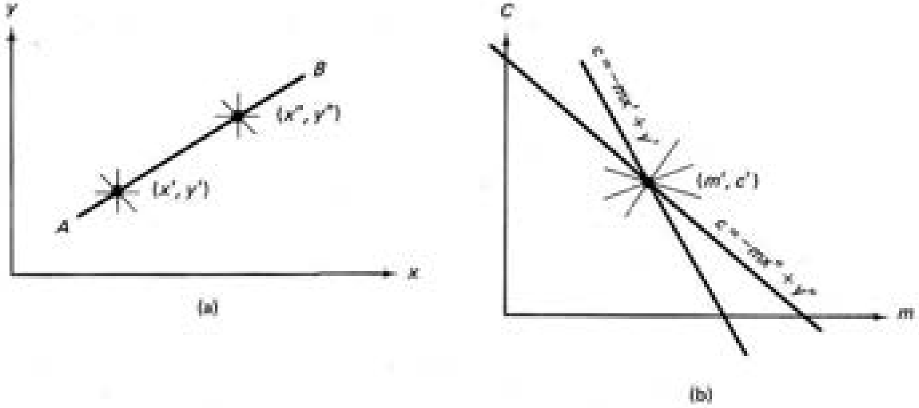


Fig. 4.5 A line (a) in image space; (b) in parameter space.

4.3 THE HOUGH METHOD FOR CURVE DETECTION

The classical Hough technique for curve detection is applicable if little is known about the location of a boundary, but its shape can be described as a parametric curve (e.g., a straight line or conic). Its main advantages are that it is relatively unaffected by gaps in curves and by noise.

To introduce the method [Duda and Hart 1972], consider the problem of detecting straight lines in images. Assume that by some process image points have been selected that have a high likelihood of being on linear boundaries. The Hough technique organizes these points into straight lines, basically by considering all possible straight lines at once and rating each on how well it explains the data.

Consider the point $\mathbf{x}'$ in Fig. 4.5a, and the equation for a line $y = mx + c$. What are the lines that could pass through $\mathbf{x}'$? The answer is simply all the lines with m and c satisfying $y' = mx' + c$. Regarding (x', y') as fixed, the last equation is that of a line in $m-c$ space, or parameter space. Repeating this reasoning, a second point $(\mathbf{x}'', \mathbf{y}'')$ will also have an associated line in parameter space and, furthermore, these lines will intersect at the point (m', c') which corresponds to the line AB connecting these points. In fact, all points on the line AB will yield lines in parameter space which intersect at the point (m', c'), as shown in Fig. 4.5b.

This relation between image space $\mathbf{x}$ and parameter space suggests the following algorithm for detecting lines:

Algorithm 4.1: Line Detection with the Hough Algorithm

1. Quantize parameter space between appropriate maximum and minimum values for c and m.
2. Form an accumulator array $A(c, m)$ whose elements are initially zero.
3. For each point (x,y) in a *gradient* image such that the strength of the gradient

exceeds some threshold, increment all points in the accumulator array along the appropriate line, i.e.,

$$A(c, m) := A(c, m) + 1$$

for m and c satisfying $c = -mx + y$ within the limits of the digitization.

4. Local maxima in the accumulator array now correspond to collinear points in the image array. The values of the accumulator array provide a measure of the number of points on the line.

This technique is generally known as the Hough technique [Hough 1962].

Since m may be infinite in the slope-intercept equation, a better parameterization of the line is $x \sin\theta + y\cos\theta = r$. This produces a sinusoidal curve in (r, θ) space for fixed x, y, but otherwise the procedure is unchanged.

The generalization of this technique to other curves is straightforward and this method works for any curve $f(\mathbf{x}, \mathbf{a}) = 0$, where a is a parameter vector. (In this chapter we often use the symbol f as various general functions unrelated to the image gray-level function.) In the case of a circle parameterized by

$$(x-a)^2 + (y-b)^2 = r^2 \tag{4.1}$$

for fixed $\mathbf{x}$, the modified algorithm 4.1 increments values of a, b, r lying on the surface of a cone. Unfortunately, the computation and the size of the accumulator array increase exponentially as the number of parameters, making this technique practical only for curves with a small number of parameters.

The Hough method is an efficient implementation of a generalized matched filtering strategy (i.e., a template-matching paradigm). For instance, in the case of a circle, imagine a template composed of a circle of 1's (at a fixed radius R) and 0's everywhere else. If this template is convolved with the gradient image, the result is the portion of the accumulator array $A(a, b, R)$.

In its usual form, the technique yields a set of parameters for a curve that best explains the data. The parameters may specify an infinite curve (e.g., a line or parabola). Thus, if a finite curve segment is desired, some further processing is necessary to establish end points.

4.3.1 Use of the Gradient

Dramatic reductions in the amount of computation can be achieved if the gradient direction is integrated into the algorithm [Kimme et al. 1975]. For example, consider the problem of detecting a circle of fixed radius R.

Without gradient information, all values a, b lying on the circle given by (4.1) are incremented. With the gradient direction, only the points near (a,b) in Fig. 4.6 need be incremented. From geometrical considerations, the point (a,b) is given by

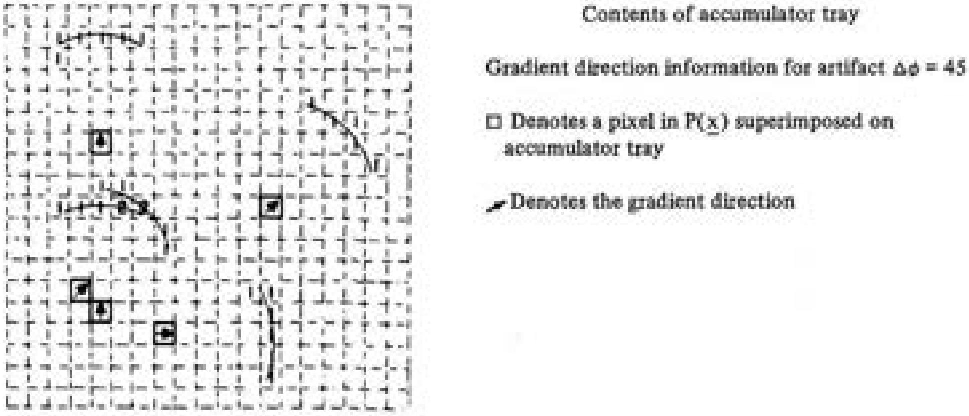

Contents of accumulator tray
Gradient direction information for artifact Δϕ = 45
□ Denotes a pixel in P(x) superimposed on accumulator tray
Denotes the gradient direction

Fig 4.6 Reduction in computation with gradient information

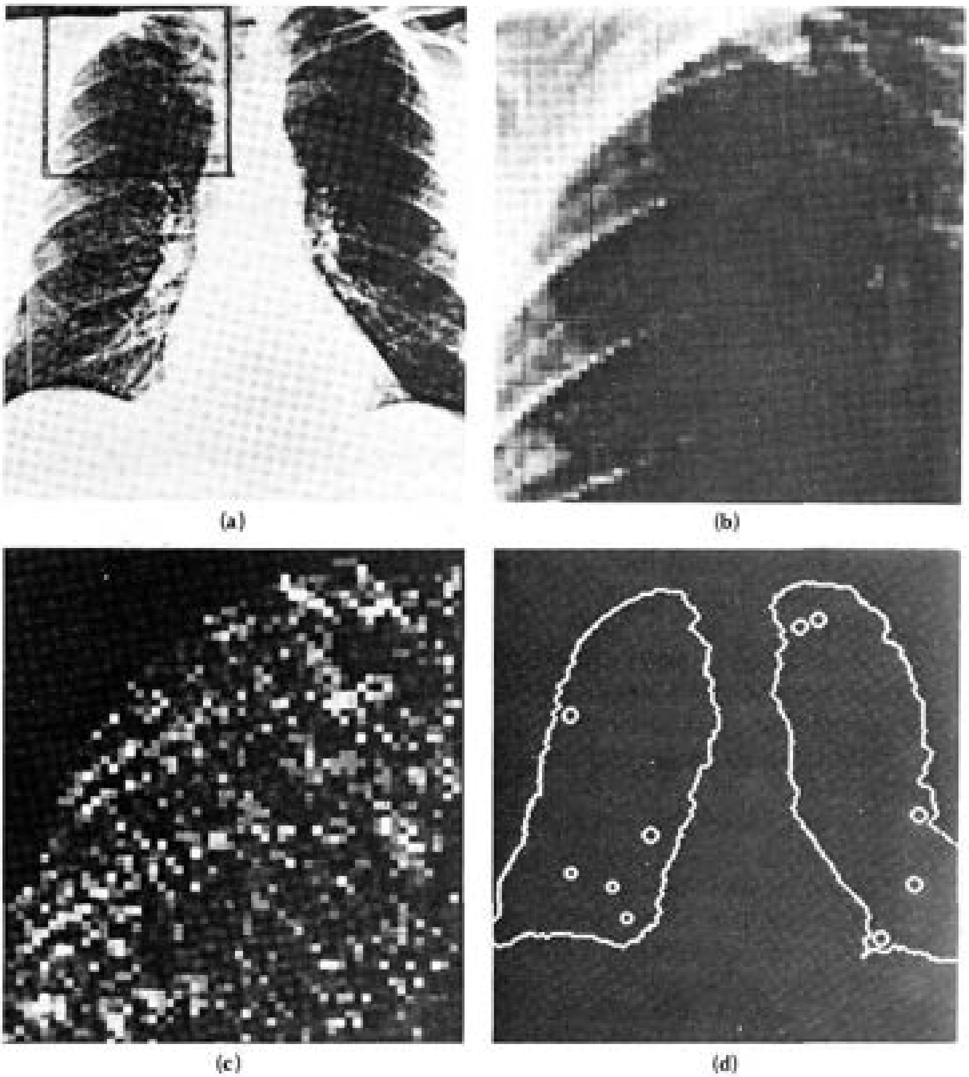
$$a = x - r \sin \phi \qquad (4.2)$$
$$b = y + r \cos \phi$$

where $\phi(x)$ is the gradient angle returned by an edge operator. Implicit in these equations is the assumption that the circle is the boundary of a disk that has gray levels greater than its surroundings. These equations may also be derived by differentiating (4.2), recognizing that $dy/dx = \tan\phi$, and solving for a and b between the resultant equation and (4.2). Similar methods can be applied to other conics. In each case, the use of the gradient saves one dimension in the accumulator array.

The gradient magnitude can also be used as a heuristic in the incrementing procedure. Instead of incrementing by unity, the accumulator array location may be incremented by a function of the gradient magnitude. This heuristic can balance the magnitude of brightness change across a boundary with the boundary length, but it can lead to detection of phantom lines indicated by a few bright points, or to missing dim but coherent boundaries.

4.3.2 Some Examples

The Hough technique has been used successfully in a variety of domains. Some examples include the detection of human hemoglobin fingerprints [Ballard et al. 1975], the detection of tumors in chest films [Kimme et al. 1975], the detection of storage tanks in aerial images [Lantz et al. 1978], and the detection of ribs in chest radiographs [Wechsler and Sklansky 1977]. Figure 4.7 shows the tumor-detection application. A section of the chest film (Fig. 4.7b) is searched for disks of radius 3 units. In Fig. 4.7c, the resultant accumulator array $A[a, b, 3]$ is shown in a pictoral fashion, by interpreting the array values as gray levels. This process is repeated for various radii and then a set of likely circles is chosen by setting a radius-dependent threshold for the accumulator array contents. This result is shown in Fig. 4.7d. The



(a)

(b)

(c)

(d)

Fig. 4.7 Using the Hough technique for circular shapes. (a) Radiograph. (b) Window. (c) Accumulator array for $r = 3$. (d) Results of maxima detection.

circular boundaries detected by the Hough technique are overlaid on the original image.

4.3.3 Trading Off Work in Parameter Space for Work in Image Space

Consider the example of detecting ellipses that are known to be oriented so that a principal axis is parallel to the x axis. These can be specified by four parameters. Using the equation for the ellipse together with its derivative, and substituting for the known gradient as before, one can solve for two parameters. In the equation

$$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}=1 \tag{4.3}$$

x is an edge point and x_0, y_0, a, and b are parameters. The equation for its derivative is

$$\frac{(x-x_0)}{a}+\frac{(y-y_0)^2}{b^2}\frac{dy}{dx}=0 \tag{4.4}$$

where $dy/dx = \tan\phi(x)$. The Hough algorithm becomes:

Algorithm 4.2: Hough technique applied to ellipses

For each discrete value of x and y, increment the point in parameter space given by a, b, x_0, y_0, where

$$x = x_0 \pm \frac{a}{(1 + b^2/a^2 \tan^2\phi)^{1/2}} \tag{4.5}$$

$$y = y_0 \pm \frac{b}{(1 + a^2 \tan^2\phi/b^2)^{1/2}} \tag{4.6}$$

that is,

$$A(a, b, x_0, y_0) := A(a, b, x_0, y_0) + 1$$

For a and b each having m values the computational cost is proportional to m^2.

Now suppose that we consider all pairwise combinations of edge elements. This introduces two additional equations like (4.3) and (4.4), and now the four-parameter point can be determined exactly. That is, the following equations can be solved for a unique x_0, y_0, a, b.

$$\frac{(x_1-x_0)^2}{a^2}+\frac{(y_1-y_0)^2}{b^2}=1 \tag{4.7a}$$

$$\frac{(x_2-x_0)^2}{a^2}+\frac{(y_2-y_0)^2}{b^2}=1 \tag{4.7b}$$

$$\frac{x_1-x_0}{a^2}+\frac{y_1-y_0}{b^2}\frac{dy}{dx}=0 \tag{4.7c}$$

$$\frac{x_2-x_0}{a^2}+\frac{y_2-y_0}{b^2}\frac{dy}{dx}=0 \tag{4.7d}$$

$$\frac{dy}{dx}=\tan\phi \qquad \left(\frac{dy}{dx} \text{ is known from the edge operator}\right)$$

Their solution is left as an exercise. The amount of effort in the former case was proportional to the product of the number of discrete values of a and b, whereas this case involves effort proportional to the square of the number of edge elements.

4.3.4 Generalizing the Hough Transform

Consider the case where the object being sought has no simple analytic form, but has a particular silhouette. Since the Hough technique is so closely related to template matching, and template matching can handle this case, it is not surprising that the Hough technique can be generalized to handle this case also. Suppose for the moment that the object appears in the image with known shape, orientation, and scale. (If orientation and scale are unknown, they can be handled in the same way that additional parameters were handled earlier.) Now pick a reference point in the silhouette and draw a line to the boundary. At the boundary point compute the gradient direction and store the reference point as a function of this direction. Thus it is possible to precompute the location of the reference point from boundary points given the gradient angle. The set of all such locations, indexed by gradient angle, comprises a table termed the *R*-table [Ballard 1981]. Remember that the basic strategy of the Hough technique is to compute the possible loci of reference points in parameter space from edge point data in image space and increment the parameter points in an accumulator array. Figure 4.8 shows the relevant geometry and Table 4.1 shows the form of the *R*-table. For the moment, the reference point coordinates (x_c, y_c) are the only parameters (assuming that rotation and scaling have been fixed). Thus an edge point (x, y) with gradient orientation ϕ constrains the possible reference points to be at $\{x + r_1(\phi) \cos [\alpha_1(\phi)],\ y + r_1(\phi) \sin [\alpha_1(\phi)]\}$ and so on.

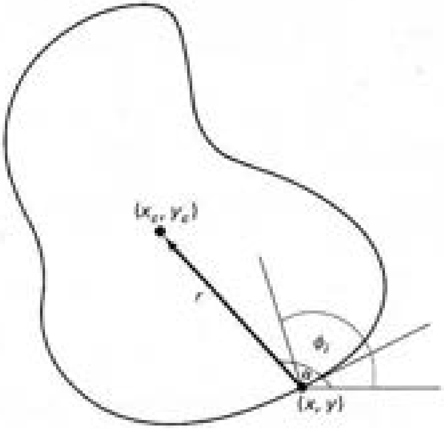


Fig. 4.8 Geometry used to form the *R*-Table.

Table 4.1

INCREMENTATION IN THE GENERALIZED HOUGH CASE

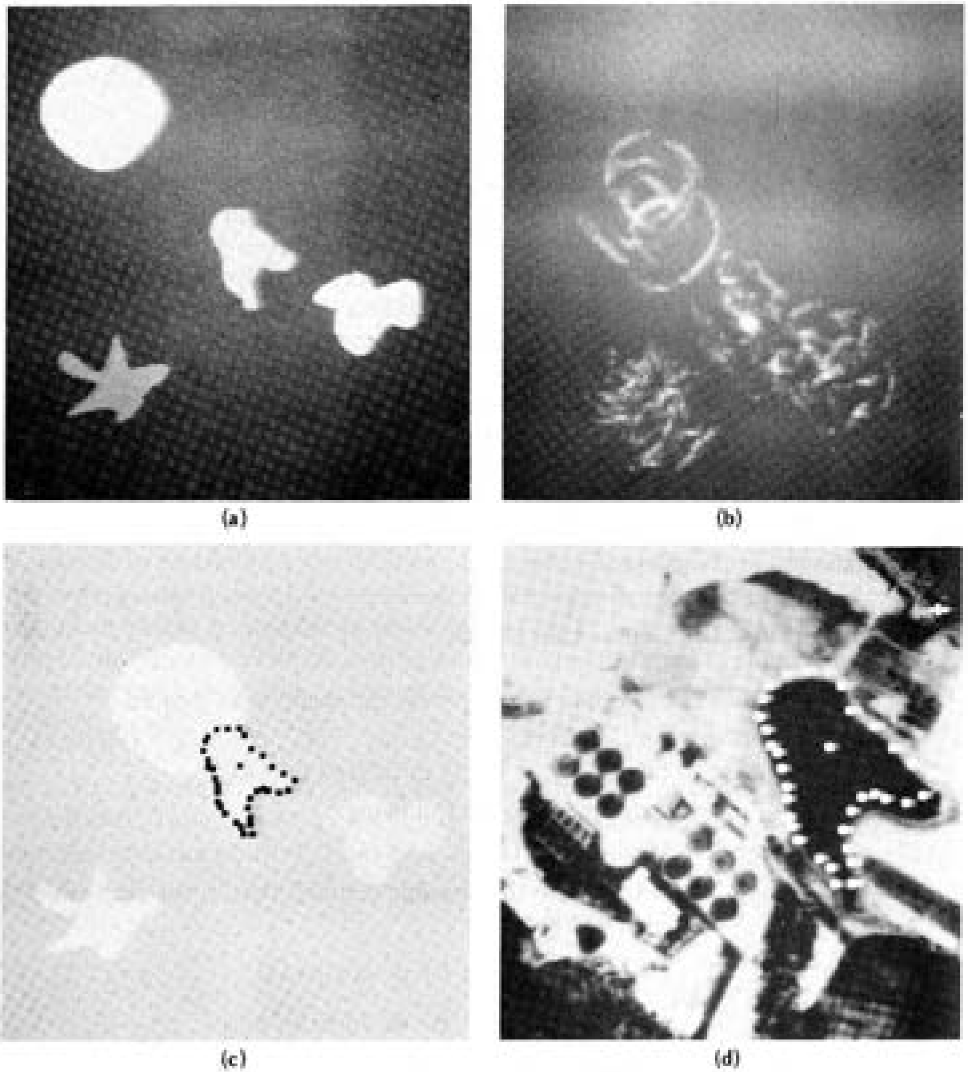
Angle measured from figure boundary to reference point	*Set of radii* $\{\mathbf{r}^k\}$ *where* $\mathbf{r} = (r, \alpha)$
ϕ_1	$\mathbf{r}_1^1, \mathbf{r}_2^1, \ldots, \mathbf{r}_{n_1}^1$
ϕ_2	$\mathbf{r}_1^2, \mathbf{r}_2^2, \ldots, \mathbf{r}_{n_2}^2$
.	.
.	.
.	.
ϕ_m	$\mathbf{r}_1^m, \mathbf{r}_2^m, \ldots, \mathbf{r}_{n_m}^m$

The generalized Hough algorithm may be described as follows:

Algorithm 4.3: Generalized Hough

```
Step 0. Make a table (like Table 4.1) for the shape to be located.
Step 1. Form an accumulator array of possible reference points
        $A(x_{c\min}:x_{c\max}, y_{c\min}:y_{c\max})$ initialized to zero.
Step 2. For each edge point do the following:
    Step 2.1.  Compute $\phi(\mathbf{x})$
    Step 2.2a. Calculate the possible centers; that is, for each table entry for
               $\phi$, compute
                   $x_c := x + r\ \phi\ \cos[\alpha(\phi)]$
                   $y_c := y + r\ \phi\ \sin[\alpha(\phi)]$
    Step 2.2b. Increment the accumulator array
                   $A(x_c, y_c) := A(x_c, y_c) + 1$
Step 3. Possible locations for the shape are given by maxima in array $A$.
```

The results of using this transform to detect a shape are shown in Fig. 4.9. Figure 4.9a shows an image of shapes. The *R*-table has been made for the middle shape. Figure 4.9b shows the Hough transform for the shape, that is, $A(x_c, y_c)$ displayed as an image. Figure 4.9c shows the shape given by the maxima of



(a)

(b)

(c)

(d)

Fig. 4.9 Applying the Generalized Hough technique. (a) Synthetic image. (b) Hough Transform $A(x_c, y_c)$ for middle shape. (c) Detected shape. (d) Same shape in an aerial image setting.

$A(x_c, y_c)$ overlaid on top of the image. Finally, Fig. 4.9d shows the Hough transform used to detect a pond of the same shape in an aerial image.

What about the parameters of scale and rotation, S and θ? These are readily accommodated by expanding the accumulator array and doing more work in the incrementation step. Thus in step 1 the accumulator array is changed to

$$(x_{c\min} : x_{c\max}, y_{c\min} : y_{c\max}, S_{\min} : S_{\max}, \theta_{\min} : \theta_{\max})$$

and step 2.2a is changed to

```
for each table entry for $\phi$ do
    for each $S$ and $\theta$
        $x_c := x + r(\phi) S \cos[\alpha(\phi) + \theta]$
        $y_c := y + r(\phi) S \sin[\alpha(\phi) + \theta]$
```

Finally, step 2.2b is now

$$A(x_c, y_c, S, \theta) := A(x_c, y_c, S, \theta) + 1$$

4.4 EDGE FOLLOWING AS GRAPH SEARCHING

A graph is a general object that consists of a set of nodes $\{n_i\}$ and arcs between nodes $<n_i, n_j>$. In this section we consider graphs whose arcs may have numerical weights or *costs* associated with them. The search for the boundary of an object is cast as a search for the lowest-cost path between two nodes of a weighted graph.

Assume that a gradient operator is applied to the gray-level image, creating the magnitude image $s(\mathbf{x})$ and direction image $\phi(\mathbf{x})$. Now interpret the elements of the direction image $\phi(\mathbf{x})$ as nodes in a graph, each with a weighting factor $s(\mathbf{x})$. Nodes $\mathbf{x}_i$, $\mathbf{x}_j$ have arcs between them if the contour directions $\phi(\mathbf{x}_i)$, $\phi(\mathbf{x}_j)$ are appropriately aligned with the arc directed in the same sense as the contour direction. Figure 4.10 shows the interpretation. To generate Fig. 4.10b impose the following restrictions. For an arc to connect from $\mathbf{x}_i$ to $\mathbf{x}_j$, $\mathbf{x}_j$ must be one of the three possible eight-neighbors in front of the contour direction $\phi(\mathbf{x}_i)$ and, furthermore, $g(\mathbf{x}_i)$

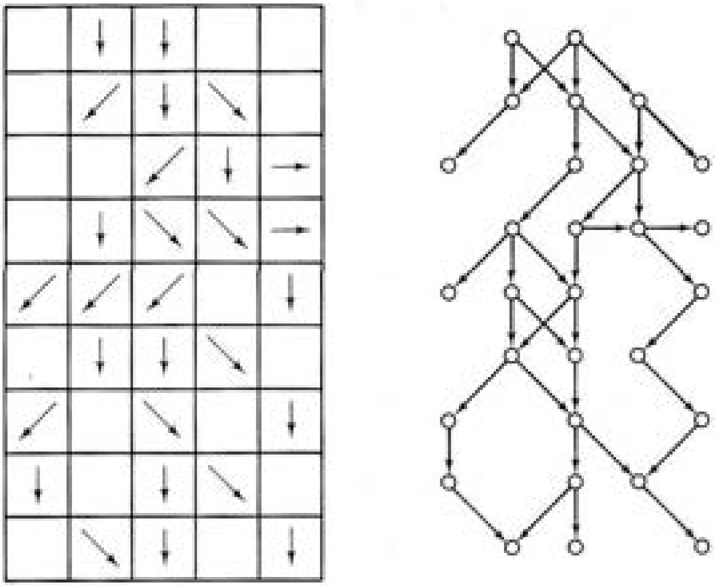


Fig. 4.10 Interpreting a gradient image as a graph (see text).

$> T$; $g(\mathbf{x}_j) > T$, where T is a chosen constant, and $|[[\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)] \bmod 2\pi]| < \pi/2$. (Any or all of these restrictions may be modified to suit the requirements of a particular problem.)

To generate a path in a graph from $\mathbf{x}_A$ to $\mathbf{x}_B$ one can apply the well-known technique of heuristic search [Nilsson 1971, 1980]. The specific use of heuristic search to follow edges in images was first proposed by [Martelli 1972]. Suppose:

1. That the path should follow contours that are directed from $\mathbf{x}_A$ to $\mathbf{x}_B$
2. That we have a method for generating the successor nodes of a given node (such as the heuristic described above)
3. That we have an evaluation function $f(\mathbf{x}_j)$ which is an estimate of the optimal cost path from $\mathbf{x}_A$ to $\mathbf{x}_B$ constrained to go through $\mathbf{x}_j$

Nilsson expresses $f(\mathbf{x}_i)$ as the sum of two components: $g(\mathbf{x}_i)$, the estimated cost of journeying from the *start node* $\mathbf{x}_A$ to $\mathbf{x}_i$, and $h(\mathbf{x}_i)$, the estimated cost of the path from $\mathbf{x}_i$ to $\mathbf{x}_B$, the *goal node*.

With the foregoing preliminaries, the heuristic search algorithm (called the A algorithm by Nilsson) can be stated as:

Algorithm 4.4: Heuristic Search (the A Algorithm)

1. "Expand" the start node (put the successors on a list called OPEN with pointers back to the start node).
2. Remove the node $\mathbf{x}_i$ of minimum f from OPEN. If $\mathbf{x}_i = \mathbf{x}_B$, then stop. Trace back through pointers to find optimal path. If OPEN is empty, fail.
3. Else expand node $\mathbf{x}_i$, putting successors on OPEN with pointers back to $\mathbf{x}_i$. Go to step 2.

The component $h(\mathbf{x}_i)$ plays an important role in the performance of the algorithm; if $h(\mathbf{x}_i) = 0$ for all i, the algorithm is a *minimum-cost search* as opposed to a *heuristic search*. If $h(\mathbf{x}_i) > h^*(\mathbf{x}_i)$ (the actual optimal cost), the algorithm may run faster, but may miss the minimum-cost path. If $h(\mathbf{x}_i) < h^*(\mathbf{x}_i)$, the search will always produce a minimum-cost path, provided that h also satisfies the following consistency condition:

> If for any two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$, $k(\mathbf{x}_i, \mathbf{x}_j)$ is the minimum cost of getting from $\mathbf{x}_i$ to $\mathbf{x}_j$ (if possible), then
>
> $$k(\mathbf{x}_i, \mathbf{x}_j) \geqslant h^*(\mathbf{x}_i) - h^*(\mathbf{x}_j)$$

With our edge elements, there is no guarantee that a path can be found since there may be insurmountable gaps between $\mathbf{x}_A$ and $\mathbf{x}_B$. If finding the edge is crucial, steps should be taken to interpolate edge elements prior to the search, or gaps may be crossed by using the edge element definition of [Martelli 1972]. He defines edges on the image grid structure so that an edge can have a direction even though there is no local gray-level change. This definition is depicted in Fig. 4.11a.

4.4.1 Good Evaluation Functions

A good evaluation function has components specific to the particular task as well as components that are relatively task-independent. The latter components are discussed here.

1. *Edge strength.* If edge strength is a factor, the cost of adding a particular edge element at **x** can be included as
$$M - s(\mathbf{x}) \qquad \text{where } M = \max_{\mathbf{x}} s(\mathbf{x})$$
2. *Curvature.* If low-curvature boundaries are desirable, curvature can be measured as some monotonically increasing function of
$$diff\,[\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)]$$
where diff measures the angle between the edge elements at $\mathbf{x}_j$ and $\mathbf{x}_i$.
3. *Proximity to an approximation.* If an approximate boundary is known, boundaries near this approximation can be favored by adding:
$$d = dist\,(\mathbf{x}_i, B)$$
to the cost measure. The dist operator measures the minimum distance of the new point $\mathbf{x}_i$ to the approximate boundary B.
4. *Estimates of the distance to the goal.* If the curve is reasonably linear, points near the goal may be favored by estimating h as $d(\mathbf{x}_i, \mathbf{x}_{\text{goal}})$, where d is a distance measure.

Specific implementations of these measures appear in [Ashkar and Modestino 1978; Lester et al. 1978].

4.4.2 Finding All the Boundaries

What if the objective is to find *all* boundaries in the image using heuristic search? In one system [Ramer 1975] Hueckel's operator (Chapter 3) is used to obtain

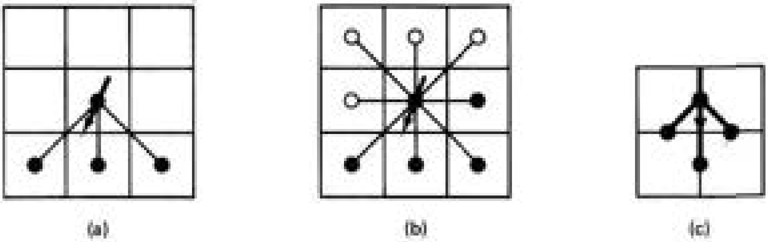


Fig. 4.11 Successor conventions in heuristic search (see text).

strokes, another name for the magnitude and direction of the local gray-level changes. Then these strokes are combined by heuristic search to form sequences of edge elements called *streaks*. Streaks are an intermediate organization which are used to assure a slightly broader coherence than is provided by the individual Hueckel edges. A bidirectional search is used with four eight-neighbors defined in front of the edge and four eight-neighbors behind the edge, as shown in Fig. 4.11b. The search algorithm is as follows:

1. Scan the stroke (edge) array for the most prominent edge.
2. Search in front of the edge until no more successors exist (i.e., a gap is encountered).
3. Search behind the edge until no more predecessors exist.
4. If the bidirectional search generates a path of 3 or more strokes, the path is a streak. Store it in a streak list and go to step 1.

Strokes that are part of a streak cannot be reused; they are marked when used and subsequently skipped.

There are other heuristic procedures for pruning the streaks to retain only *prime streaks*. These are shown in Fig. 4.12. They are essentially similar to the re-

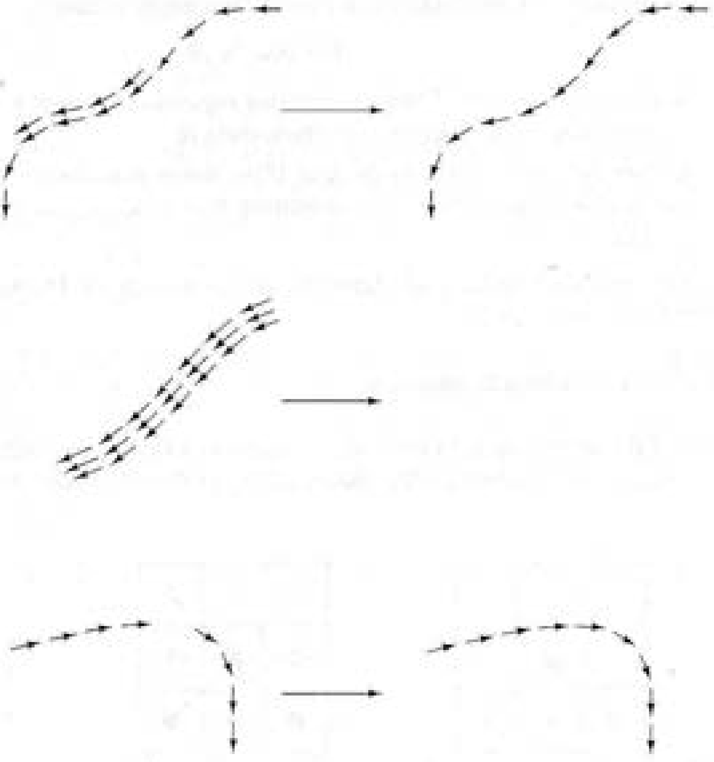


Fig. 4.12 Operations in the creation of prime streaks.

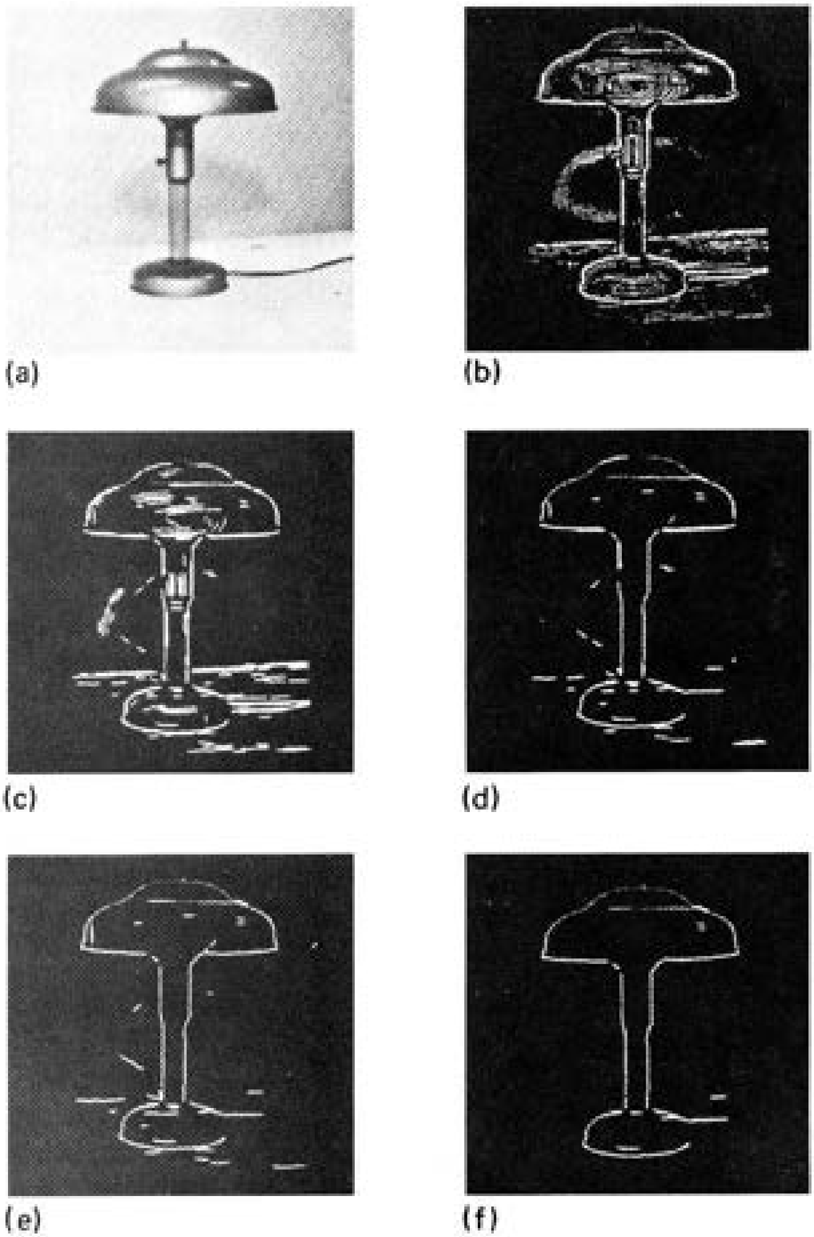


Fig. 4.13 Ramer's results.

laxation operations described in Section 3.3.5. The resultant streaks must still be analyzed to determine the objects they represent. Nevertheless, this method represents a cogent attempt to organize bottom-up edge following in an image. Fig. 4.13 shows an example of Ramer's technique.

4.4.3 Alternatives to the A Algorithm

The primary disadvantage with the heuristic search method is that the algorithm must keep track of a set of current best paths (nodes), and this set may become very large. These nodes represent tip nodes for the portion of the tree of possible paths that has been already examined. Also, since all the costs are nonnegative, a good path may eventually look expensive compared to tip nodes near the start node. Thus, paths from these newer nodes will be extended by the algorithm even though, from a practical standpoint, they are unlikely. Because of these disadvantages, other less rigorous search procedures have proven to be more practical, five of which are described below.

Pruning the Tree of Alternatives

At various points in the algorithm the tip nodes on the OPEN list can be pruned in some way. For example, paths that are short or have a high cost per unit length can be discriminated against. This pruning operation can be carried out whenever the number of alternative tip nodes exceeds some bound.

Modified Depth-First Search

Depth-first search is a meaningful concept if the search space is structured as a tree. Depth-first search means always evaluating the most recent expanded son. This type of search is performed if the OPEN list is structured as a stack in the A algorithm and the top node is always evaluated next. Modifications to this method use an evaluation function f to rate the successor nodes and expand the best of these. Practical examples can be seen in [Ballard and Sklansky 1976; Wechsler and Sklansky 1977; Persoon 1976].

Least Maximum Cost

In this elegant idea [Lester 1978], only the maximum-cost arc of each path is kept as an estimate of g. This is like finding a mountain pass at minimum altitude. The advantage is that g does not build up continuously with depth in the search tree, so that good paths may be followed for a long time. This technique has been applied to finding the boundaries of blood cells in optical microscope images. Some results are shown in Fig. 4.14.

Branch and Bound

The crux of this method is to have some upper bound on the cost of the path [Chien and Fu 1974]. This may be known beforehand or may be computed by actually generating a path between the desired end points. Also, the evaluation function must be monotonically increasing with the length of the path. With these conditions we start generating paths, excluding partial paths when they exceed the current bound.

Modified Heuristic Search

Sometimes an evaluation function that assigns negative costs leads to good results. Thus good paths keep getting better with respect to the evaluation function, avoiding the problem of having to look at all paths near the starting point.

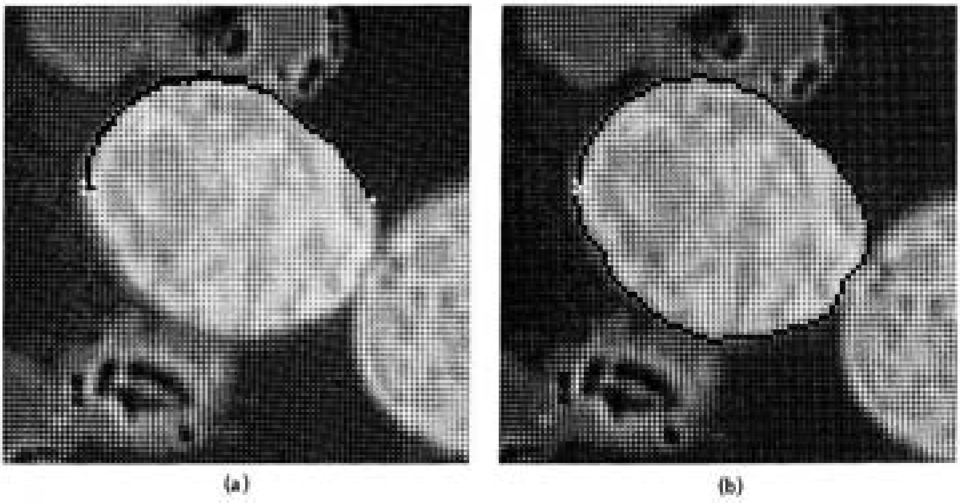


Fig. 4.14 Using least maximum cost in heuristic search to find cell boundaries in microscope images. (a) A stage in the search process. (b) The completed boundary.

However, the price paid is the sacrifice of the mathematical guarantee of finding the least-cost path. This could be reflected in unsatisfactory boundaries. This method has been used in cineangiograms with satisfactory results [Ashkar and Modestino 1978].

4.5 EDGE FOLLOWING AS DYNAMIC PROGRAMMING

4.5.1 Dynamic Programming

Dynamic programming [Bellman and Dreyfus 1962] is a technique for solving optimization problems when not all variables in the evaluation function are interrelated simultaneously. Consider the problem

$$\max_{x_i} h(x_1, x_2, x_3, x_4) \tag{4.8}$$

If nothing is known about h, the only technique that guarantees a global maximum is exhaustive enumeration of all combinations of discrete values of $x_1, \ldots, x_4$. Suppose that

$$h(\cdot) = h_1(x_1, x_2) + h_2(x_2, x_3) + h_3(x_3, x_4) \tag{4.9}$$

x_1 only depends on x_2 in h_1. Maximize over x_1 in h_1 and tabulate the best value of $h_1(x_{1,}\, x_{2)}$ for each x_2:

$$f_1(x_2) = \max_{x_1} h_1(x_1, x_2) \tag{4.10}$$

Since the values of h_2 and h_3 do not depend on x_1, they need not be considered at this point. Continue in this manner and eliminate x_2 by computing $f_2(x_3)$ as

$$f_2(x_3) = \max_{x_2}[f_1(x_2) + h_2(x_2, x_3)] \tag{4.11}$$

and

$$f_3(x_4) = \max_{x_3}[f_2(x_3) + h_3(x_3, x_4)] \tag{4.12}$$

so that finally

$$\max_{x_i} h = \max_{x_4} f_3(x_4) \tag{4.13}$$

Generalizing the example to N variables, where $f_0(x_1) = 0$,

$$f_{n-1}(x_n) = \max_{x_{n-1}}[f_{n-2}(x_{n-1}) + h_{n-1}(x_{n-1}, x_n)] \tag{4.14}$$

$$\max_{x_i} h(x_i, \ldots, x_N) = \max_{x_N} f_{N-1}(x_N)$$

If each x_i took on 20 discrete values, then to compute $f_N(x_{N+1})$ one must evaluate the maximand for 20 different combinations of x_N and x_{N+1}, so that the resultant computational effort involves $(N-1)20^2 + 20$ such evaluations. This is a striking improvement over exhaustive evaluation, which would involve 20^N evaluations of h!

Consider the artificial example summarized in Table 4.2. In this example, each **x** can take on one of three discrete values. The h_i are completely described by their respective tables. For example, the value of $h_i(0, 1) = 5$. The solution steps are summarized in Table 4.3. In step 1, for each x_2 the value of x_1 that maximizes $h_1(x_1, x_2)$ is computed. This is the largest entry in each of the columns of h. Store the function value as $f_1(x_2)$ and the optimizing value of x_1 also as a function of x_2. In step 2, add $f_1(x_2)$ to $h_2(x_2, x_3)$. This is done by adding f_1 to each row of h_2, thus computing the quantity inside the braces of (4.11). Now to complete step 2, for each x_3, compute the x_2 that maximizes $h_2 + f_1$ by selecting the largest entry in each row of the appropriate table. The rest of the steps are straightforward once these are understood. The solution is found by tracing back through the tables. For example, for $x_4 = 2$ we see that the best x_3 is -1, and therefore the best x_2 is 3 and x_1 is 1. This step is denoted by arrows.

Table 4.2

DEFINITION OF h

x_1 \ x_2	1	2	3
0	5	7	3
1	2	1	8
2	6	3	3

h_1

x_2 \ x_3	−1	0	1
1	1	7	1
2	1	1	3
3	5	6	2

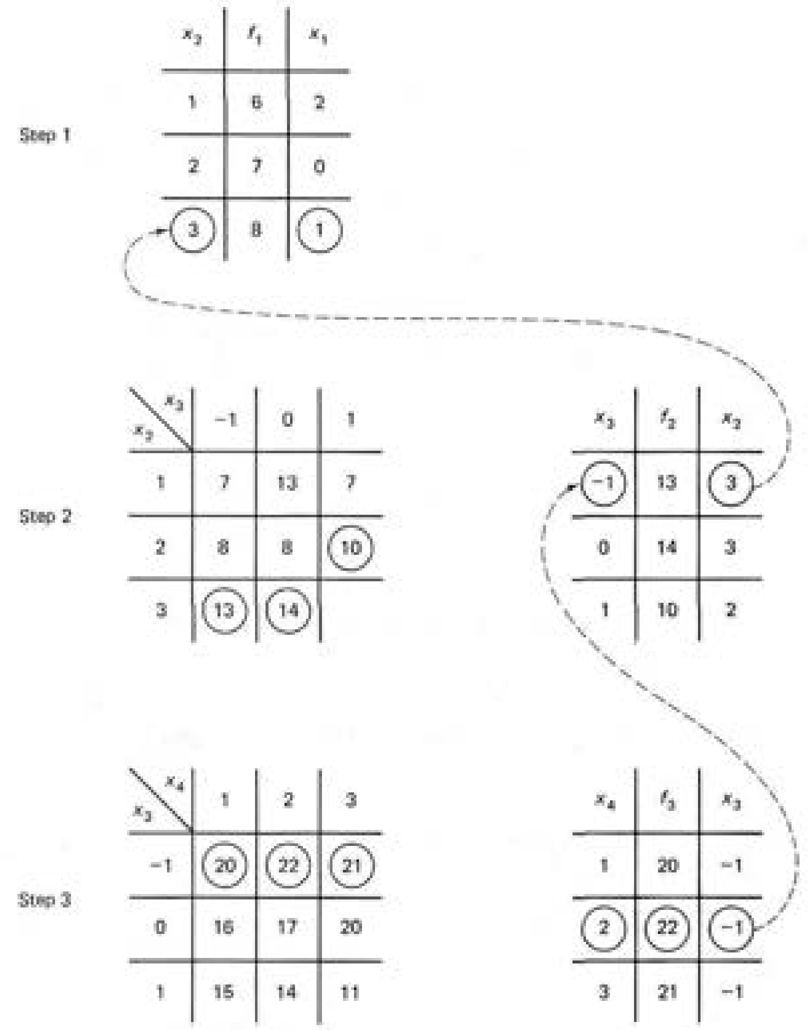
h_2

x_3 \ x_4	1	2	3
−1	7	9	8
0	2	3	6
1	5	4	1

h_3

Table 4.3

METHOD OF SOLUTION USING DYNAMIC PROGRAMMING



Step 1

x_2	f_1	x_1
1	6	2
2	7	0
3	8	1

Step 2

x_2 \ x_3	−1	0	1
1	7	13	7
2	8	8	10
3	13	14	

x_3	f_2	x_2
−1	13	3
0	14	3
1	10	2

Step 3

x_3 \ x_4	1	2	3
−1	20	22	21
0	16	17	20
1	15	14	11

x_4	f_3	x_3
1	20	−1
2	22	−1
3	21	−1

Step 4: Optimal x_i's are found by examing tables (dashed line shows the order in which they are recovered).

Solution: $h^* = 22$
$x_1^* = 1, x_2^* = 3, x_3^* = -1, x_4^* = 2$

4.5.2 Dynamic Programming for Images

To formulate the boundary-following procedure as dynamic programming, one must define an evaluation function that embodies a notion of the "best boundary" [Montanari 1971; Ballard 1976]. Suppose that a local edge detection operator is applied to a gray-level picture to produce edge magnitude and direction information. Then one possible criterion for a "good boundary" is a weighted sum of high cumulative edge strength and low cumulative curvature; that is, for an n-segment curve,

$$h(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{k=1}^{n} s(\mathbf{x}_k) + \alpha \sum_{k=1}^{n-1} q(\mathbf{x}_k, \mathbf{x}_{k+1}) \tag{4.16}$$

where the implicit constraint is that consecutive $\mathbf{x}_k$'s must be grid neighbors:

$$\|\mathbf{x}_k - \mathbf{x}_{k+1}\| \leqslant \sqrt{2} \tag{4.17}$$

$$q(\mathbf{x}_k, \mathbf{x}_{k+1}) = diff\,[\phi(\mathbf{x}_k), \phi(\mathbf{x}_{k+1})] \tag{4.18}$$

where α is negative. The function g we take to be edge strength, i.e., $g(x) = s(x)$. Notice that this evaluation function is in the form of (4.9) and can be optimized in stages:

$$f_0(\mathbf{x}_1) \equiv 0 \tag{4.19}$$

$$f_1(\mathbf{x}_2) = \max_{x_1} [s(\mathbf{x}_1) + \alpha q(\mathbf{x}_1, \mathbf{x}_2) + f_0(\mathbf{x}_1)] \tag{4.20}$$

$$f_k(\mathbf{x}_{k+1}) = \max_{x_k} [s(\mathbf{x}_k) + \alpha q(\mathbf{x}_k, \mathbf{x}_{k+1}) + f_{k-1}(\mathbf{x}_k)] \tag{4.21}$$

These equations can be put into the following steps:

Algorithm 4.5: Dynamic Programming for Edge Finding

1. Set $k = 1$.
2. Consider only $\mathbf{x}$ such that $s(\mathbf{x}) \geqslant T$. For each of these $\mathbf{x}$, define low-curvature pixels "in front of" the contour direction.
3. Each of these pixels may have a curve emanating from it. For $k = 1$, the curve is one pixel in length. Join the curve to $\mathbf{x}$ that optimizes the left-hand side of the recursion equation.
4. If $k = N$, pick the best f_{N-1} and stop. Otherwise, set $k = k + 1$ and go to step 2.

This algorithm can be generalized to the case of picking a *curve* emanating from $\mathbf{x}$ (that we have already generated): Find the end of that curve, and join the best of three curves emanating from the end of that curve. Figure 4.15 shows this process. The equations for the general case are

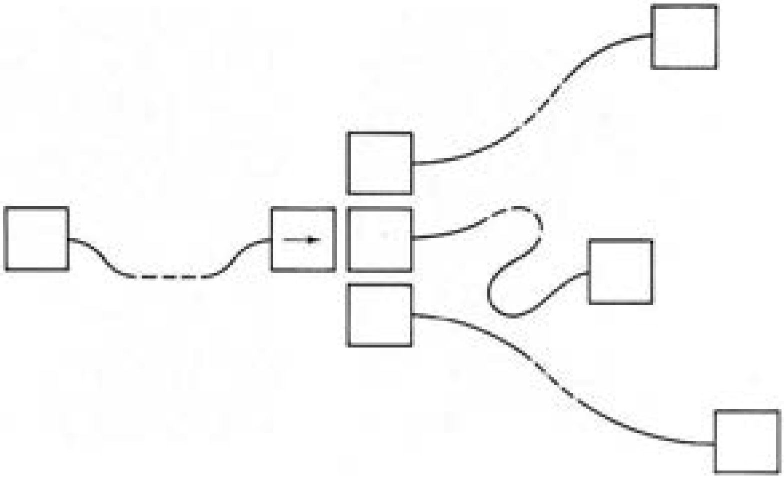


Fig. 4.15 DP optimization for boundary tracing.

$$
\begin{aligned}
f_0(\mathbf{x}_1) &\equiv 0 \\
f_l(\mathbf{x}_{k+1}) &= \max_{x_k}[s(\mathbf{x}_k) + \alpha q(\mathbf{x}_k, t(\mathbf{x}_{k+1})) \\
&\qquad + f_{l-1}(\mathbf{x}_k)]
\end{aligned}
\tag{4.22}
$$

where the curve length n is related to α by a building sequence $n(l)$ such that $n(1) = 1$, $n(L) = N$, and $n(l) - n(l-1)$ is a member of $\{n(k)|k = 1, ..., l-1\}$. Also, $t(\mathbf{x}_k)$ is a function that extracts the tail pixel of the curve headed by $\mathbf{x}_k$. Further details may be found in [Ballard 1976].

Results from the area of tumor detection in radiographs give a sense of this method's performance. Here it is known that the boundary inscribes an approximately circular tumor, so that circular cues can be used to assist the search. In Fig. 4.16, (a) shows the image containing the tumor, (b) shows the cues, and (c) shows the boundary found by dynamic programming overlaid on the image.

Another application of dynamic programming may be found in the pseudoparallel road finder of Barrow [Barrow 1976].

4.5.3 Lower Resolution Evaluation Functions

In the dynamic programming formulation just developed, the components $g(\mathbf{x}_k)$ and $q(\mathbf{x}_k, \mathbf{x}_{k+1})$ in the evaluation function are very localized; the variables $\mathbf{x}$ for successive s and q are in fact constrained to be grid neighbors. This need not be the case: The $\mathbf{x}$ can be very distant from each other without altering the basic technique. Furthermore, the functions g and q need not be local gradient and absolute curvature, respectively, but can be any functions defined on permissible $\mathbf{x}$. This general formulation of the problem for images was first described by [Fischler and

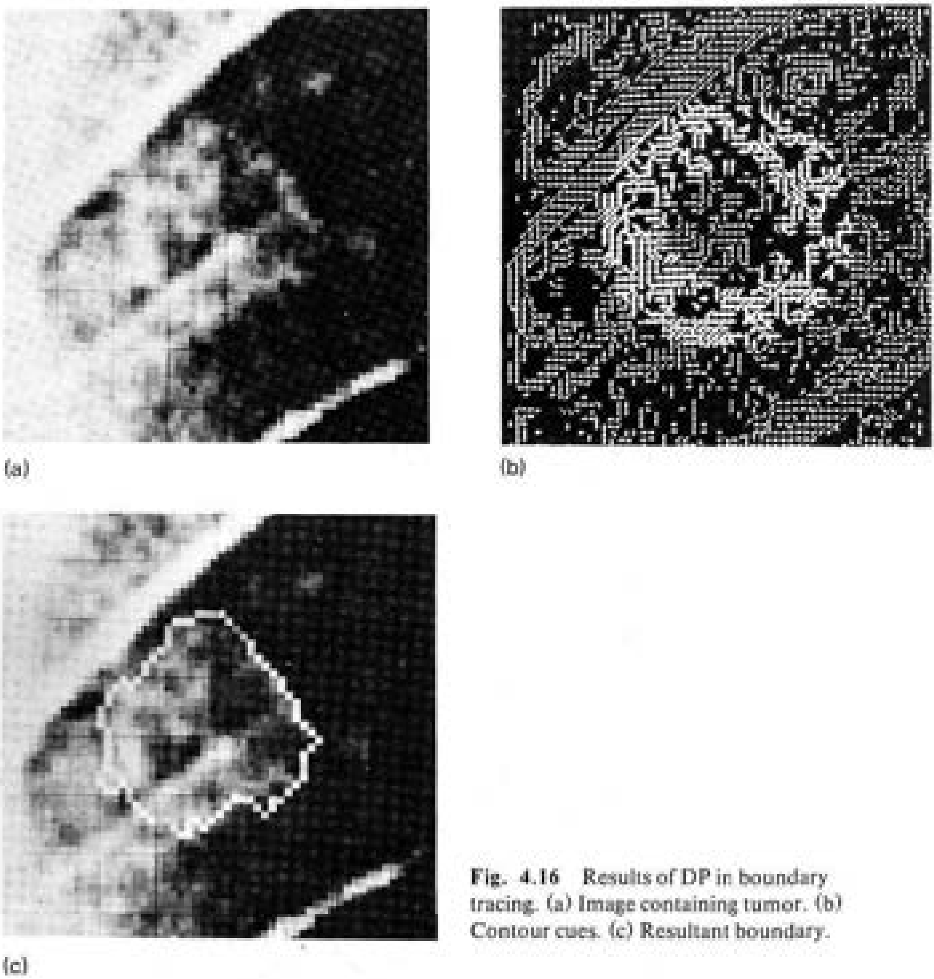


Fig. 4.16 Results of DP in boundary tracing. (a) Image containing tumor. (b) Contour cues. (c) Resultant boundary.

Elschlager 1973]. The Fischler and Elschlager formulation models an object as a set of parts and relations between parts, represented as a graph. Template functions, denoted by $g(\mathbf{x})$, measure how well a part of the model matches a part of the image at the point $\mathbf{x}$. (These local functions may be defined in any manner whatsoever.) "Relational functions," denoted by $q_{kj}(\mathbf{x}, \mathbf{y})$, measure how well the position of the match of the kth part at $(\mathbf{x})$ agrees with the position of the match of the jth part at $(\mathbf{y})$.

The basic notions are shown by a technique simplified from [Chien and Fu 1974] to find the boundaries of lungs in chest films. The lung boundaries are modeled with a polygonal approximation defined by the five key points. These points are the top of the lung, the two clavicle-lung junctions, and the two lower corners. To locate these points, local functions $g(\mathbf{x}_k)$ are defined which should be maximized when the corresponding point $\mathbf{x}_k$ is correctly determined. Similarly, $q(\mathbf{x}_k, \mathbf{x}_j)$ is a function relating points $\mathbf{x}_k$ and $\mathbf{x}_j$. In their case, Chien and Fu used the following functions:

$$T(\mathbf{x}) \equiv \text{template centered at } \mathbf{x} \text{ computed as an aggregate of a set of chest radiographs}$$

$$g(\mathbf{x}_k) = \sum_{\mathbf{x}} \frac{T(\mathbf{x} - \mathbf{x}_k) f(x)}{|T||f|}$$

and

$$\theta(\mathbf{x}_k, \mathbf{x}_j) = \text{expected angular orientation of } \mathbf{x}_k \text{ from } \mathbf{x}_j$$

$$q(\mathbf{x}_k \, \mathbf{x}_j) = \left[\theta(\mathbf{x}_k, \mathbf{x}_j) - \arctan \frac{y_k - y_j}{x_k - x_j} \right]$$

With this formulation no further modifications are necessary and the solution may be obtained by solving Eqs. (4.19) through (4.21), as before. For purposes of comparison, this method was formalized using a lower-resolution objective function. Figure 4.17 shows Chien and Fu's results using this method with five template functions.

4.5.4 Theoretical Questions about Dynamic Programming

The Interaction Graph

This graph describes the interdependence of variables in the objective function. In the examples the interaction graph was simple: Each variable depended on only two others, resulting in the graph of Fig. 4.18a. A more complicated case is the one in 4.18b, which describes an objective function of the following form:

$$h(\,) = h_1(x_1, x_2) + h_2(x_2, x_3, x_4) + h_3(x_3, x_4, x_5, x_6)$$

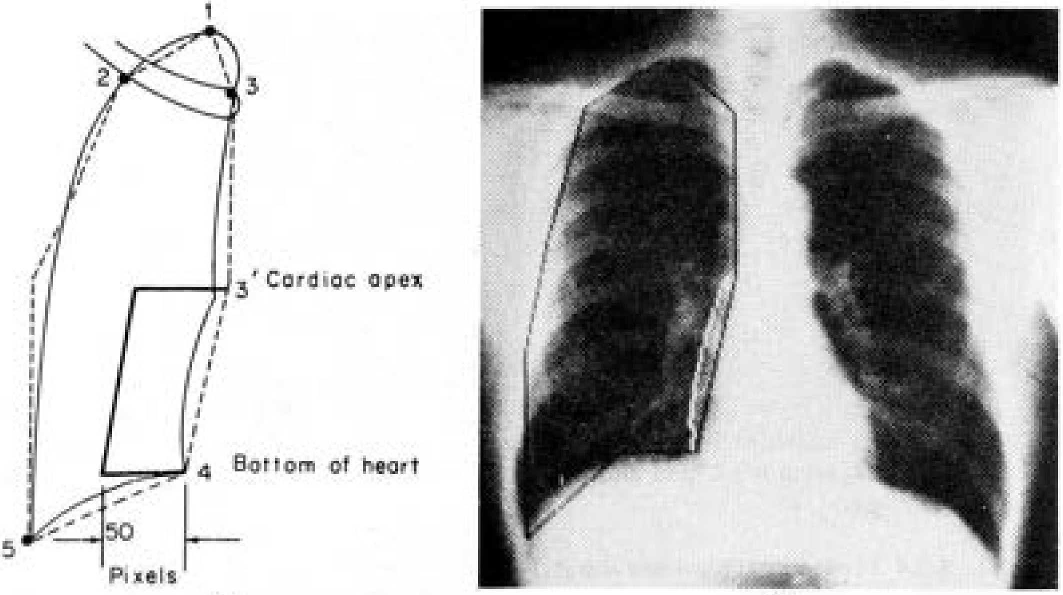
For these cases the dynamic programming technique still applies, but the computational effort increases exponentially with the number of interdependencies. For example, to eliminate x_2 in h_2, all possible combinations of x_3 and x_4 must be considered. To eliminate x_3 in h_3, all possible combinations of x_4, x_5, and x_6, and so forth.

Dynamic Programming versus Heuristic Search

It has been shown [Martelli 1976] that for finding a path in a graph between two points, which is an abstraction of the work we are doing here, heuristic search methods can be more efficient than dynamic programming methods. However, the point to remember about dynamic programming is that it efficiently builds paths from multiple starting points. If this is required by a particular task, then dynamic programming would be the method of choice, unless a very powerful heuristic were available.

4.6 CONTOUR FOLLOWING

If nothing is known about the boundary shape, but regions have been found in the image, the boundary is recovered by one of the simplest edge-following operations: "blob finding" in images. The ideas are easiest to present for binary images:



(a) (b)

Fig. 4.17 Results of using local templates and global relations. (a) Model. (b) Results.

Given a binary image, the goal is find the boundaries of all distinct regions in the image.

This can be done simply by a procedure that functions like Papert's turtle [Papert 1973; Duda and Hart 1973]:

1. Scan the image until a region pixel is encountered.
2. If it is a region pixel, turn left and step; else, turn right and step.
3. Terminate upon return to the starting pixel.

Figure 4.19 shows the path traced out by the procedure. This procedure requires the region to be four-connected for a consistent boundary. Parts of an eight-connected region can be missed. Also, some bookkeeping is necessary to generate an exact sequence of boundary pixels without duplications.

A slightly more elaborate algorithm due to [Rosenfeld 1968] generates the boundary pixels exactly. It works by first finding a four-connected background pixel from a known boundary pixel. The next boundary pixel is the first pixel encountered when the eight neighbors are examined in a counter clockwise order from the background pixel. Many details have to be introduced into algorithms that follow contours of irregular eight-connected figures. A good exposition of these is given in [Rosenfeld and Kak 1976].

4.6.1 Extension to Gray-Level Images

The main idea behind contour following is to start with a point that is believed to be on the boundary and to keep extending the boundary by adding points in the contour directions. The details of these operations vary from task to task. The gen-

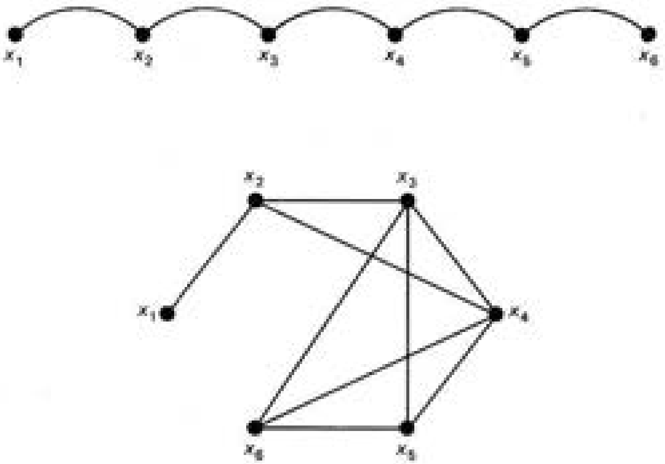


Fig. 4.18 Interaction graphs for DP (see text).

eralization of the contour follower to gray-level images uses local gradients with a magnitude $s(\mathbf{x})$ and direction $\phi(\mathbf{x})$ associated with each point $\mathbf{x}$. ϕ points in the direction of maximum change. If $\mathbf{x}$ is on the boundary of an image object, neighboring points on the boundary should be in the general direction of the contour directions, $\phi(\mathbf{x}) \pm \pi/2$, as shown by Fig. 4.20. A representative procedure is adapted from [Martelli 1976]:

1. Assume that an edge has been detected up to a point $\mathbf{x}_i$. Move to the point $\mathbf{x}_j$ adjacent to $\mathbf{x}_i$ in the direction perpendicular to the gradient of $\mathbf{x}_i$. Apply the gradient operator to $\mathbf{x}_j$; if its magnitude is greater than (some) threshold, this point is added to the edge.
2. Otherwise, compute the average gray level of the 3 × 3 array centered on $\mathbf{x}_j$, compare it with a suitably chosen threshold, and determine whether $\mathbf{x}_j$ is inside or outside the object.
3. Make another attempt with a point $\mathbf{x}_k$ adjacent to $\mathbf{x}_i$ in the direction perpendicular to the gradient at $\mathbf{x}_i$ plus or minus $(\pi/4)$, according to the outcome of the previous test.

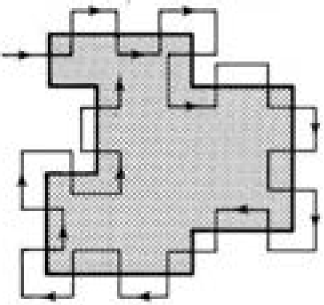


Fig. 4.19 Finding the boundary in a binary image.

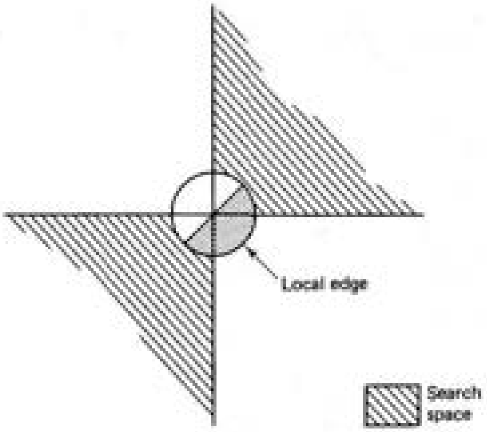


Fig. 4.20 Angular orientations for contour following.

4.6.2 Generalization to Higher-Dimensional Image Data

The generalization of contour following to higher-dimensional spaces is straightforward [Liu 1977; Herman and Liu 1978]. The search involved is, in fact, slightly more complex than contour following and is more like the graph searching methods described in Section 4.4. Higher-dimensional image spaces arise when the image has more than two spatial dimensions, is time-varying, or both. In these images the notion of a gradient is the same (a vector describing the maximum gray-level change and its corresponding direction), but the intuitive interpretation of the corresponding edge element may be difficult. In three dimensions, edge elements are primitive surface elements, separating volumes of differing gray level. The objective of contour following is to link together neighboring surface elements with high gradient modulus values and similar orientations into larger boundaries. In four dimensions, "edge elements" are primitive volumes; contour following links neighboring volumes with similar gradients.

The contour following approach works well when there is little noise present and no "spurious" boundaries. Unfortunately, if either of these conditions is present, the contour-following algorithms are generally unsatisfactory; they are easily thwarted by gaps in the data produced by noise, and readily follow spurious boundaries. The methods described earlier in this chapter attempt to overcome these difficulties through more elaborate models of the boundary structure.

EXERCISES

4.1 Specify a heuristic search algorithm that will work with "crack" edges such as those in Fig. 3.12.

4.2 Describe a modification of Algorithm 4.2 to detect parabolae in gray-level images.

4.3 Suppose that a relation $h(x_1, x_6)$ is added to the model described by Fig. 4.18a so that now the interaction graph is cyclical. Show formally how this changes the optimization steps described by Eqs. (4.11) through (4.13).

4.4 Show formally that the Hough technique without gradient direction information is equivalent to template matching (Chapter 3).

4.5 Extend the Hough technique for ellipses described by Eqs. (4.7a) through (4.7d) to ellipses oriented at an arbitrary angle θ to the x axis.

4.6 Show how to use the generalized Hough technique to detect hexagons.

REFERENCES

Ashkar, G. P. and J. W. Modestino. "The contour extraction problem with biomedical applications." *CGIP 7*, 1978, 331–355.

Ballard, D. H. *Hierarchic detection of tumors in chest radiographs.* Basel: Birkhäuser-Verlag (ISR-16), January 1976.

Ballard, D. H. "Generalizing the Hough transform to detect arbitrary shapes." *Pattern Recognition 13*, 2, 1981, 111–122.

Ballard, D. H. and J. Sklansky. "A ladder-structured decision tree for recognizing tumors in chest radiographs." *IEEE Trans. Computers 25*, 1976, 503-513.

Ballard, D. H., M. Marinucci, F. Proietti-Orlandi, A. Rossi-Mari, and L. Tentari. "Automatic analysis of human haemoglobin fingerprints." *Proc.*, 3rd Meeting, International Society of Haemotology, London, August 1975.

Barrow, H. G. "Interactive aids for cartography and photo interpretation." Semi-Annual Technical Report, AI Center, SRI International, December 1976.

Bellman, R. and S. Dreyfus. *Applied Dynamic Programming.* Princeton, NJ: Princeton University Press, 1962.

Bolles, R. "Verification vision for programmable assembly." *Proc.*, 5th IJCAI, August 1977, 569-575.

Chien, Y. P. and K. S. Fu. "A decision function method for boundary detection." *CGIP 3*, 2, June 1974, 125-140.

Duda, R. O. and P. E. Hart. "Use of the Hough transformation to detect lines and curves in pictures." *Commun. ACM 15*, 1, January 1972, 11–15.

Duda, R. O. and P. E. Hart. *Pattern Recognition and Scene Analysis.* New York: Wiley, 1973.

Fischler, M. A. and R. A. Elschlager. "The representation and matching of pictoral patterns." *IEEE Trans. Computers 22*, January 1973.

Herman, G. T. and H. K. Liu. "Dynamic boundary surface detection." *CGIP 7*, 1978, 130-138.

Hough, P. V. C. "Method and means for recognizing complex patterns." U.S. Patent 3,069,654; 1962.

Kelly, M.D. "Edge detection by computer using planning." In *MI6*, 1971.

Kimme, C., D. Ballard, and J. Sklansky. "Finding circles by an array of accumulators." *Commun. ACM 18*, 2, 1975, 120-122.

Lantz, K. A., C. M. Brown and D. H. Ballard. "Model-driven vision using procedure decription: motivation and application to photointerpretation and medical diagnosis." *Proc.*, 22nd International Symp., Society of Photo-optical Instrumentation Engineers, San Diego, CA, August 1978.

Lester, J. M., H. A. Williams, B. A. Weintraub, and J. F. Brenner, "Two graph searching techniques for boundary finding in white blood cell images." *Computers in Biology and Medicine 8*, 1978, 293-308.

Liu, H. K. "Two- and three-dimensional boundary detection." *CGIP 6*, 2, April 1977, 123-134.

Marr, D. "Analyzing natural images; a computational theory of texture vision." Technical Report 334, AI Lab, MIT, June 1975.

Martelli, A. "Edge detection using heuristic search methods." *CGIP 1*, 2, August 1972, 169-182.

Martelli, A. "An application of heuristic search methods to edge and contour detection." *Commun. ACM 19*, 2, February 1976, 73–83.

MONTANARI, U. "On the optimal detection of curves in noisy pictures." *Commun. ACM 14*, 5, May 1971, 335-345.

NILSSON, N. J. *Problem-Solving Methods in Artificial Intelligence.* New York: McGraw-Hill, 1971.

NILSSON, N. J. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980.

PAPERT, S. "Uses of technology to enhance education." Technical Report 298, AI Lab, MIT, 1973.

PERSOON, E. "A new edge detection algorithm and its applications in picture processing." *CGIP 5*, 4, December 1976, 425-446.

RAMER, U. "Extraction of line structures from photographs of curved objects." *CGIP 4*, 2, June 1975, 81-103.

ROSENFELD, A. *Picture Processing by Computer.* New York: Academic Press, 1968.

ROSENFELD, A. and A. C. KAK. *Digital Picture Processing.* New York: Academic Press, 1976.

SELFRIDGE, P. G., J. M. S. PREWITT, C. R. DYER, and S. RANADE. "Segmentation algorithms for abdominal computerized tomography scans." *Proc.*, 3rd COMPSAC, November 1979, 571-577.

WECHSLER, H. and J. SKLANSKY. "Finding the rib cage in chest radiographs." *Pattern Recognition* 9, 1977, 21-30.

Region Growing 5

5.1 REGIONS

Chapter 4 concentrated on the linear features (discontinuities of image gray level) that often correspond to object boundaries, interesting surface detail, and so on. The "dual" problem to finding edges around regions of differing gray level is to find the regions themselves. The goal of region growing is to use image characteristics to map individual pixels in an input image to sets of pixels called *regions*. An image region might correspond to a world object or a meaningful part of one.

Of course, very simple procedures will derive a boundary from a connected region of pixels, and conversely can fill a boundary to obtain a region. There are several reasons why both region growing and line finding survive as basic segmentation techniques despite their redundant-seeming nature. Although perfect regions and boundaries are interconvertible, the processing to find them initially differs in character and applicability; besides, perfect edges or regions are not always required for an application. Region-finding and line-finding techniques can cooperate to produce a more reliable segmentation.

The geometric characteristics of regions depend on the domain. Usually, they are considered to be connected two-dimensional areas. Whether regions can be disconnected, non-simply connected (have holes), should have smooth boundaries, and so forth depends on the region-growing technique and the goals of the work. Ultimately, it is often the segmentation goal to partition the entire image into quasi-disjoint regions. That is, regions have no two-dimensional overlaps, and no pixel belongs to the interior of more than one region. However, there is no single definition of region—they may be allowed to overlap, the whole image may not be partitioned, and so forth.

Our discussion of region growers will begin with the most simple kinds and progress to the more complex. The most primitive region growers use only aggregates of properties of local groups of pixels to determine regions. More sophisticated techniques "grow" regions by *merging* more primitive regions. To do this in a structured way requires sophisticated representations of the regions and boundaries. Also, the merging *decisions* can be complex, and can depend on descriptions of the boundary structure separating regions in addition to the region semantics. A good survey of early techniques is [Zucker 1976].

The techniques we consider are:

1. *Local techniques.* Pixels are placed in a region on the basis of their properties or the properties of their close neighbors.
2. *Global techniques.* Pixels are grouped into regions on the basis of the properties of large numbers of pixels distributed throughout the image.
3. *Splitting and merging techniques.* The foregoing techniques are related to individual pixels or sets of pixels. State space techniques merge or split regions using graph structures to represent the regions and boundaries. Both local and global merging and splitting criteria can be used.

The effectiveness of region growing algorithms depends heavily on the application area and input image. If the image is sufficiently simple, say a dark blob on a light background, simple local techniques can be surprisingly effective. However, on very difficult scenes, such as outdoor scenes, even the most sophisticated techniques still may not produce a satisfactory segmentation. In this event, region growing is sometimes used conservatively to preprocess the image for more knowledgeable processes [Hanson and Riseman 1978].

In discussing the specific algorithms, the following definitions will be helpful. Regions R_k are considered to be sets of points with the following properties:

$\mathbf{x}_i$ in a region R is *connected* to $\mathbf{x}_j$ iff there
is a sequence $\{\mathbf{x}_i, \ldots, \mathbf{x}_j\}$ such that $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$
are connected and all the points are in R. (5.1)

R is a *connected region* if the set of points $\mathbf{x}$ in R has the
property that every pair of points is connected. (5.2)

$$I, \text{ the entire image} = \bigcup_{k=1}^{m} R_k \tag{5.3}$$

$$R_i \bigcap R_j = \phi, \qquad i \neq j \tag{5.4}$$

A set of regions satisfying (5.2) through (5.4) is known as a *partition*. In segmentation algorithms, each region often is a unique, homogeneous area. That is, for some Boolean function $H(R)$ that measures region homogeneity,

$$H(R_k) = \text{true for all } k \tag{5.5}$$

$$H(R_i \bigcup R_j) = \text{false for } i \neq j \tag{5.6}$$

Note that R_i does not have to be connected. A weaker but still useful criterion is that neighboring regions not be homogeneous.

5.2 A LOCAL TECHNIQUE: BLOB COLORING

The counterpart to the edge tracker for binary images is the blob-coloring algorithm. Given a binary image containing four-connected blobs of 1's on a background of 0's, the objective is to "color each blob"; that is, assign each blob a different label. To do this, scan the image from left to right and top to bottom with a special L-shaped template shown in Fig. 5.1. The coloring algorithm is as follows.

Algorithm 5.1: Blob Coloring

Let the initial color, $k = 1$. Scan the image from left to right and top to bottom.

```
If $f(\mathbf{x}_C) = 0$ then continue
else
        begin

        if ($f(\mathbf{x}_U) = 1$ and $f(\mathbf{x}_L) = 0$)
        then color ($\mathbf{x}_C$) := color ($\mathbf{x}_U$)

        if ($f(\mathbf{x}_L) = 1$ and $f(\mathbf{x}_U) = 0$)
        then color ($\mathbf{x}_C$) := color ($\mathbf{x}_L$)

        if ($f(\mathbf{x}_L) = 1$ and $f(\mathbf{x}_U) = 1$)
        then begin
                color ($\mathbf{x}_C$) := color ($\mathbf{x}_L$)
                color ($\mathbf{x}_L$) is equivalent to color ($\mathbf{x}_U$)
                end

    comment: two colors are equivalent.

        if ($f(\mathbf{x}_L) = 0$ and $f(\mathbf{x}_U) = 0$)
        then color ($\mathbf{x}_L$) := $k$; $k := k + 1$

        comment: new color

        end
```

After one complete scan of the image the color equivalences can be used to assure that each object has only one color. This binary image algorithm can be used as a simple region-grower for gray-level images with the following modifications. If in a

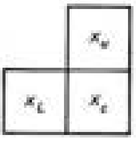


Fig. 5.1 L-shaped template for blob coloring.

gray-level image $f(\mathbf{x}_C)$ is approximately equal to $f(\mathbf{x}_U)$, assign $\mathbf{x}_C$ to the same region (blob) as $\mathbf{x}_U$. This is equivalent to the condition $f(\mathbf{x}_C) = f(\mathbf{x}_U) = 1$ in Algorithm 5.1. The modifications to the steps in the algorithm are straightforward.

5.3 GLOBAL TECHNIQUES: REGION GROWING VIA THRESHOLDING

This approach assumes an object-background image and picks a threshold that divides the image pixels into either object or background:

$\mathbf{x}$ is part of the Object iff $f(\mathbf{x}) > T$
Otherwise it is part of the Background

The best way to pick the threshold T is to search the histogram of gray levels, assuming it is bimodal, and find the minimum separating the two peaks, as in Fig. 5.2. Finding the right valley between the peaks of a histogram can be difficult when the histogram is not a smooth function. Smoothing the histogram can help but does not guarantee that the correct minimum can be found. An elegant method for treating bimodal images assumes that the histogram is the sum of two composite normal functions and determines the valley location from the normal parameters [Chow and Kaneko 1972].

The single-threshold method is useful in simple situations, but primitive. For example, the region pixels may not be connected, and further processing such as that described in Chapter 2 may be necessary to smooth region boundaries and remove noise. A common problem with this technique occurs when the image has a background of varying gray level, or when collections we would like to call regions vary smoothly in gray level by more than the threshold. Two modifications of the threshold approach to ameliorate the difficulty are: (1) high-pass filter the image to deemphasize the low-frequency background variation and then try the original technique; and (2) use a spatially varying threshold method such as that of [Chow and Kaneko 1972].

The Chow-Kaneko technique divides the image up into rectangular subimages and computes a threshold for each subimage. A subimage can fail to have a threshold if its gray-level histogram is not bimodal. Such subimages receive inter-

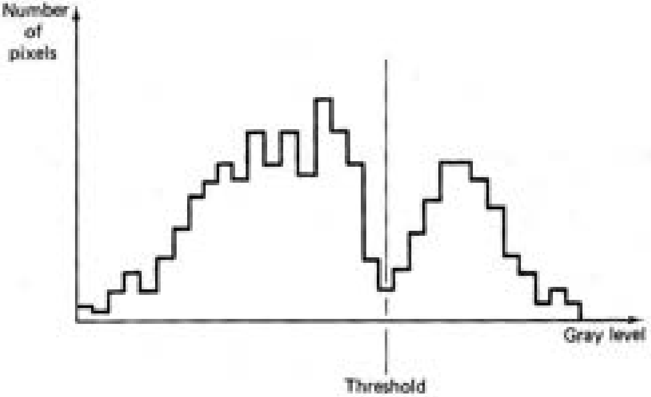


Fig. 5.2 Threshold determination from gray-level histogram.

polated thresholds from neighboring subimages that are bimodal, and finally the entire picture is thresholded by using the separate thresholds for each subimage.

5.3.1 Thresholding in Multidimensional Space

An interesting variation to the basic thresholding paradigm uses color images; the basic digital picture function is vector-valued with red, blue, and green components. This vector is augmented with possibly nonlinear combinations of these values so that the augmented picture vector has a number of components. The idea is to re-represent the color solid redundantly and hope to find color parameters for which thresholding does the desired segmentation. One implementation of this idea used the red, green, and blue color components; the intensity, saturation, and hue components; and the N.T.S.C. *Y*, *I*, *Q* components (Chapter 2) [Ohlander et al. 1979].

The idea of thresholding the components of a picture vector is used in a primitive form for multispectral LANDSAT imagery [Robertson et al. 1973]. The novel extension in this algorithm is the recursive application of this technique to nonrectangular subregions.

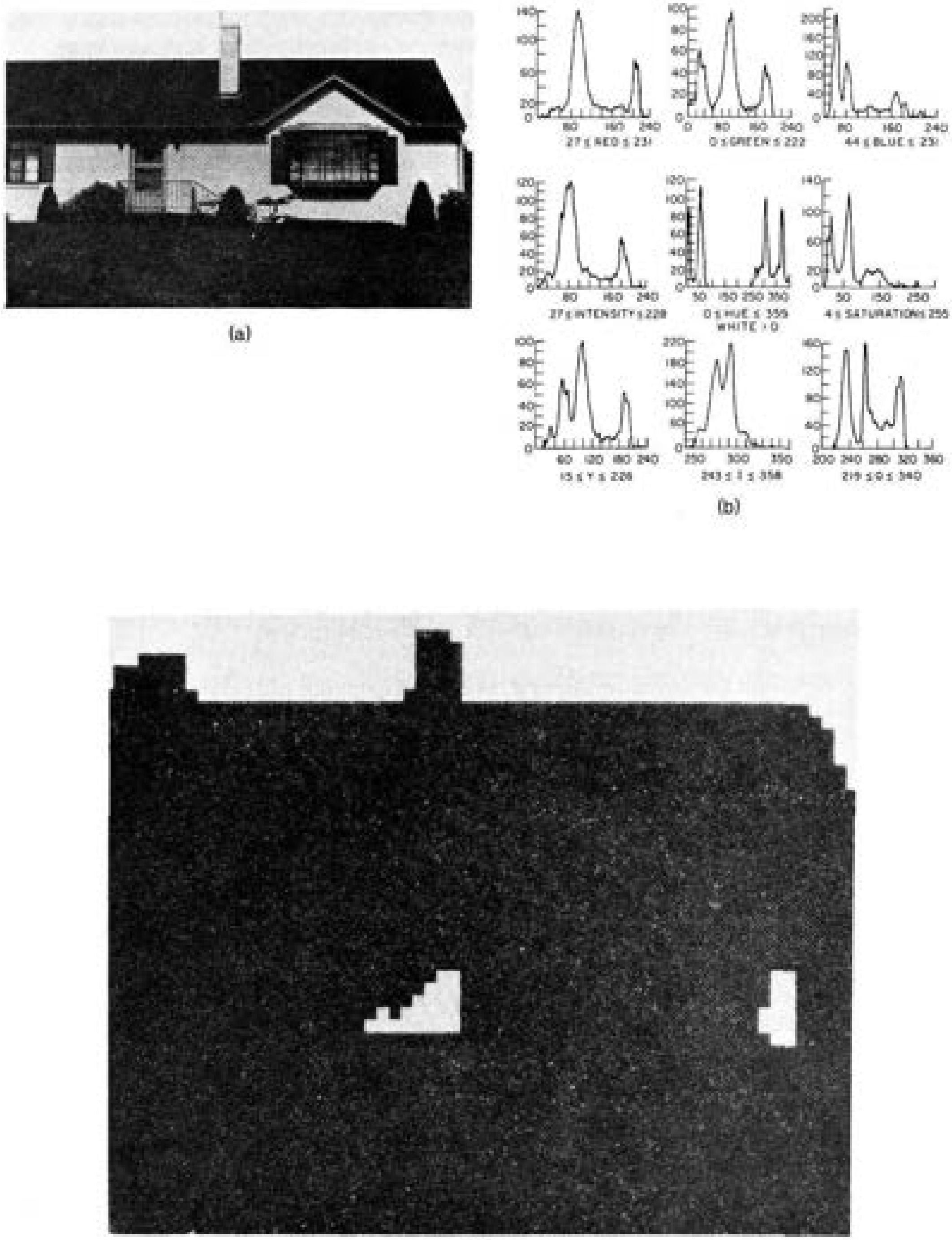
The region partitioning is then as follows:

Algorithm 5.2: Region Growing via Recursive Splitting

1. Consider the entire image as a region and compute histograms for each of the picture vector components.
2. Apply a peak-finding test to each histogram. If at least one component passes the test, pick the component with the most significant peak and determine two thresholds, one either side of the peak (Fig. 5.3). Use these thresholds to divide the region into subregions.
3. Each subregion may have a "noisy" boundary, so the binary representation of the image achieved by thresholding is smoothed so that only a single connected subregion remains. For binary smoothing see ch. 8 and [Rosenfeld and Kak 1976].
4. Repeat steps 1 through 3 for each subregion until no new subregions are created (no histograms have significant peaks).

A refinement of step 2 of this scheme is to create histograms in higher-dimensional space [Hanson and Riseman 1978]. Multiple regions are often in the same histogram peak when a single measurement is used. The advantage of the multimeasurement histograms is that these different regions are often separated into individual peaks, and hence the segmentation is improved. Figure 5.4 shows some results using a three-dimensional RGB color space.

The figure shows the clear separation of peaks in the three-dimensional histogram that is not evident in either of the one-dimensional histograms. How many

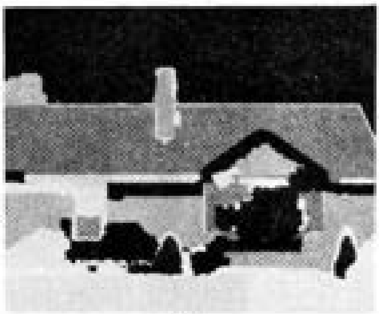


(a)

(b)

(c)

Fig. 5.3 Peak detection and threshold determination. (a) Original image. (b) Histograms. (c) Image segments resulting from first histogram peak.



(d)

Fig. 5.3 (d) Final segments.

dimensions should be used? Obviously, there is a trade-off here: As the dimensionality becomes larger, the discrimination improves, but the histograms are more expensive to compute and noise effects may be more pronounced.

5.3.2 Hierarchical Refinement

This technique uses a pyramidal image representation (Section 3.7) [Harlow and Eisenbeis 1973]. Region growing is applied to a coarse resolution image. When the algorithm has terminated at one resolution level, the pixels near the boundaries of regions are disassociated with their regions. The region-growing process is then repeated for just these pixels at a higher-resolution level. Figure 5.5 shows this structure.

5.4 SPLITTING AND MERGING

Given a set of regions R_k, $k = 1, \ldots, m$, a low-level segmentation might require the basic properties described in Section 5.1 to hold. The important properties from the standpoint of segmentation are Eqs. (5.5) and (5.6).

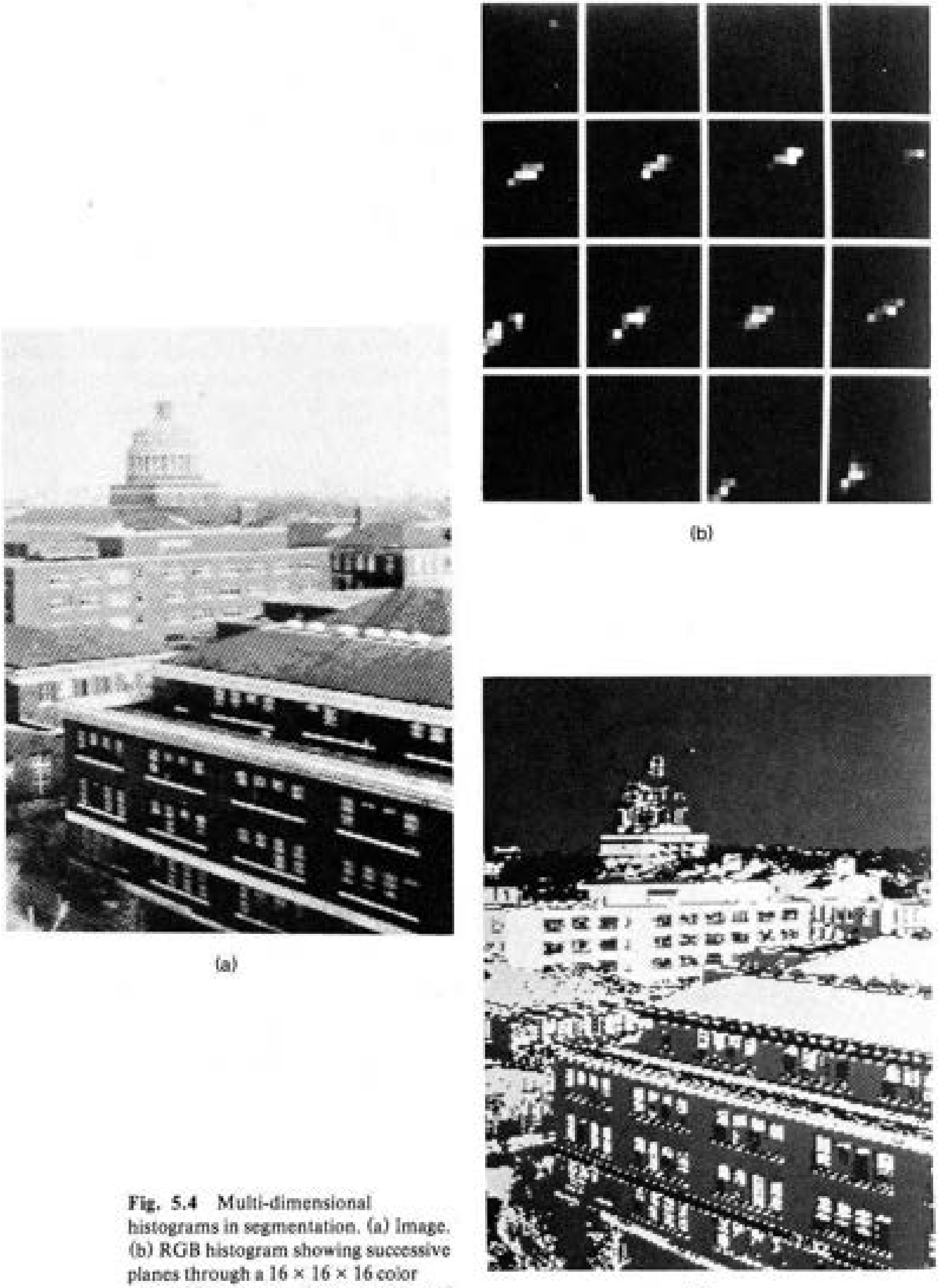
If Eq. (5.5) is not satisfied for some k, it means that that region is inhomogeneous and should be split into subregions. If Eq. (5.6) is not satisfied for some i and j, then regions i and j are collectively homogeneous and should be merged into a single region.

In our previous discussions we used

$$H(R) = \begin{cases} \text{true} & \text{if all neighboring pairs of points} \\ & \text{in } R \text{ are such that } f(\mathbf{x}) - f(\mathbf{y}) < T \\ \text{false} & \text{otherwise} \end{cases} \tag{5.7}$$

and

$$H(R) = \begin{cases} \text{true} & \text{if the points in } R \text{ pass a} \\ & \text{bimodality or peak test} \\ \text{false} & \text{otherwise} \end{cases} \tag{5.8}$$



(a)

(b)

(c)

Fig. 5.4 Multi-dimensional histograms in segmentation. (a) Image. (b) RGB histogram showing successive planes through a 16 × 16 × 16 color space. (c) Segments. (See color inserts.)

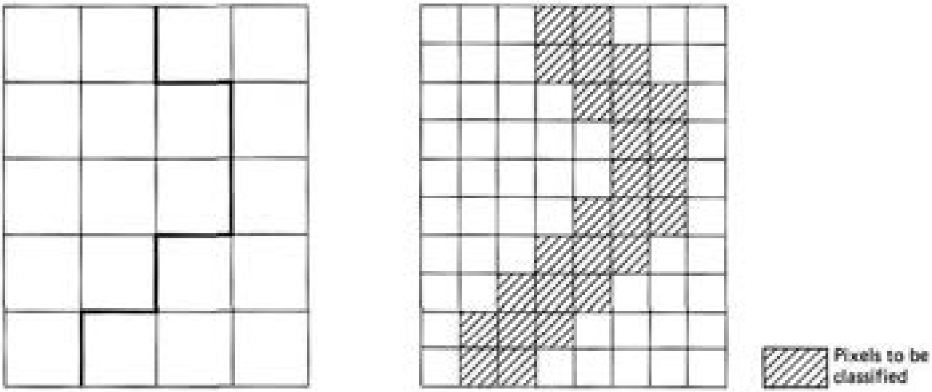


Fig. 5.5 Hierarchical region refinement.

A way of working toward the satisfaction of these homogeneity criteria is the split-and-merge algorithm [Horowitz and Pavlidis 1974]. To use the algorithm it is necessary to organize the image pixels into a pyramidal grid structure of regions. In this grid structure, regions are organized into groups of four. Any region can be split into four subregions (except a region consisting of only one pixel), and the appropriate groups of four can be merged into a single larger region. This structure is incorporated into the following region-growing algorithm.

Algorithm 5.3: Region Growing via Split and Merge [Horowitz and Pavlidis 1974]

1. Pick any grid structure, and homogeneity property H. If for any region R in that structure, $H(R)$ = false, split that region into four subregions. If for any four appropriate regions R_{k1} ...., R_{k4}, $H(R_{k1} \bigcup R_{k2} \bigcup R_{k3} \bigcup R_{k4})$ = true, merge them into a single region. When no regions can be further split or merged, stop.
2. If there are any neighboring regions R_i and R_j (perhaps of different sizes) such that $H(R_i \bigcup R_j)$ = true, merge these regions.

5.4.1 State-Space Approach to Region Growing

The "classical" state-space approach of artificial intelligence [Nilsson 1971, 1980] was first applied to region growing in [Brice and Fennema 1970] and significantly extended in [Feldman and Yakimovsky 1974]. This approach regards the initial two-dimensional image as a discrete state, where every sample point is a separate region. Changes of state occur when a boundary between regions is either removed or inserted. The problem then becomes one of searching allowable changes in state to find the best partition.

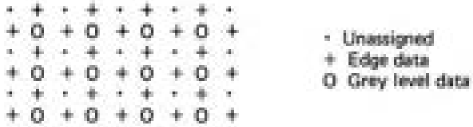


Fig. 5.6 Grid structure for region representation [Brice and Fennema 1970].

An important part of the state-space approach is the use of data structures to allow regions and boundaries to be manipulated as units. This moves away from earlier techniques, which labeled each individual pixel according to its region. The high-level data structures do away with this expensive practice by representing regions with their boundaries and then keeping track of what happens to these boundaries during split-and merge-operations.

5.4.2 Low-level Boundary Data Structures

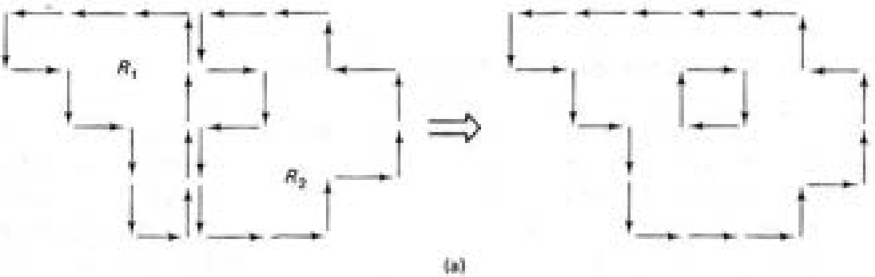
A useful representation for boundaries allows the splitting and merging of regions to proceed in a simple manner [Brice and Fennema 1970]. This representation introduces the notion of a supergrid S to the image grid G. These grids are shown in Fig. 5.6, where · and + correspond to supergrid and O to the subgrid. The representation is assumed to be four-connected (i.e., $\mathbf{x}1$ is a neighbor of $\mathbf{x}2$ if $\|\mathbf{x}1 - \mathbf{x}2\| \leqslant 1$).

With this notation boundaries of regions are directed crack edges (see Sec. 3.1) at the points marked +. That is, if point $\mathbf{x}_k$ is a neighbor of $\mathbf{x}_j$ and $\mathbf{x}_k$ is in a different region than $\mathbf{x}_j$, insert two edges for the boundaries of the regions containing $\mathbf{x}_j$ and $\mathbf{x}_k$ at the point + separating them, such that each edge traverses its associated region in a counterclockwise sense. This makes merge operations very simple: To merge regions R_k and R_l, remove edges of the opposite sense from the boundary as shown in Fig. 5.7a. Similarly, to split a region along a line, insert edges of the opposite sense in nearby points, as shown in Fig. 5.7b.

The method of [Brice and Fennema 1970] uses three criteria for merging regions, reflecting a transition from local measurements to global measurements. These criteria use measures of boundary strength s_{ij} and w_{ij} defined as

$$s_{ij} = |f(\mathbf{x}_i) - f(\mathbf{x}_j)| \tag{5.9}$$

$$w_k = \begin{cases} 1 & \text{if } s_k < T_1 \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$



(a)

Fig. 5.7 Region operations on the grid structure of Fig. 5.6.

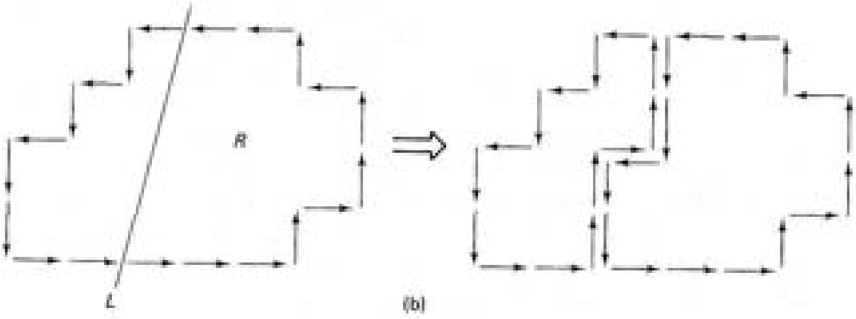


Fig. 5.7 (cont.)

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are assumed to be on either side of a crack edge (Chapter 3). The three criteria are applied sequentially in the following algorithm:

Algorithm 5.4: Region Growing via Boundary Melting (T_k, $k = 1, 2, 3$ are preset thresholds)

1. For all neighboring pairs of points, remove the boundary between $\mathbf{x}_i$ and $\mathbf{x}_j$ if $i \neq j$ and $w_{ij} = 1$. When no more boundaries can be removed, go to step 2.
2. Remove the boundary between R_i and R_j if

$$\frac{W}{\min\,[p_i,\, p_j]} \geqslant T_2 \tag{5.11}$$

where W is the sum of the w_{ij} on the common boundary between R_i and R_j, that have perimeters p_i and p_j respectively. When no more boundaries can be removed, go to step 3.

3. Remove the boundary between R_i and R_j if

$$W \geqslant T_3 \tag{5.12}$$

5.4.3 Graph-Oriented Region Structures

The Brice–Fennema data structure stores boundaries explicitly but does not provide for explicit representation of regions. This is a drawback when regions must be referred to as units. An adjunct scheme of region representation can be developed using graph theory. This scheme represents both regions and their boundaries explicitly, and this facilitates the storing and indexing of their semantic properties.

The scheme is based on a special graph called the *region adjacency graph*, and its "dual graph." In the region adjacency graph, nodes are regions and arcs exist between neighboring regions. This scheme is useful as a way of keeping track of regions, even when they are inscribed on arbitrary nonplanar surfaces (Chapter 9).

Consider the regions of an image shown in Fig. 5.8a. The region adjacency graph has a node in each region and an arc crossing each separate boundary segment. To allow a uniform treatment of these structures, define an artificial region that surrounds the image. This node is shown in Fig. 5.8b. For regions on a plane, the region adjacency graph is *planar* (can lie in a plane with no arcs intersecting) and its edges are undirected. The "dual" of this graph is also of interest. To constuct the dual of the adjacency graph, simply place nodes in each separate region and connect them with arcs wherever the regions are separated by an arc in the adjacency graph. Figure 5.8c shows that the dual of the region adjacency graph is like the original region boundary map; in Fig. 5.8b each arc may be associated with a specific boundary segment and each node with a junction between three or more boundary segments. By maintaining both the region adjacency graph and its dual, one can merge regions using the following algorithm:

Algorithm 5.5: Merging Using the Region-Adjacency Graph and Its Dual

Task: Merge neighboring regions R_i and R_j.

Phase 1. Update the region-adjacency graph.

1. Place edges between R_i and all neighboring regions of R_j (excluding, of course, R_i) that do not already have edges between themselves and R_i.
2. Delete R_j and all its associated edges.

Phase 2. Take care of the dual.

1. Delete the edges in the dual corresponding to the borders between R_i and R_j.
2. For each of the nodes associated with these edges:
    (a) if the resultant degree of the node is less than or equal to 2, delete the node and join the two dangling edges into a single edge.
    (b) otherwise, update the labels of the edges that were associated with j to reflect the new region label i.

Figure 5.9 shows these operations.

5.5 INCORPORATION OF SEMANTICS

Up to this point in our treatment of region growers, domain-dependent "semantics" has not explicitly appeared. In other words, region-merging decisions were based on raw image data and rather weak heuristics of general applicability about the likely shape of boundaries. As in early processing, the use of domain-dependent knowledge can affect region finding. Possible interpretations of regions can affect the splitting and merging process. For example, in an outdoor scene possible region interpretations might be sky, grass, or car. This kind of knowledge is quite separate from but related to measurable region properties such as intensity

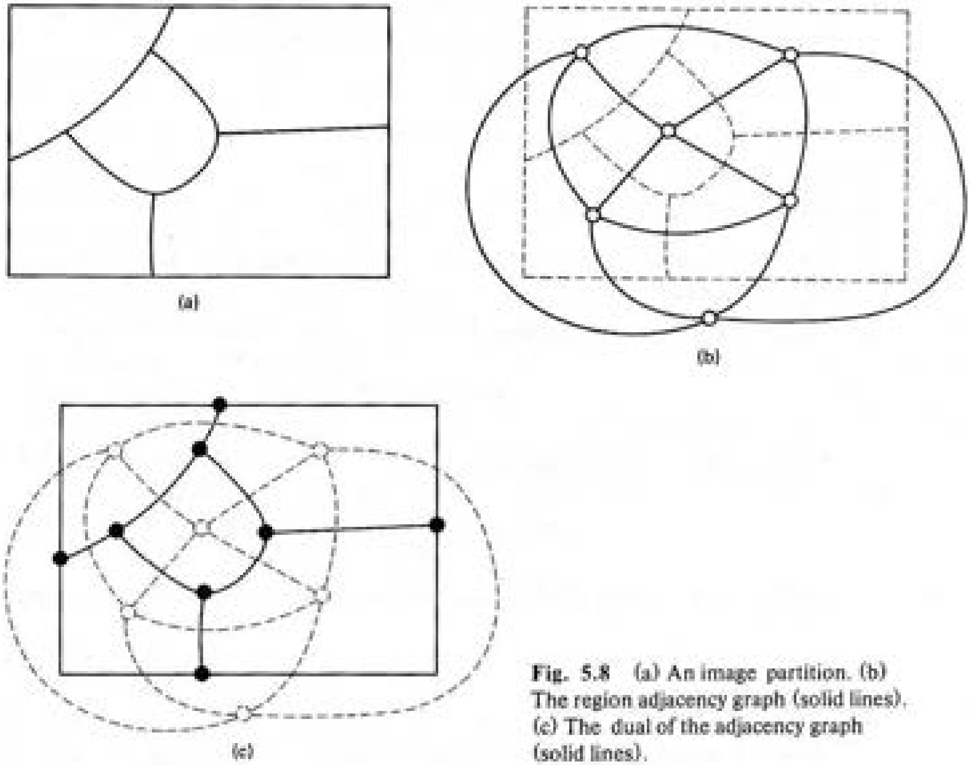


Fig. 5.8 (a) An image partition. (b) The region adjacency graph (solid lines). (c) The dual of the adjacency graph (solid lines).

and hue. An example shows how semantic labels for regions can guide the merging process. This approach was originally developed in [Feldman and Yakimovsky 1974], it has found application in several complex vision systems [Barrow and Tenenbaum 1977; Hanson and Riseman 1978].

Early steps in the Feldman–Yakimovsky region grower used essentially the same steps as Brice–Fennema. Once regions attain significant size, semantic cri-

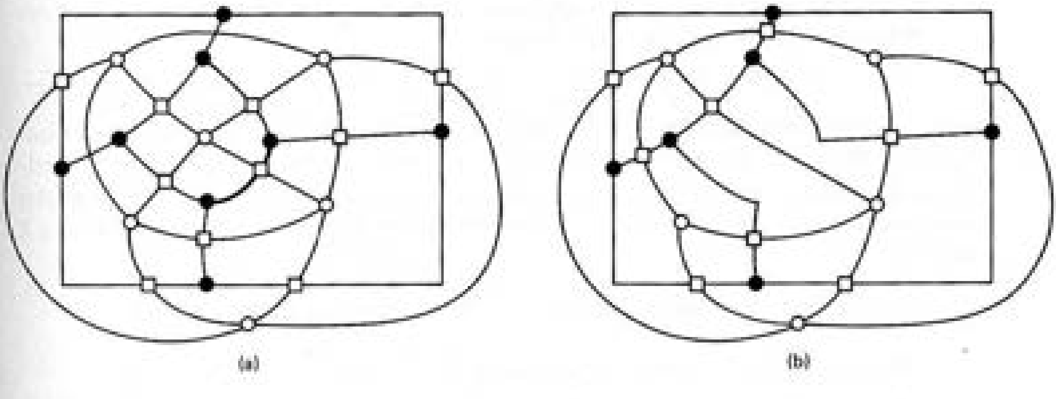


Fig. 5.9 Merging operations using the region adjacency graph and its dual. (a) Before merging regions separated by dark boundary line. (b) After merging.

teria are used. The region growing consists of four steps, as summed up in the following algorithm:

Algorithm 5.6 Semantic Region Growing

Nonsemantic Criteria

T_1 and T_2 are preset thresholds

1. Merge regions i, j as long as they have one weak separating edge until no two regions pass this test.
2. Merge regions i, j where $S(i, j) \leqslant T_2$ where

$$S(i, j) = \frac{c_1 + \alpha_{ij}}{c_2 + \alpha_{ij}}$$

where c_1 and c_2 are constants,

$$\alpha ij = \frac{(\text{area}_i)^{1/2} + (\text{area}_j)^{1/2}}{\text{perimeter}_i \cdot \text{perimeter}_j}$$

until no two regions pass this test. (This is a similar criterion to Algorithm 5.4, step 2.)

Semantic Criteria

3. Let B_{ij} be the boundary between R_i and R_j. Evaluate each B_{ij} with a Bayesian decision function that measures the (conditional) probability that B_{ij} separates two regions R_i and R_j of the *same interpretation*. Merge R_i and R_j if this conditional probability is less than some threshold. Repeat step 3 until no regions pass the threshold test.
4. Evaluate the interpretation of each region R_i with a Bayesian decision function that measures the (conditional) probability that an interpretation is the correct one for that region. Assign the interpretation to the region with the highest confidence of correct interpretation. Update the conditional probabilities for different interpretations of neighbors. Repeat the entire process until all regions have interpretation assignments.

The semantic portion of algorithm 5.6 had the goal of maximizing an evaluation function measuring the probability of a correct interpretation (labeled partition), given the measurements on the boundaries and regions of the partition. An expression for the evaluation function is (for a given partition and interpretations X and Y):

$$\begin{aligned}\max_{X, Y} \prod_{i, j} &\{P[B_{ij} \text{ is a boundary between } X \text{ and } Y \mid \text{measurements on } B_{ij}]\} \\ &\times \prod_{i} \{P[R_i \text{ is an } X \mid \text{measurements on } R_i]\} \\ &\times \prod_{j} \{P[R_j \text{ is an } Y \mid \text{measurements on } R_j]\}\end{aligned}$$

where P stands for probability and Π is the product operator.

How are these terms to be computed? Ideally, each conditional probability function should be known to a reasonable degree of accuracy; then the terms can be obtained by lookup.

However, the straightforward computation and representation of the conditional probability functions requires a massive amount of work and storage. An approximation used in [Feldman and Yakimovsky 1974] is to quantize the measurements and represent them in terms of a classification tree. The conditional probabilities can then be computed from data at the leaves of the tree. Figure 5.10 shows a hypothetical tree for the region measurements of intensity and hue, and interpretations ROAD, SKY, and CAR. Figure 5.11 shows the equivalent tree for two boundary measurements m and n and the same interpretations. These two figures indicate that $P[R_i$ is a CAR $|0 \leq i < I, 0 \leq h < H_1] =$, and $P[B_{ij}$ divides two car regions $| M_k \leq m < M_{k+1}, N_l < n \leq N_{l+1} =$. These trees were created by laborious trials with correct segmentations of test images.

Now, finally, consider again step 3 of Algorithm 5.6. The probability that a boundary B_{ij} between regions R_i and R_j is false is given by

$$P_{\text{false}} = \frac{P_f}{P_t + P_f} \tag{5.13}$$

where

$$P_f = \sum\{P[B_{ij} \text{ is between two subregions } X \mid B_{ij}\text{'s measurements}]\} \tag{5.14a}$$

$$\times\{P[R_i \text{ is } X \mid \text{meas}]\}\times\{P[R_j \text{ is } X \mid \text{meas}]\}$$

$$P_t = \sum_{X,Y} \{P[B_{ij} \text{ is between } X \text{ and } Y \mid \text{meas}]\} \tag{5.14b}$$

$$\times \{P[R_i \text{ is } X \mid \text{meas }]\}\times\{P[R_j \text{ is } Y \mid \text{meas}]\}$$

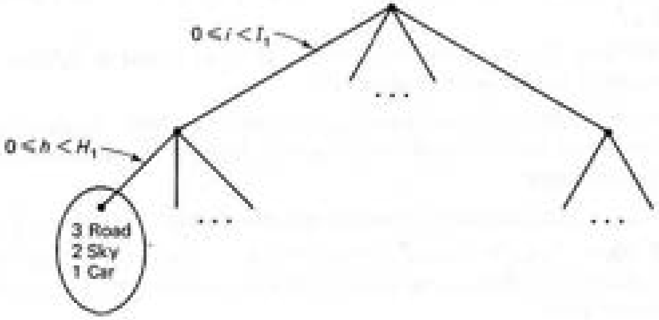


Fig. 5.10 Hypothetical classification tree for region measurements showing a particular branch for specific ranges of intensity and hue.

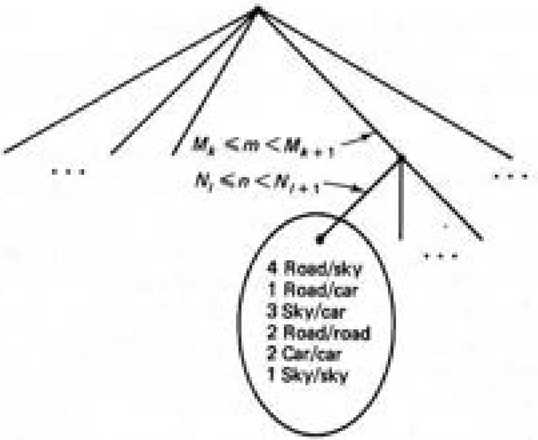


Fig. 5.11 Hypothetical classification tree for boundary measurements showing a specific branch for specific ranges of two measurements *m* and *n*.

And for step 4 of the algorithm,

$$\text{Confidence}_i = \frac{P[R_i \text{ is } X1 \mid \text{meas}]}{P[R_i \text{ is } X2 \mid \text{meas}]} \tag{5.15}$$

where $X1$, $X2$ are the first and second most likely interpretations, respectively. After the region is assigned interpretation $X1$, the neighbors are updated using

$$\begin{aligned} P[R_i \text{ is } X \mid \text{meas}] :=& \, \mathit{Prob}\, [Rj \text{ is } X \mid \text{meas}] \\ &\times P[B_{ij} \text{ is between } X \text{ and } X1 \mid \text{meas}] \end{aligned} \tag{5.16}$$

EXERCISES

5.1 In Algorithm 5.1, show how one can handle the case where colors are equivalent. Do you need more than one pass over the image?

5.2 Show for the heuristic of Eq. (5.11) that

(a) $IT_2 \geqslant WT_2 > P_j$

(b) $P_m < P_i + I(1/T_2 - 2)$

where P_m is the perimeter of $R_i \bigcup R_j$, I is the perimeter common to both i and j and $P_m = \min(P_i, P_j)$. What does part (b) imply about the relation between T_2 and P_m?

5.3 Write a "histogram-peak" finder; that is, detect satisfying valleys in histograms separating intuitive hills or peaks.

5.4 Suppose that regions are represented by a neighbor list structure. Each region has an associated list of neighboring regions. Design a region-merging algorithm based on this structure.

5.5 Why do junctions of regions in segmented images tend to be trihedral?

5.6 Regions, boundaries, and junctions are the structures behind the region-adjacency graph and its dual. Generalize these structures to three dimensions. Is another structure needed?

5.7 Generalize the graph of Figure 5.8 to three dimensions and develop the merging algorithm analogous to Algorithm 5.5. (Hint: see Exercise 5.6.)

REFERENCES

Barrow, H. G. and J. M. Tenenbaum. "Experiments in model-driven scene segmentation." *Artificial Intelligence 8*, 3, June 1977, 241–274.

Brice, C. and C. Fennema. "Scene analysis using regions." *Artificial Intelligence 1*, 3, Fall 1970, 205–226.

Chow, C. K. and T. Kaneko. "Automatic boundary detection of the left ventricle from cineangiograms." *Computers and Biomedical Research 5*, 4, August 1972, 388–410.

Feldman, J. A. and Y. Yakimovsky. "Decision theory and artificial intelligence: I. A semantics-based region analyzer." *Artificial Intelligence 5*, 4, 1974, 349–371.

Hanson, A. R. and E. M. Riseman. "Segmentation of natural scenes." In *CVS*, 1978.

Harlow, C. A. and S. A. Eisenbeis. "The analysis of radiographic images." *IEEE Trans. Computers 22*, 1973, 678–688.

Horowitz, S. L. and T. Pavlidis. "Picture segmentation by a directed split-and-merge procedure." *Proc.*, 2nd IJCPR, August 1974, 424–433.

Nilsson, N. J. *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980.

Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence.* New York: McGraw-Hill, 1971.

Ohlander, R., K. Price, and D. R. Reddy. "Picture segmentation using a recursive region splitting method." *CGIP 8*, 3, December 1979.

Robertson, T. V., P. H. Swain, and K. S. Fu. "Multispectral image partitioning." TR-EE 73-26 (LARS Information Note 071373), School of Electrical Engineering, Purdue Univ., August 1973.

Rosenfeld, A. and A. C. Kak. *Digital Picture Processing.* New York: Academic Press, 1976.

Zucker, S. W. "Region growing: Childhood and adolescence." *CGIP 5*, 3, September 1976, 382–399.

Texture 6

6.1 WHAT IS TEXTURE?

The notion of texture admits to no rigid description, but a dictionary definition of texture as "something composed of closely interwoven elements" is fairly apt. The description of interwoven elements is intimately tied to the idea of texture resolution, which one might think of as the average amount of pixels for each discernable texture element. If this number is large, we can attempt to describe the individual elements in some detail. However, as this number nears unity it becomes increasingly difficult to characterize these elements individually and they merge into less distinct spatial patterns. To see this variability, we examine some textures.

Figure 6.1 shows "cane," "paper," "coffee beans," "brickwall," "coins," and "wire braid" after Brodatz's well-known book [Brodatz 1966]. Five of these examples are high-resolution textures: they show repeated primitive elements that exhibit some kind of variation. "Coffee beans," "brick wall" and "coins" all have obvious primitives (even if it is not so obvious how to extract these from image data). Two more examples further illustrate that one sometimes has to be creative in defining primitives. In "cane" the easiest primitives to deal with seem to be the physical holes in the texture, whereas in "wire braid" it might be better to model the physical relations of a loose weave of metallic wires. However, the paper texture does not fit nicely into this mold. This is not to say that there are not possibilities for primitive elements. One is regions of lightness and darkness formed by the ridges in the paper. A second possibility is to use the reflectance models described in Section 3.5 to compute "pits" and "bumps." However, the elements seem to be "just beyond our perceptual resolving power" [Laws 1980], or in our terms, the elements are very close in size to individual pixels.

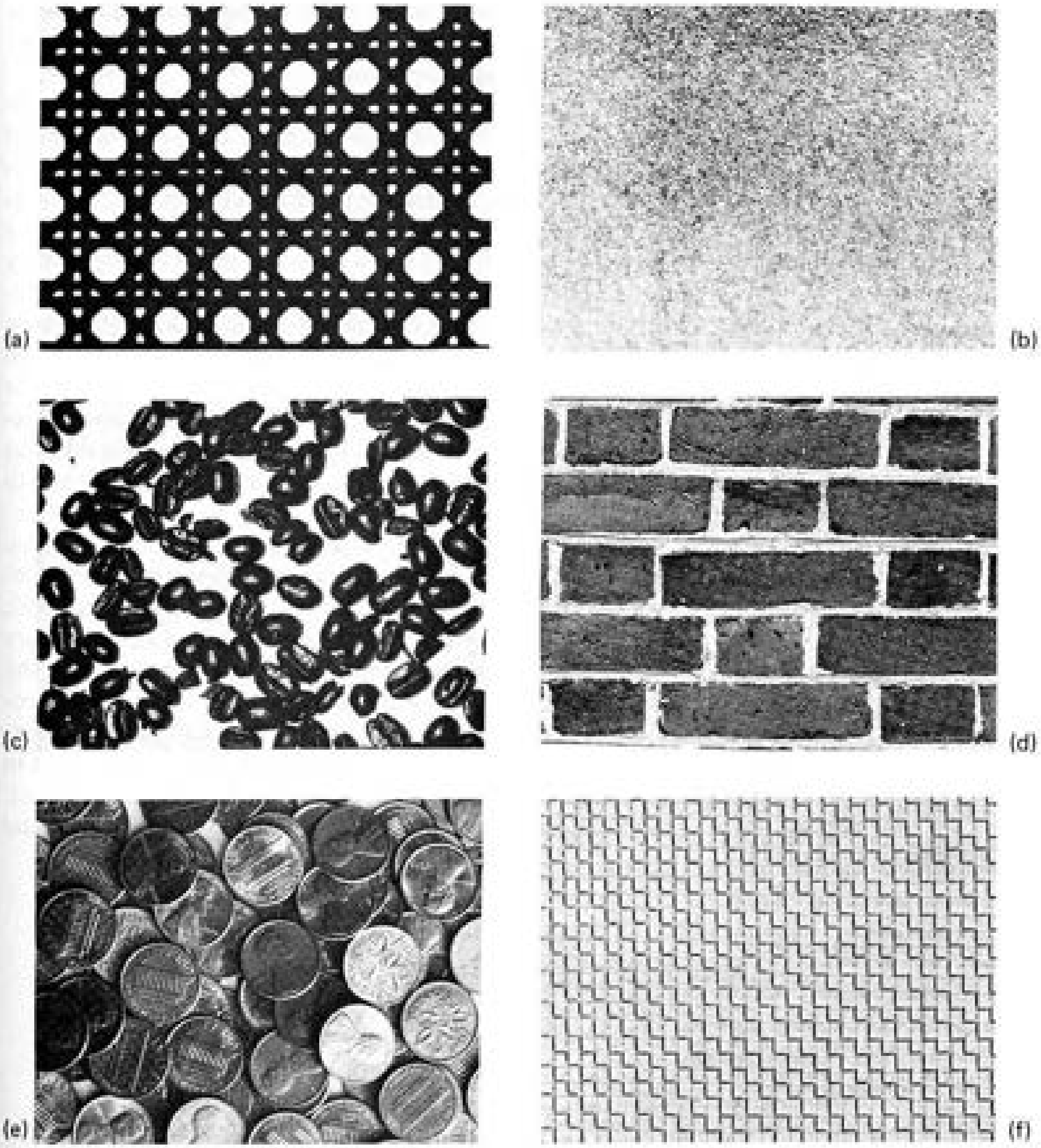


Fig. 6.1 Six examples of texture. (a) Cane. (b) Paper. (c) Coffee beans. (d) Brick wall. (e) Coins. (f) Wire braid.

The exposition of texture takes place under four main headings:

1. Texture primitives
2. Structural models
3. Statistical models
4. Texture gradients

We have already described texture as being composed of elements of *texture primitives.* The main point of additional discussion on texture primitives is to refine the idea of a primitive and its relation to image resolution.

The main work that is unique to texture is that which describes how primitives are related to the aim of recognizing or classifying the texture. Two broad classes of techniques have emerged and we shall study each in turn. The *structural* model regards the primitives as forming a repeating pattern and describes such patterns in terms of rules for generating them. Formally, these rules can be termed a grammar. This model is best for describing textures where there is much regularity in the placement of primitive elements and the texture is imaged at high resolution. The "reptile" texture in Fig. 6.9 is an example that can be handled by the structured approach. The *statistical* model usually describes texture by statistical rules governing the distribution and relation of gray levels. This works well for many natural textures which have barely discernible primitives. The "paper" texture is such an example. As we shall see, we cannot be too rigid about this division since statistical models can describe pattern-like textures and vice versa, but in general the dichotomy is helpful.

The examples suggest that texture is almost always a property of *surfaces.* Indeed, as the example of Fig. 6.2 shows, human beings tend to relate texture elements of varying size to a plausible surface in three dimensions [Gibson 1950; Stevens 1979]. Techniques for determining surface orientation in this fashion are termed texture *gradient* techniques. The gradient is given both in terms of the direction of greatest change in size of primitives and in terms of the spatial placement of primitives. The notion of a gradient is very useful. For example, if the texture is embedded on a flat surface, the gradient points toward a vanishing point in the image. The chapter concludes with algorithms for computing this gradient. The gradient may be computed directly or indirectly via the computation of the vanishing point.

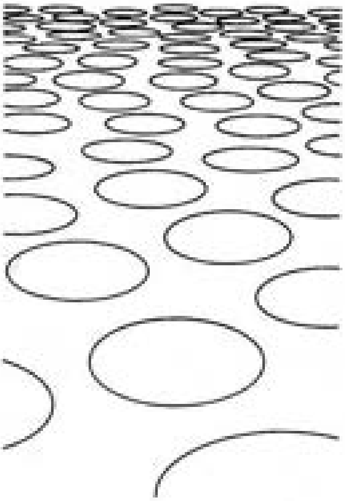


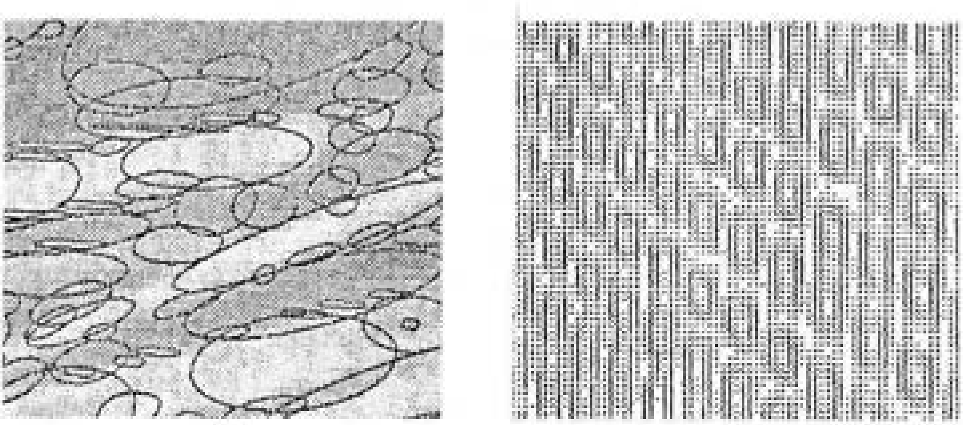
Fig. 6.2 Texture as a surface property.

6.2 TEXTURE PRIMITIVES

The notion of a primitive is central to texture. To highlight its importance, we shall use the appelation *texel* (for texture element) [Kender 1978]. A texel is (loosely) a visual primitive with certain invariant properties which occurs repeatedly in different positions, deformations, and orientations inside a given area. One basic invariant property of such a unit might be that its pixels have a constant gray level, but more elaborate properties related to shape are possible. (A detailed discussion of planar shapes is deferred until Chapter 8.) Figure 6.3 shows examples of two kinds of texels: (a) ellipses of approximately constant gray level and (b) linear edge segments. Interestingly, these are nearly the two features selected as texture primitives by [Julesz, 1981], who has performed extensive studies of human texture perception.

For textures that can be described in two dimensions, image-based descriptions are sufficient. Texture primitives may be pixels, or aggregates of pixels such as curve segments or regions. The "coffee beans" texture can be described by an image-based model: repeated dark ellipses on a lighter background. These models describe equally well an image of texture or an image of a picture of texture. The methods for creating these aggregates were discussed in Chapters 4 and 5. As with all image-based models, three-dimensional phenomena such as occlusion must be handled indirectly. In contrast, structural approaches to texture sometimes require knowledge of the three-dimensional world producing the texture image. One example of this is Brodatz's "coins" shown in Fig. 6.1. A three-dimensional model of the way coins can be stacked is needed to understand this texture fully.

An important part of the texel definition is that primitives must occur repeatedly inside a given area. The question is: How many times? This can be answered qualitatively by imagining a window that corresponds approximately to our field of view superimposed on a very large textured area. As this window is made smaller, corresponding to moving the viewpoint closer to the texture, fewer and fewer texels are contained in it. At some distance, the image in the window no longer


(a)

(b)

Fig. 6.3 Examples of texels. (a) Ellipses. (b) Linear segments.

appears textured, or if it does, translation of the window changes the perceived texture drastically. At this point we no longer have a texture. A similar effect occurs if the window is made increasingly larger, corresponding to moving the field of view farther away from the image. At some distance textural details are blurred into continuous tones and repeated elements are no longer visible as the window is translated. (This is the basis for halftone images, which are highly textured patterns meant to be viewed from enough distance to blur the texture.) Thus the idea of an appropriate *resolution*, or the number of texels in a subimage, is an implicit part of our qualitative definition of texture. If the resolution is appropriate, the texture will be apparent and will "look the same" as the field of view is translated across the textured area. Most often the appropriate resolution is not known but must be computed. Often this computation is simpler to carry out than detailed computations characterizing the primitives and hence has been used as a precursor to the latter computations. Figure 6.4 shows such a resolution-like computation, which examines the image for repeating peaks [Connors 1979].

Textures can be hierarchical, the hierarchies corresponding to different resolutions. The "brick wall" texture shows such a hierarchy. At one resolution, the highly structured pattern made by collections of bricks is in evidence; at higher resolution, the variations of the texture of each brick are visible.

6.3 STRUCTURAL MODELS OF TEXEL PLACEMENT

Highly patterned textures tesselate the plane in an ordered way, and thus we must understand the different ways in which this can be done. In a regular tesselation the

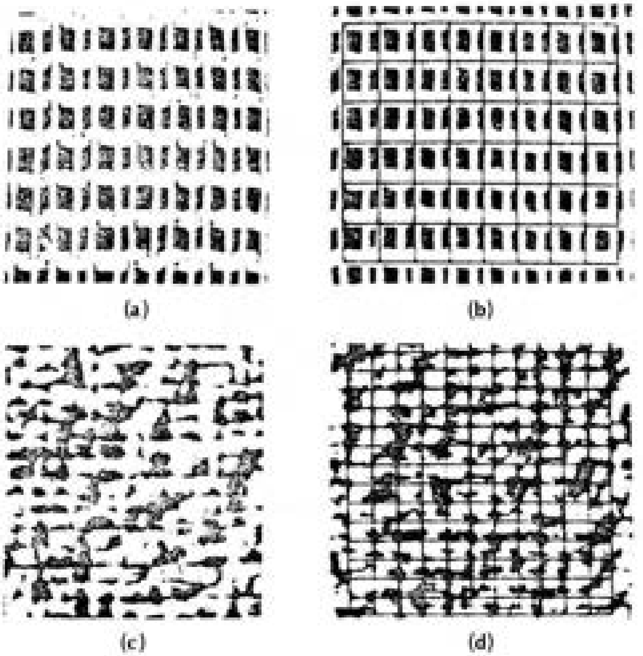
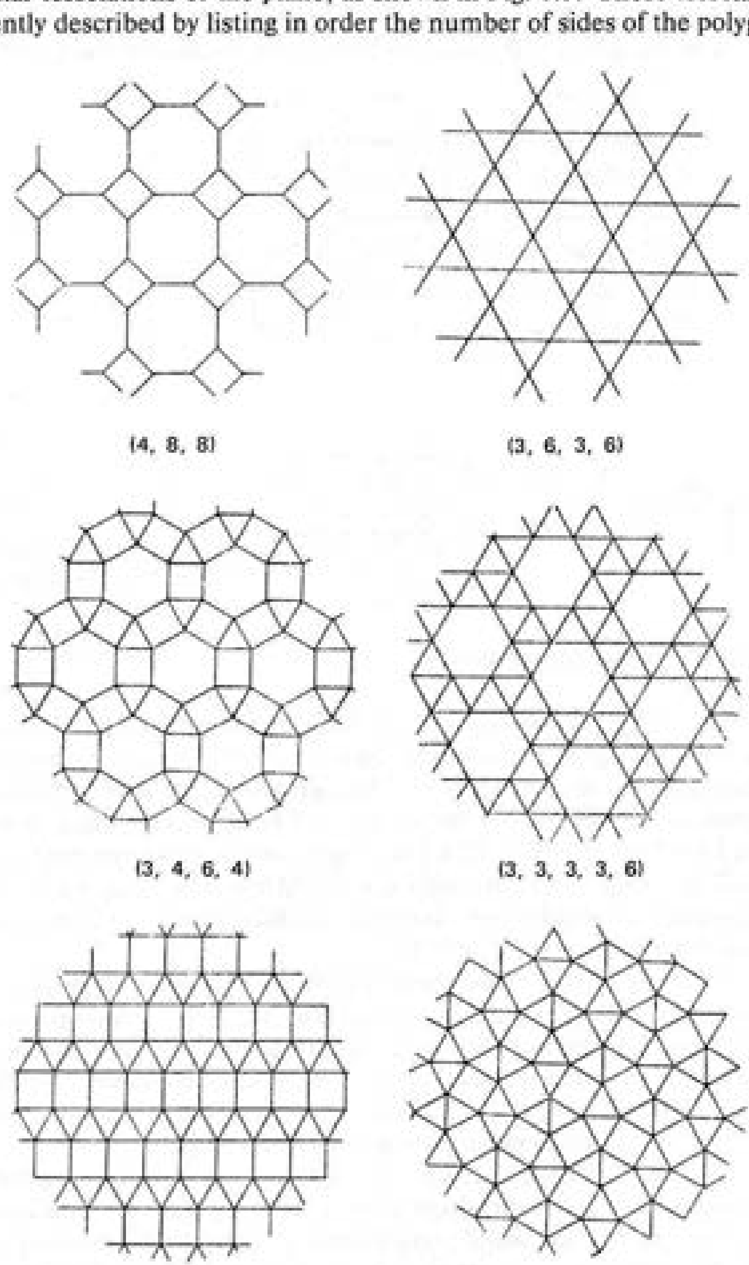


Fig. 6.4 Computing texture resolutions. (a) French canvas. (b) Resolution grid for canvas. (c) Raffia. (d) Grid for raffia.

polygons surrounding a vertex all have the same number of sides. Semiregular tesselations have two kinds of polygons (differing in number of sides) surrounding a vertex. Figure 2.11 depicts the regular tesselations of the plane. There are eight semiregular tesselations of the plane, as shown in Fig. 6.5. These tesselations are conveniently described by listing in order the number of sides of the polygons sur-



(4, 8, 8)

(3, 6, 3, 6)

(3, 4, 6, 4)

(3, 3, 3, 3, 6)

(3, 3, 3, 4, 4)

(3, 3, 4, 3, 4)

Fig. 6.5 Semiregular tesselations.

rounding each vertex. Thus a hexagonal tesselation is described by (6,6,6) and every vertex in the tesselation of Fig. 6.5 can be denoted by the list (3,12,12). It is important to note that the tesselations of interest are those which describe the *placement* of primitives rather than the primitives themselves. When the primitives define a tesselation, the tesselation describing the primitive placement will be the dual of this graph in the sense of Section 5.4. Figure 6.6 shows these relationships.

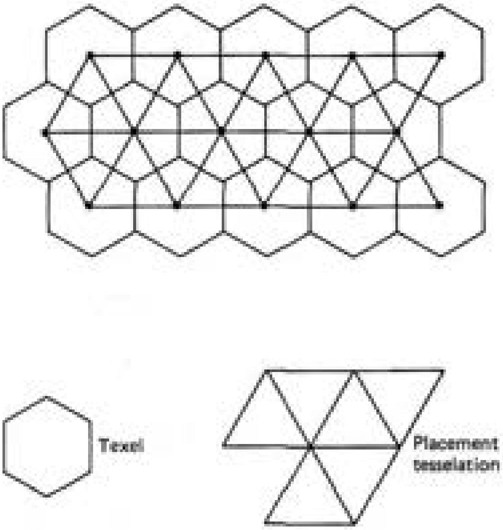


Fig. 6.6 *The primitive placement tesselation as the dual of the primitive tesselation.*

6.3.1 Grammatical Models

A powerful way of describing the rules that govern textural structure is through a grammar. A grammar describes how to generate patterns by applying *rewriting rules* to a small number of *symbols.* Through a small number of rules and symbols, the grammar can generate complex textural patterns. Of course, the symbols turn out to be related to texels. The mapping between the stored model prototype texture and an image of texture with real-world variations may be incorporated into the grammar by attaching probabilities to different rules. Grammars with such rules are termed *stochastic* [Fu 1974].

There is no unique grammar for a given texture; in fact, there are usually infinitely many choices for rules and symbols. Thus texture grammars are described as *syntactically ambiguous.* Figure 6.7 shows a syntactically ambiguous texture and two of the possible choices for primitives. This texture is also *semantically ambiguous* [Zucker 1976] in that alternate ridges may be thought of in three dimensions as coming out of or going into the page.

There are many variants of the basic idea of formal grammars and we shall examine three of them: shape grammars, tree grammars, and array grammars. For a basic reference, see [Hopcroft and Ullman 1979]. Shape grammars are distinguished from the other two by having high-level primitives that closely correspond to the shapes in the texture. In the examples of tree grammars and array grammars that we examine, texels are defined as pixels and this makes the

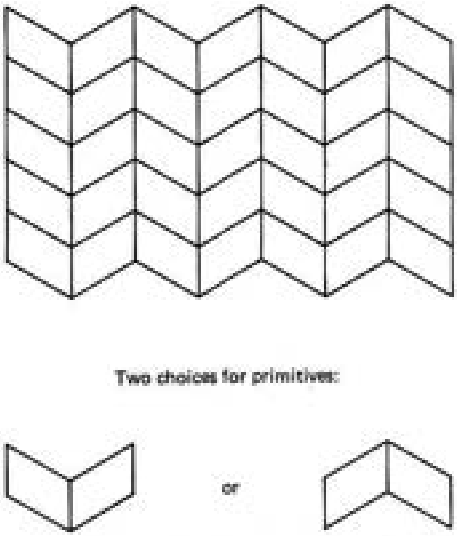


Fig. 6.7 Ambiguous texture.

grammars correspondingly more complicated. A particular texture that can be described in eight rules in a shape grammar requires 85 rules in a tree grammar [Lu and Fu 1978]. The compensating trade-off is that pixels are gratis with the image; considerable processing must be done to derive the more complex primitives used by the shape grammar.

6.3.2 Shape Grammars

A shape grammar [Stiny and Gips 1972] is defined as a four-tuple $< V_t, V_m, R, S>$ where:

1. V_t is a finite set of shapes
2. V_m is a finite set of shapes such that $V_t \cap V_m = \phi$
3. R is a finite set of ordered pairs (u, v) such that u is a shape consisting of elements of V_t^+ and v is a shape consisting of an element of V_t^* combined with an element of V_m^*
4. S is a shape consisting of an element of V_t^* combined with an element of V_m^*.

Elements of the set V_t are called terminal shape elements (or terminals). Elements of the set V_m are called nonterminal shape elements (or markers). The sets V_t and V_m must be disjoint. Elements of the set V_t^+ are formed by the finite arrangement of one or more elements of V_t in which any elements and/or their mirror images may be used a multiple number of times in any location, orientation, or scale. The set $V_t^* = V_t^+ \cup \{\Lambda\}$, where Λ is the empty shape. The sets V_m^+ and V_m^* are defined similarly. Elements (u, v) of R are called shape rules and are written $u\ v$. $u$is called the left side of the rule; v the right side of the rule. u and v usually are enclosed in identical dashed rectangles to show the correspondence between the two shapes. S is called the initial shape and normally contains a u such that there is a (u, v) which is an element of R.

A texture is generated from a shape grammar by beginning with the initial shape and repeatedly applying the shape rules. The result of applying a shape rule R to a given shape s is another shape, consisting of s with the right side of R substituted in S for an occurrence of the left side of R. Rule application to a shape proceeds as follows:

1. Find part of the shape that is geometrically similar to the left side of a rule in terms of both terminal elements and nonterminal elements (markers). There must be a one-to-one correspondence between the terminals and markers in the left side of the rule and the terminals and markers in the part of the shape to which the rule is to be applied.
2. Find the geometric transformations (scale, translation, rotation, mirror image) which make the left side of the rule identical to the corresponding part in the shape.
3. Apply those transformations to the right side of the rule.
4. Substitute the transformed right side of the rule for the part of the shape that corresponds to the left side of the rule.

The generation process is terminated when no rule in the grammar can be applied.

As a simple example, one of the many ways of specifying a hexagonal texture $\{V_t, V_m, R, S\}$ is

$V_t = \{$ ⬡ $\}$

$V_m = \{\ \cdot\ \}$ (6.1)

R: ⬡ → ⬡⬡ ; ⬡⬡ ;etc.

$S = \{$ ⬡ $\}$

Hexagonal textures can be *generated* by the repeated application of the single rule in R. They can be *recognized* by the application of the rule in the opposite direction to a given texture until the initial shape, I, is produced. Of course, the rule will generate only hexagonal textures. Similarly, the hexagonal texture in Fig. 6.8a will be recognized but the variants in Fig. 6.8b will not.

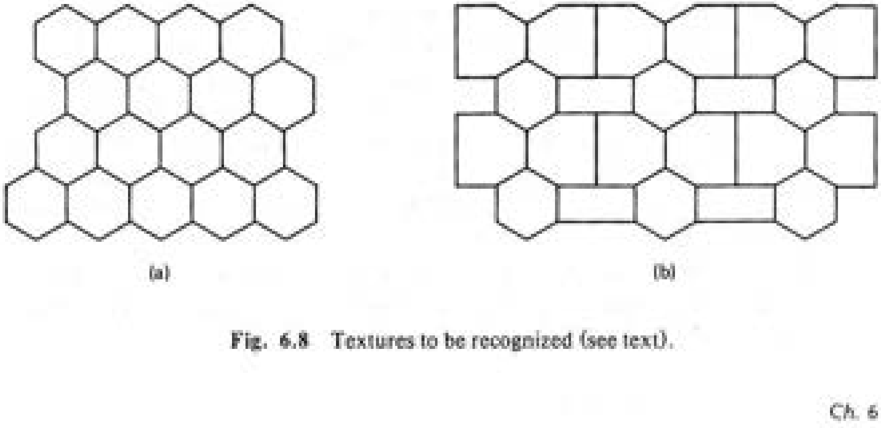
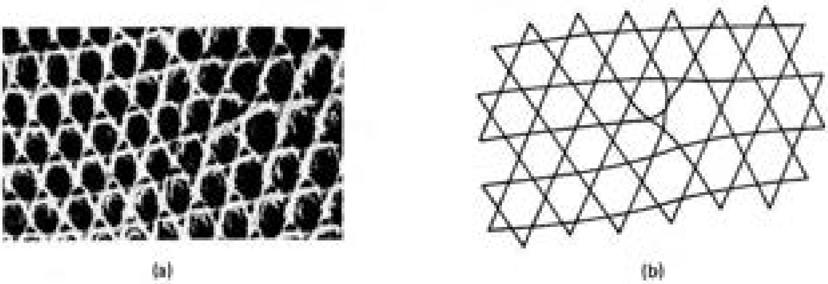


Fig. 6.8 Textures to be recognized (see text).

A more difficult example is given by the "reptile" texture. Except for the occasional new rows, a (3, 6, 3, 6) tesselation of primitives would model this texture exactly. As shown in Fig. 6.9, the new row is introduced when a seven-sided polygon splits into a six-sided polygon and a five-sided polygon. To capture this with a shape grammar, we examine the dual of this graph, which is the primitive placement graph, Fig. 6.9b. This graph provides a simple explanation of how the extra row is created; that is, the diamond pattern splits into two. Notice that the dual graph is composed solely of four-sided polygons but that some vertices are (4, 4, 4) and some are (4, 4, 4, 4, 4, 4). A shape grammar for the dual is shown in Fig. 6.10. The image texture can be obtained by forming the dual of this graph. One further refinement should be added to rules (6) and (7); so that rule (7) is used less often, the appropriate probabilities should be associated with each rule. This would make the grammar stochastic.


(a)

(b)

Fig. 6.9 (a) The reptile texture. (b) The reptile texture as a (3, 6, 3, 6) semiregular tesselation with local deformations.

6.3.3 Tree Grammars

The symbolic form of a tree grammar is very similar to that of a shape grammar. A grammar

$$G_t = (V_t,\ V_m,\ r,\ R,\ S)$$

is a tree grammar if

V_t is a set of terminal symbols
V_m is a set of symbols such that
$V_m \cap V_t = \phi$
$r : V_t \rightarrow N$ (where N is the set of nonnegative integers)
is the rank associated with symbols in V_t
S is the start symbol
R is the set of rules of the form

$$X_0 \rightarrow \begin{array}{c} x \\ \diagup \quad | \\ X_0 \cdots X_{r(x)} \end{array} \qquad \text{or} \qquad X_0 \rightarrow x$$

with x in V_t and $X_0 \ldots X_{r(x)}$ in V_m

For a tree grammar to generate arrays of pixels, it is necessary to choose some way of embedding the tree in the array. Figure 6.11 shows two such embeddings.

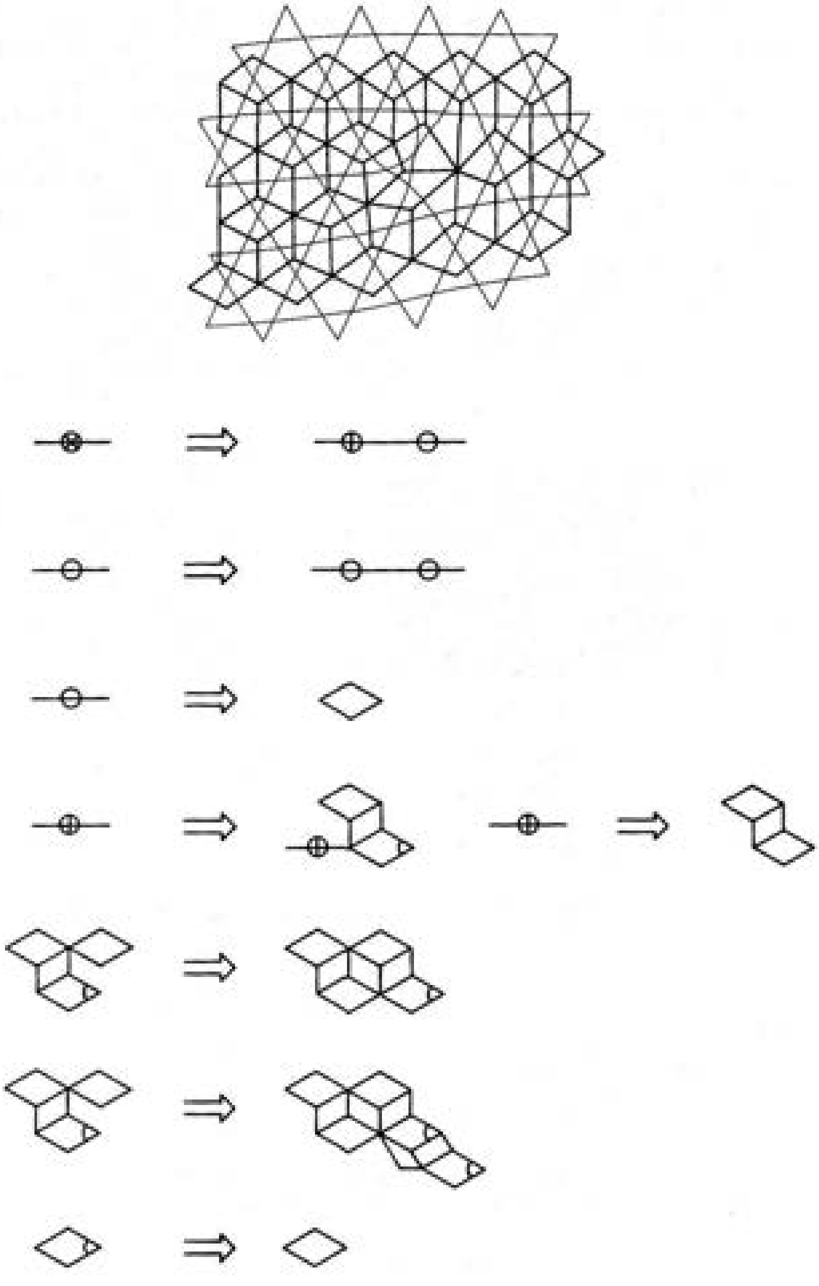
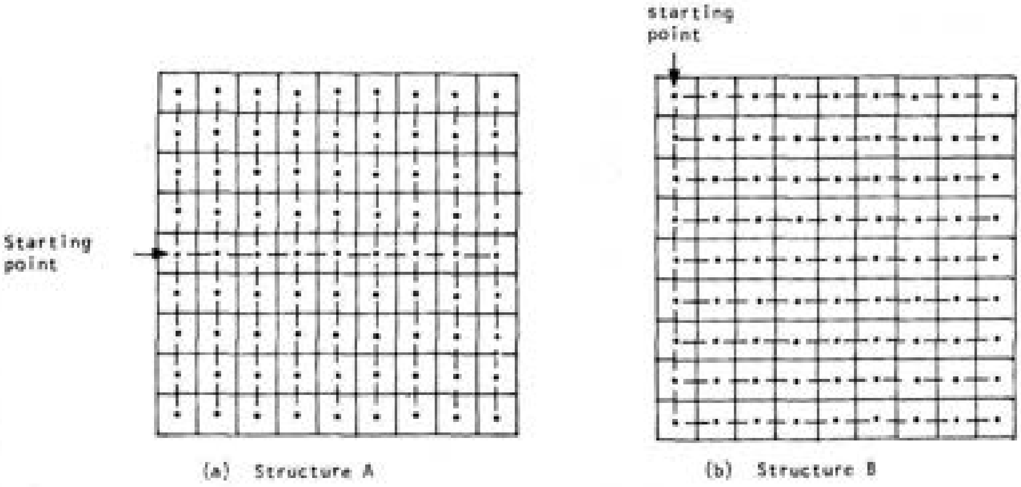


Fig. 6.10 Shape grammar for the reptile texture.

In the application to texture [Lu and Fu 1978], the notion of pyramids or hierarchical levels of resolution in texture is used. One level describes the placement of repeating patterns in texture windows—a rectangular texel placement tesselation—and another level describes texels in terms of pixels. We shall illus-



(a) Structure A

(b) Structure B

Fig. 6.11 Two ways of embedding a tree structure in an array.

trate these ideas with Lu and Fu's grammar for "wire braid." The texture windows are shown in Fig. 6.12a. Each of these can be described by a "sentence" in a second tree grammar. The grammar is given by:

$$G_w = (V_t, V_m, r, R, S)$$

where

$$\begin{aligned} V_t &= \{A_1, C_1\} \\ V_m &= \{X, Y, Z\} \\ r &= \{0, 1, 2\} \\ R: X &\rightarrow \begin{array}{c} A_1 \\ \diagup \quad \diagdown \\ X \qquad Y \end{array} \quad \text{or } \begin{array}{c} A_1 \\ | \\ Y \end{array} \\ Y &\rightarrow \begin{array}{c} C_1 \\ | \\ Z \end{array} \quad \text{or } C_1 \\ Z &\rightarrow \begin{array}{c} A_1 \\ | \\ Y \end{array} \quad \text{or } A_1 \end{aligned} \tag{6.2}$$

and the first embedding in Fig. 6.11 is used. The pattern inside each of these windows is specified by another grammatical level:

$$G = (V_t, V_m, r, R, S)$$

where

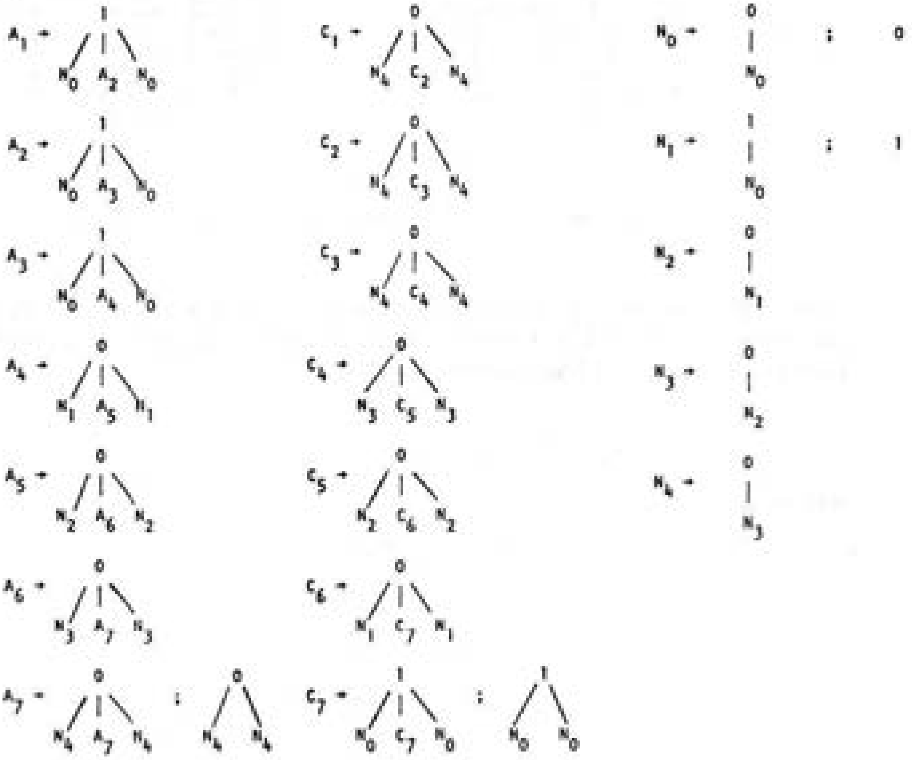
$$V_t = \{1, 0\}$$

$$V_m = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, C_1, C_2, C_3, C_4, C_5, C_6, C_7, N_0, N_1, N_2, N_3, N_4\}$$

$$r = \{0, 1, 2\}$$

$$S = \{A_1, C_1\}$$

R:



The application of these rules generates the two different patterns of pixels shown in Fig. 6.13.

6.3.4 Array Grammars

Like tree grammars, array grammars use hierarchical levels of resolution [Milgram and Rosenfeld 1971; Rosenfeld 1971]. Array grammars are different from tree grammars in that they do not use the tree-array embedding. Instead, prodigious use of a blank or null symbol is used to make sure the rules are applied in appropriate contexts. A simple array grammar for generating a checkerboard pattern is

$$G = \{V_t, V_n, R\}$$

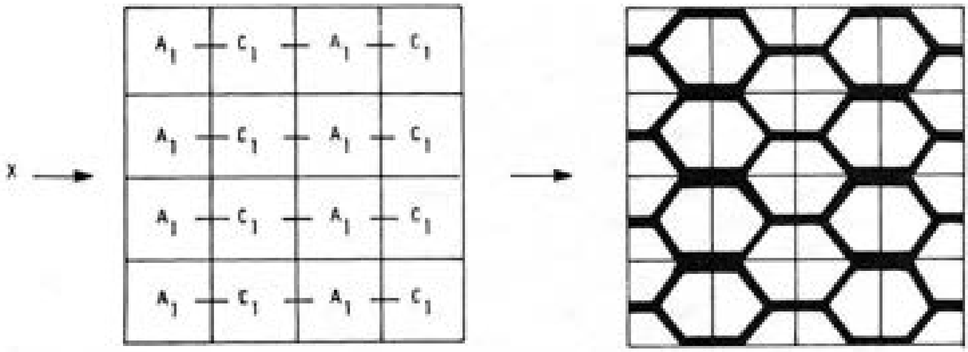


Fig. 6.12 Texture window and grammar (see text).

where

$$V_t = \{0, 1\} \text{ (corresponding to black and white pixels, respectively)}$$
$$V_n = \{b, S\}$$

b is a "blank" symbol used to provide context for the application of the rules. Another notational convenience is to use a subscript to denote the orientation of symbols. For example, when describing the rules R we use

$$0_x b \rightarrow 0_x 1 \qquad \text{where } x \text{ is one of } \{U, D, L, R\}$$

to summarize the four rules

$$\begin{matrix}0\\b\end{matrix} \rightarrow \begin{matrix}0\\1\end{matrix}, \qquad \begin{matrix}b\\0\end{matrix} \rightarrow \begin{matrix}1\\0\end{matrix}, \qquad 0b \rightarrow 01, \qquad b0 \rightarrow 10$$

Thus the checkerboard rule set is given by

$$R: S \rightarrow 0 \text{ or } 1$$
$$0_x b \rightarrow 0_x 1 \qquad x \text{ in } \{U, D, L, R\}$$
$$1_x b \rightarrow 1_x 0$$

A compact encoding of textural patterns [Jayaramamurthy 1979] uses levels of array grammars defined on a pyramid. The terminal symbols of one layer are the start symbols of the next grammatical layer defined lower down in the pyramid. This corresponds nicely to the idea of having one grammar to generate primitives and another to generate the primitive placement tesselations.

As another example, consider the herringbone pattern in Fig. 6.14a, which is composed of 4×3 arrays of a particular placement pattern as shown in Fig. 6.14b. The following grammar is sufficient to generate the placement pattern.

$$G_w = \{V_t, V_m, R, S\}$$

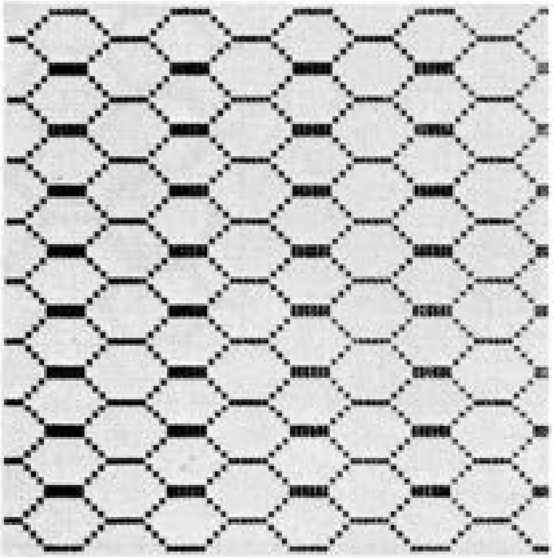


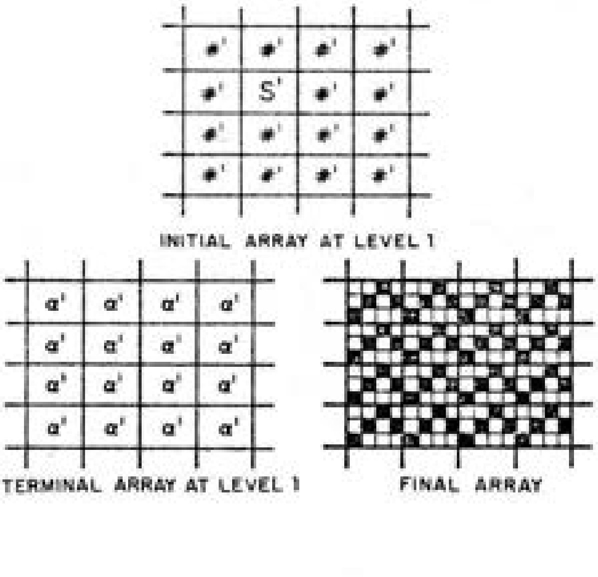
Fig. 6.13 Texture generated by tree grammar.

where

$$V_t = \{a\}$$
$$V_n = \{b, S\}$$
$$R: S \rightarrow a$$
$$a_x b \rightarrow a_x a \qquad x \text{ in } \{U, D, L, R\}$$

We have not been precise in specifying how the terminal symbol is projected onto the lower level. Assume without loss of generality that it is placed in the upper left-hand corner, the rest of the subarray being initially blank symbols. Thus a simple grammar for the primitive is

$$G_t = \{V_t, V_n, R, S\}$$



#'	#'	#'	#'
#'	S'	#'	#'
#'	#'	#'	#'
#'	#'	#'	#'

INITIAL ARRAY AT LEVEL 1

a'	a'	a'	a'
a'	a'	a'	a'
a'	a'	a'	a'
a'	a'	a'	a'

TERMINAL ARRAY AT LEVEL 1

Fig. 6.14 Steps in generating a herringbone texture with an array grammar.

where

$$V_t = \{0, 1\}$$

$$V_n = \{a, b\}$$

$$R: \begin{matrix} a & b & b & b \\ b & b & b & b \\ b & b & b & b \end{matrix} \rightarrow \begin{matrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{matrix}$$

6.4 TEXTURE AS A PATTERN RECOGNITION PROBLEM

Many textures do not have the nice geometrical regularity of "reptile" or "wire braid"; instead, they exhibit variations that are not satisfactorily described by shapes, but are best described by statistical models. *Statistical pattern recognition* is a paradigm that can classify statistical variations in patterns. (There are other statistical methods of describing texture [Pratt et al. 1981], but we will focus on statistical pattern recognition since it is the most widely used for computer vision purposes.) There is a voluminous literature on pattern recognition, including several excellent texts (e.g., [Fu 1968; Tou and Gonzalez 1974; Fukunaga 1972], and the ideas have much wider application than their use here, but they seem particularly appropriate for low-resolution textures, such as those seen in aerial images [Weszka et al. 1976]. The pattern recognition approach to the problem is to classify instances of a texture in an image into a set of classes. For example, given the textures in Fig. 6.15, the choice might be between the classes "orchard," "field," "residential," "water."

The basic notion of pattern recognition is the *feature vector*. The feature vector $\mathbf{v}$ is a set of measurements $\{v_1 \cdots v_m\}$ which is supposed to condense the description of relevant properties of the textured image into a small, Euclidean *feature space* of m dimensions. Each point in feature space represents a value for the feature vector applied to a different image (or subimage) of texture. The measurement values for a feature should be correlated with its class membership. Figure 6.16 shows a two-dimensional space in which the features exhibit the desired correlation property. Feature vector values cluster according to the texture from which they were derived. Figure 6.16 shows a bad choice of features (measurements) which does not separate the different classes.

The pattern recognition paradigm divides the problem into two phases: training and test. Usually, during a training phase, feature vectors from known samples are used to partition feature space into regions representing the different classes. However, self teaching can be done; the classifier derives its own partitions. Feature selection can be based on parametric or nonparametric models of the distributions of points in feature space. In the former case, analytic solutions are sometimes available. In the latter, feature vectors are *clustered* into groups which are taken to indicate partitions. During a test phase the feature-space partitions are used to classify feature vectors from unknown samples. Figure 6.17 shows this process.

Given that the data are reasonably well behaved, there are many methods for clustering feature vectors [Fukunaga 1972; Tou and Gonzales 1974; Fu 1974].

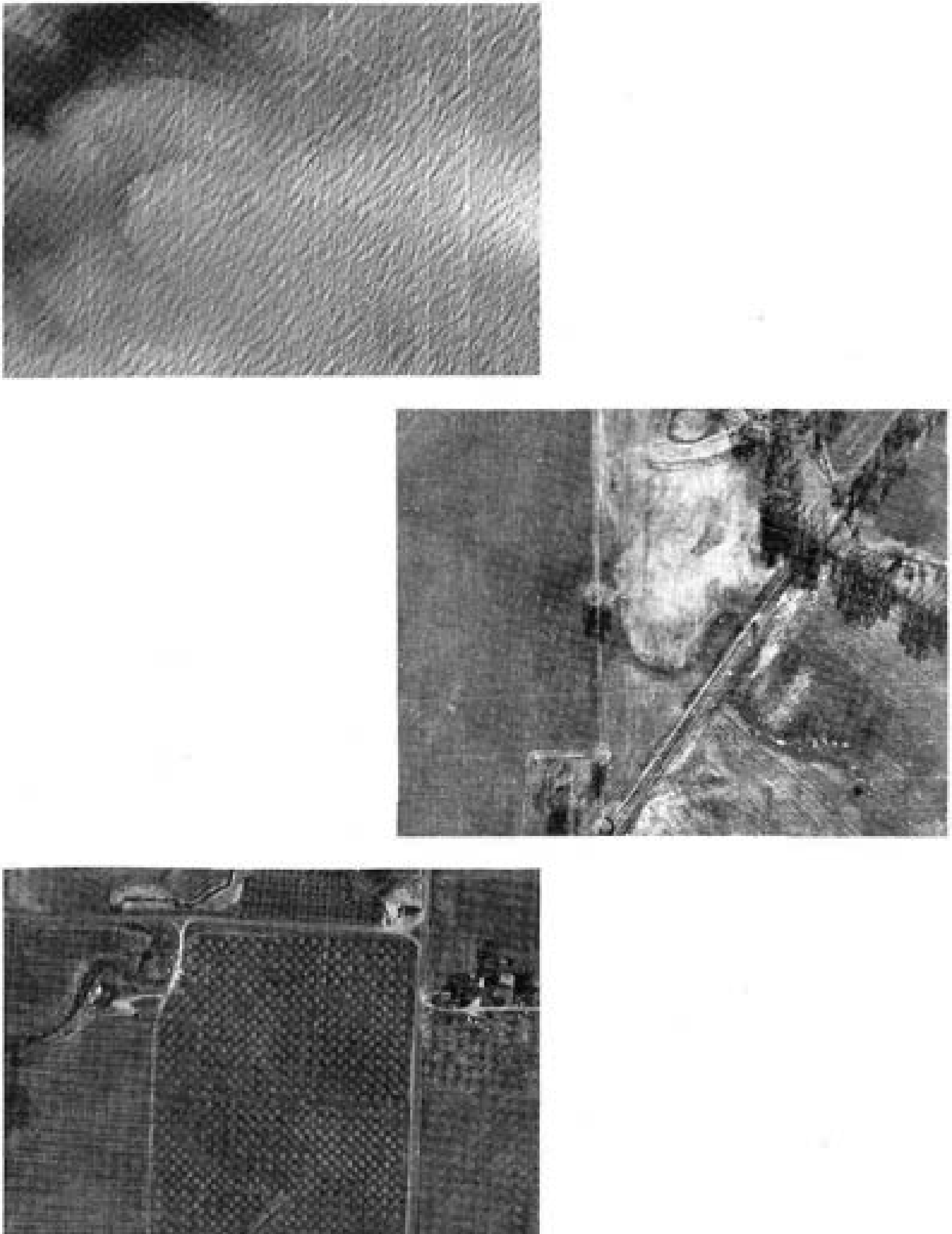


Fig. 6.15 Aerial image textures for discrimination.

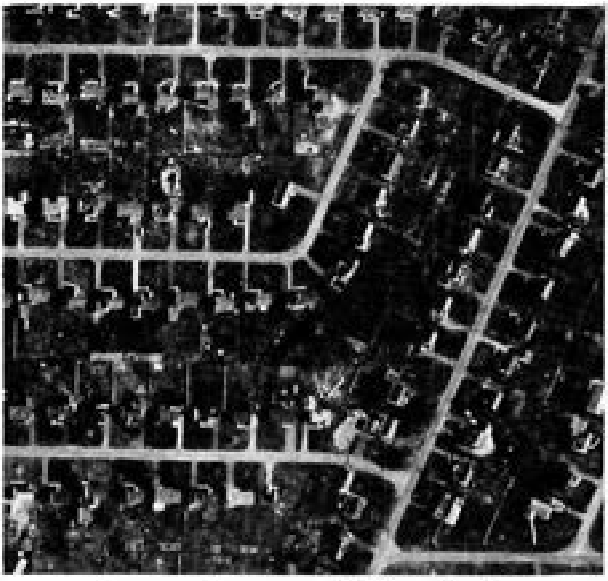


Fig. 6.15 (cont.)

One popular way of doing this is to use prototype points for each class and a *nearest-neighbor* rule [Cover 1968]:

assign $\mathbf{v}$ to class w_i if i minimizes
$\min_i d(\mathbf{v}, \mathbf{v}_{w_i})$

where $\mathbf{v}_{w_i}$ is the prototype point for class w_i.

Parametric techniques assume information about the feature vector probability distributions to find rules that maximize the likelihood of correct classification:

assign $\mathbf{v}$ to class w_i if i maximizes
$\max_i p(w_i|\mathbf{v})$

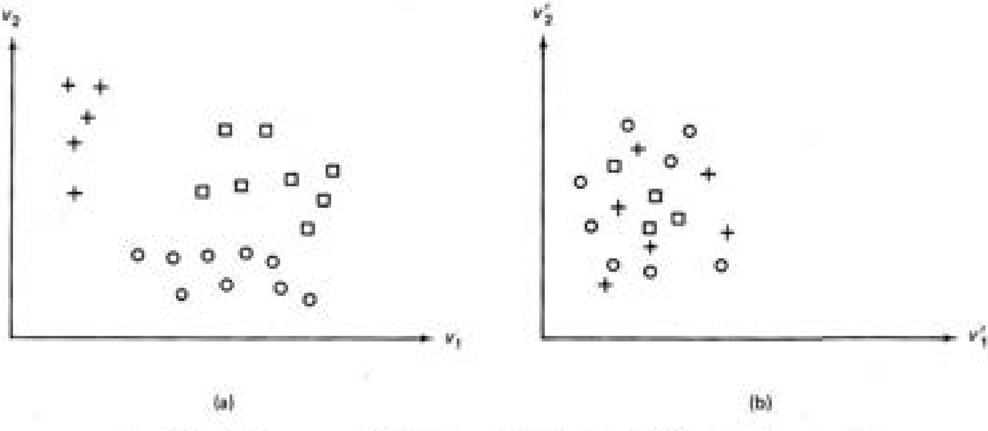


Fig. 6.16 Feature space for texture discrimination. (a) effective features (b) ineffective features.

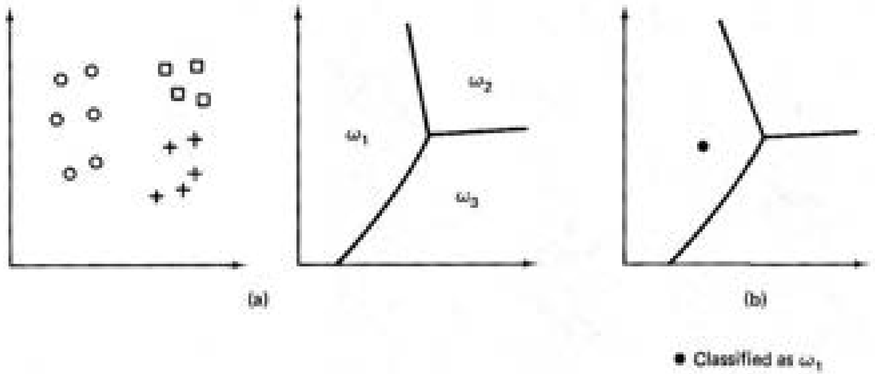


Fig. 6.17 Pattern recognition paradigm.

The distributions may also be used to formulate rules that minimize errors.

Picking good features is the essence of pattern recognition. No elaborate formalism will work well for bad features such as those of Fig. 6.15b. On the other hand, almost any method will work for very good features. For this reason, texture is a good domain for pattern recognition: it is fairly easy to define features that (1) cluster in feature space according to different classes, and (2) can separate texture classes.

The ensuing subsections describe features that have worked well. These subsections are in reverse order from those of Section 6.2 in that we begin with features defined on pixels—Fourier subspaces, gray-level dependencies—and conclude with features defined on higher-level texels such as regions. However, the lesson is the same as with the grammatical approach: hard work spent in obtaining high-level primitives can both improve and simplify the texture model. Space does not permit a discussion of many texture features; instead, we limit ourselves to a few representative samples. For further reading, see [Haralick 1978].

6.4.1 Texture Energy

Fourier Domain Basis

If a texture is at all spatially periodic or directional, its power spectrum will tend to have peaks for corresponding spatial frequencies. These peaks can form the basis of features of a pattern recognition discriminator. One way to define features is to search Fourier space directly [Bajcsy and Lieberman 1976]. Another is to partition Fourier space into bins. Two kinds of bins, radial and angular, are commonly used, as shown in Fig. 6.18. These bins, together with the Fourier power spectrum are used to define features. If F is the Fourier transform, the Fourier power spectrum is given by $|F|^2$.

Radial features are given by

$$v_{r_1 r_2} = \int\int |F(u, v)|^2 \, du \, dv \tag{6.5}$$

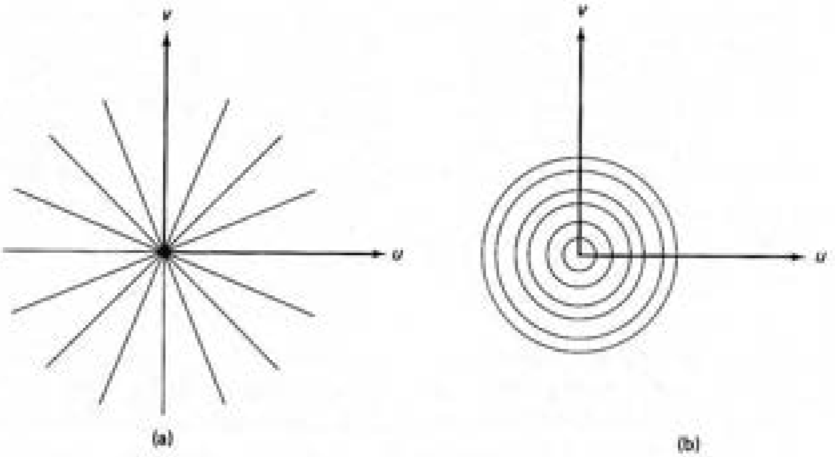


Fig. 6.18 Partitioning the Fourier domain into bins.

where the limits of integration are defined by

$$r_1^2 \leqslant u^2 + v^2 < r_2^2$$
$$0 \leqslant u, v < n-1$$

where $[r_{1,}\ r_2]$ is one of the radial bins and $\mathbf{v}$ is the vector (not related to v) defined by different values of r_1 and r_2. Radial features are correlated with texture coarseness. A smooth texture will have high values of $V_{r_1r_2}$ for small radii, whereas a coarse, grainy texture will tend to have relatively higher values for larger radii.

Features that measure angular orientation are given by

$$\mathbf{v}_{\theta_1\theta_2} = \int\int |F(u, v)|^2 \, du \, dv \tag{6.6}$$

where the limits of integration are defined by

$$\theta_1 \leqslant \tan^{-1}\left[\frac{v}{u}\right] < \theta_2$$
$$0 < u, v \leqslant n-1$$

where $[\theta_1, \theta_2)$ is one of the sectors and $\mathbf{v}$ is defined by different values of θ_1 and θ_2. These features exploit the sensitivity of the power spectrum to the directionality of the texture. If a texture has as many lines or edges in a given direction θ, $|F|^2$ will tend to have high values clustered around the direction in frequency space $\theta + \pi/2$.

Texture Energy in the Spatial Domain

From Section 2.2.4 we know that the Fourier approach could also be carried out in the image domain. This is the approach taken in [Laws 1980]. The advantage of this approach is that the basis is not the Fourier basis but a variant that is more matched to intuition about texture features. Figure 6.19 shows the most important of Laws' 12 basis functions.

The image is first histogram-equalized (Section 3.2). Then 12 new images are made by convolving the original image with each of the basis functions (i.e., $f_k' = f * h_k$ for basis functions $h_1, \ldots, h_{12}$). Then each of these images is transformed into an "energy" image by the following transformation: Each pixel in the convolved image is replaced by an average of the absolute values in a local window of 15 × 15 pixels centered over the pixel:

$$f_k''(x, y) = \sum_{x',y' \text{ in window}} (|f_k'(x', y')|) \tag{6.7}$$

The transformation $f \rightarrow f_k''$, $k = 1, \ldots 12$ is termed a "texture energy transform" by Laws and is analogous to the Fourier power spectrum. The f_k'', $k = 1, \ldots 12$ form a set of features for each point in the image which are used in a nearest-neighbor classifier. Classification details may be found in [Laws 1980]. Our interest is in the particular choice of basis functions used.

Figure 6.20 shows a composite of natural textures [Brodatz 1966] used in Laws's experiments. Each texture is digitized into a 128 × 128 pixel subimage. The texture energy transforms were applied to this composite image and each pixel was classified into one of the eight categories. The average classification accuracy was about 87% for interior regions of the subimages. This is a very good result for textures that are similar.

6.4.2 Spatial Gray-Level Dependence

Spatial gray-level dependence (SGLD) matrices are one of the most popular sources of features [Kruger et al. 1974; Hall et al. 1971; Haralick et al. 1973]. The SGLD approach computes an intermediate matrix of measures from the digitized image data, and then defines features as functions on this intermediate matrix. Given an image f with a set of discrete gray levels I, we define for each of a set of discrete values of d and θ the intermediate matrix $S(d, \theta)$ as follows:

> $S(i, j|d, \theta)$, an entry in the matrix, is the number of times gray level i is oriented with respect to gray level j such that where
>
> $$f(\mathbf{x}) = i \quad \text{and} \quad f(\mathbf{y}) = j \quad \text{then}$$
> $$\mathbf{y} = \mathbf{x} + (d\cos\theta,\ d\sin\theta)$$

$$\begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 16 & -24 & 16 & -4 \\ 6 & -24 & 36 & -24 & 6 \\ -4 & 16 & -24 & 16 & -4 \\ 1 & -4 & 6 & -4 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 2 & 0 & -1 \\ -2 & 0 & 4 & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & -4 & 0 & 2 \\ 1 & 0 & -2 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \\ -4 & 0 & 8 & 0 & -4 \\ -6 & 0 & 12 & 0 & -6 \\ -4 & 0 & 8 & 0 & -4 \\ -1 & 0 & 2 & 0 & -1 \end{bmatrix}$$

Fig. 6.19 Laws' basis functions (these are the low-order four of twelve actually used).

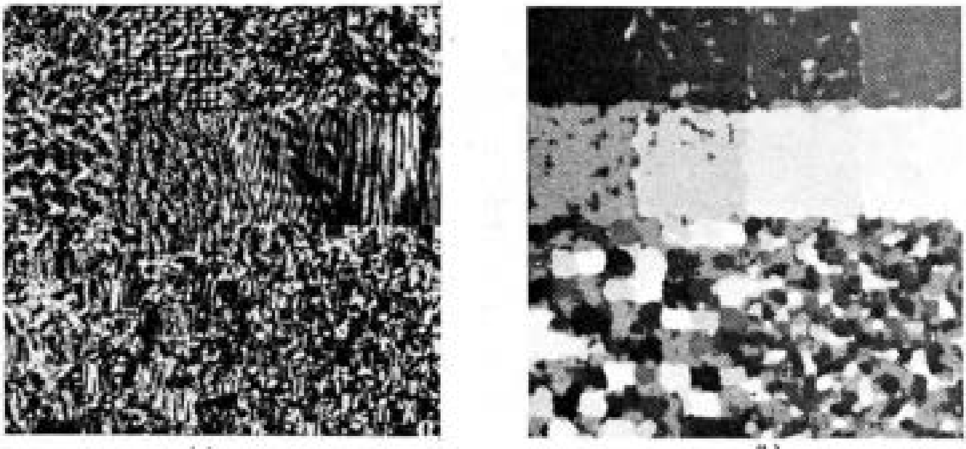


Fig. 6.20 (a) Texture composite. (b) Classification.

Note that we the gray-level values appear as indices of the matrix S, implying that they are taken from some well-ordered discrete set 0, ..., K. Since

$$S(d, \theta) = S(d, \theta + \pi).$$

common practice is to restrict θ to multiples of $\pi/4$. Furthermore, information is not usually retained at both θ and $\theta + \pi$. The reasoning for the latter step is that for most texture discrimination tasks, the information is redundant. Thus we define

$$S(d, \theta) = \tfrac{1}{2}\,[S(d, \theta) + S(d, \theta + \pi)]$$

The intermediate matrices S yield potential features. Commonly used features are:

1. *Energy*

$$E(d, \theta) = \sum_{i=0}^{K} \sum_{j=0}^{K} [S(i, j|d, \theta)]^2 \tag{6.8}$$

2. *Entropy*

$$H(d, \theta) = \sum_{i=0}^{K} \sum_{j=0}^{K} S(i, j|d, \theta) \log f(i, j|d, \theta) \tag{6.9}$$

3. *Correlation*

$$C(d, \theta) = \frac{\sum_{i=0}^{K} \sum_{j=0}^{K} (i - \mu_x)(j - \mu_y) S(i, j|d, \theta)}{\sigma_x \sigma_y} \tag{6.10}$$

4. *Inertia*

$$I(d, \theta) = \sum_{i=0}^{K} \sum_{j=0}^{K} (i - j)^2\, S(i, j|d, \theta) \tag{6.11}$$

5. *Local Homogeneity*

$$L(d,\theta) = \sum_{i=0}^{K} \sum_{j=0}^{K} \frac{1}{1+(i-j)^2} S(i,j|d,\theta) \tag{6.12}$$

where $S(i,j|d,\theta)$ is the (i,j) th element of (d,θ), and

$$\mu_x = \sum_{i=0}^{K} i \sum_{j=0}^{K} S(i,j|d,\theta) \tag{6.13a}$$

$$\mu_y = \sum_{j=0}^{K} j \sum_{j=0}^{K} S(i,j|d,\theta) \tag{6.13b}$$

$$\sigma_x^2 = \sum_{i=0}^{K} (i-\mu_x)^2 \sum_{j=0}^{K} f(i,j|d,\theta) \tag{6.13c}$$

and

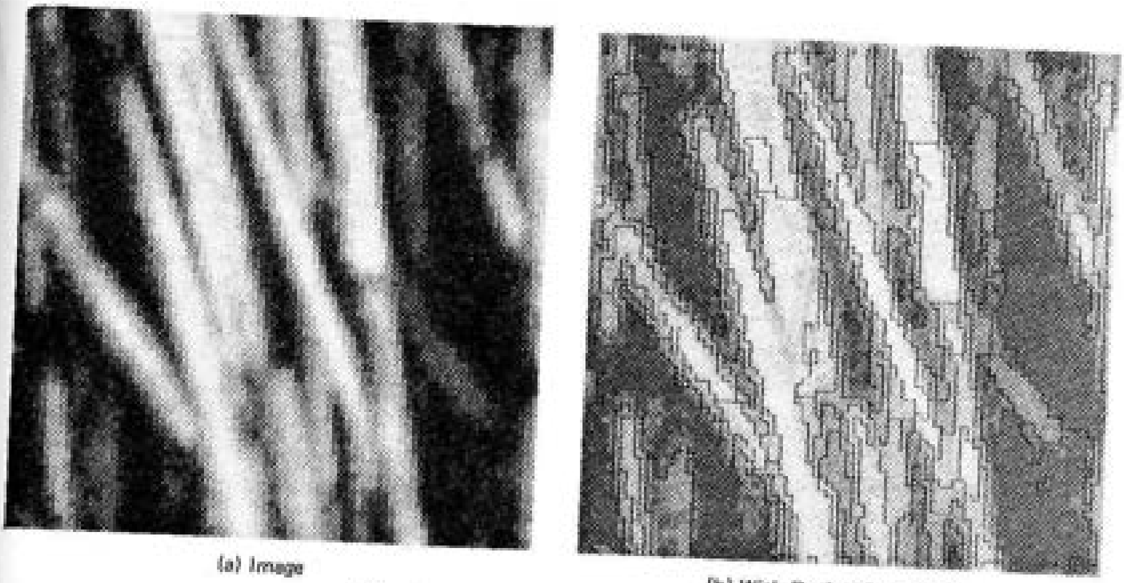
$$\sigma_y^2 = \sum_{j=0}^{K} (j-\mu_y)^2 \sum_{i=0}^{K} f(i,j|d,\theta) \tag{6.13d}$$

One important aspect of this approach is that the features chosen do not have psychological correlates [Tamura et al. 1978]. For example, none of the measures described would take on specific values corresponding to our notions of "rough" or "smooth." Also, the texture gradient is difficult to define in terms of SGLD feature values [Bajcsy and Lieberman 1976].

6.4.3 Region Texels

Region texels are an image-based way of defining primitives above the level of pixels. Rather than defining features directly as functions of pixels, a region segmentation of the image is created first. Features can then be defined in terms of the shape of the resultant regions, which are often more intuitive than the pixel-related features. Naturally, the approach of using edge elements is also possible. We shall discuss this in the context of texture gradients.

The idea of using regions as texture primitives was pursued in [Maleson et al. 1977]. In that implementation, all regions are ultimately modeled as ellipses and a corresponding five-parameter shape description is computed for each region. These parameters only define gross region shape, but the five-parameter primitives seem to work well for many domains. The texture image is segmented into regions in two steps. Initially, the modified version of Algorithm 5.1 that works for gray-level images is used. Figure 6.21 shows this example of the segmentation applied to a sample of "straw" texture. Next, parameters of the region grower are controlled so as to encourage convex regions which are fit with ellipses. Figure 6.22 shows the resultant ellipses for the "straw" texture. One set of ellipse parameters is $\mathbf{x}_0$, a, b, θ where $\mathbf{x}_0$ is the origin, a and b are the major and minor axis lengths and θ is the orientation of the major axis (Appendix 1). Besides these shape parameters, elliptical texels are also described by their average gray level. Figure 6.23 gives a qualitative indication of how ranges on feature values reflect different texels.



(a) Image

(b) With Region Boundaries

Fig. 6.21 Region segmentation for straw texture.

6.5 THE TEXTURE GRADIENT

The importance of texture in determining surface orientation was described by Gibson [Gibson 1950]. There are three ways in which this can be done. These methods are depicted in Fig. 6.24. All these methods assume that the texture is embedded on a planar surface.

First, if the texture image has been segmented into primitives, the maximum rate of change of the projected size of these primitives constrains the orientation of

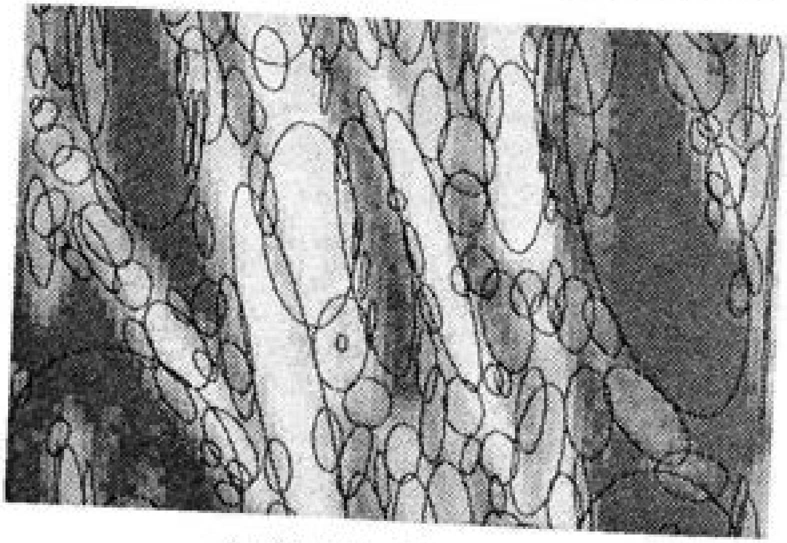


Fig. 6.22 Ellipses for straw texture.

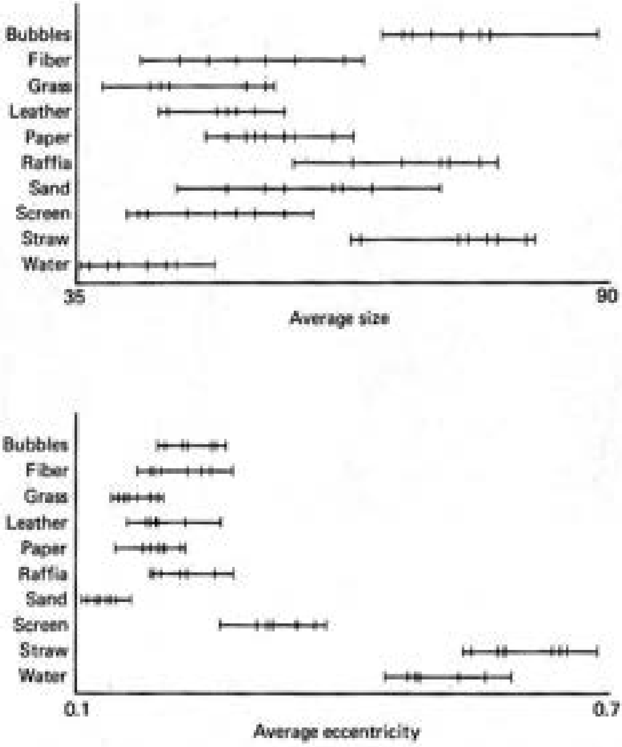


Fig. 6.23 Features defined on ellipses.

the plane in the following manner. The direction of maximum rate of change of projected primitive size is the direction of the *texture gradient*. The orientation of this direction with respect to the image coordinate frame determines how much the plane is rotated about the camera line of sight. The magnitude of the gradient can help determine how much the plane is tilted with respect to the camera, but knowledge about the camera geometry is also required. We have seen these ideas before in the form of gradient space; the rotation and tilt characterization is a polar coordinate representation of gradients.

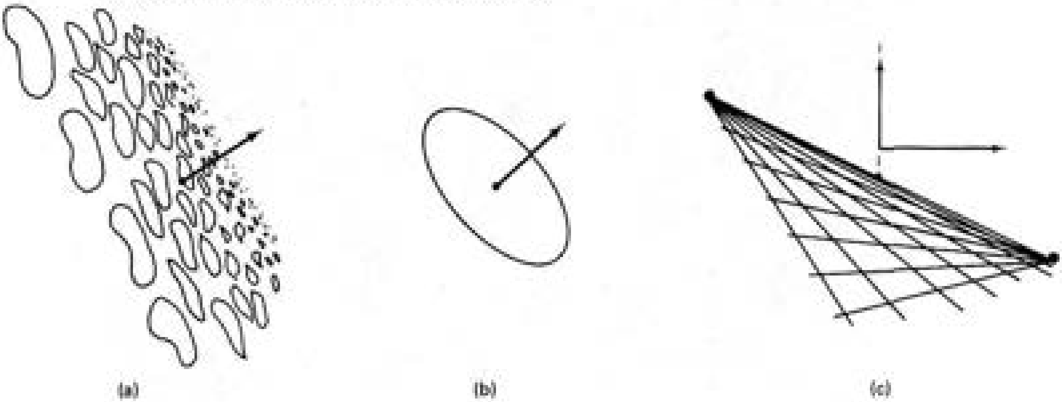


Fig. 6.24 Methods for calculating surface orientation from texture.

The second way to measure surface orientation is by knowing the shape of the texel itself. For example, a texture composed of circles appears as ellipses on the tilted surface. The orientation of the principal axes defines rotation with respect to the camera, and the ratio of minor to major axes defines tilt [Stevens 1979].

Finally, if the texture is composed of a regular grid of texels, we can compute vanishing points. For a perspective image, vanishing points on a plane P are the projection onto the image plane of the points at infinity in a given direction. In the examples here, the texels themselves are (conveniently) small line segments on a plane that are oriented in two orthogonal directions in the physical world. The general method applies whenever the placement tesselation defines lines of texels. Two vanishing points that arise from texels on the same surface can be used to determine orientation as follows. The line joining the vanishing points provides the orientation of the surface and the vertical position of the plane with respect to the z axis (i.e., the intersection of the line joining the vanishing points with $x = 0$) determines the tilt of the plane.

Line segment textures indicate vanishing points [Kender 1978]. As shown in Fig. 6.25, these segments could arise quite naturally from an urban image of the windows of a building which has been processed with an edge operator.

As discussed in Chapter 4, lines in images can be detected by detecting their parameters with a Hough algorithm. For example, by using the line parameterization

$$x \cos\theta + y \sin\theta = r$$

and by knowing the orientation of the line in terms of its gradient $\mathbf{g} = (\Delta x, \ \Delta y)$, a line segment $(x, y, \Delta x, \Delta y)$ can be mapped into r, θ space by using the relations

$$r = \frac{\Delta x x + \Delta y y}{\sqrt{\Delta x^2 + \Delta y^2}} \tag{6.14}$$

$$\theta = \tan^{-1} \left[\frac{\Delta y}{\Delta x}\right] \tag{6.15}$$

These relationships can be derived by using Fig. 6.26 and some geometry. The Cartesian coordinates of the $r-\theta$ space vector are given by

$$\mathbf{a} = \left[\frac{\mathbf{g}\cdot\mathbf{x}}{\|\mathbf{g}\|^2}\right]\mathbf{g} \tag{6.16}$$

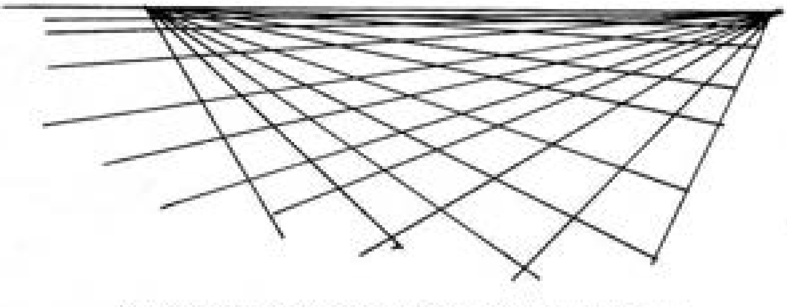


Fig. 6.25 Orthogonal line segments comprising a texture.

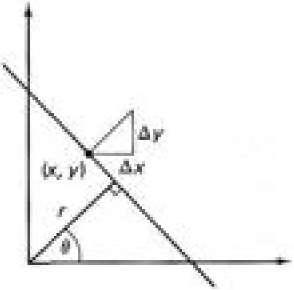


Fig. 6.26 r-θ transform.

Using this transformation, the set of line segments L_1 shown in Fig. 6.27 are all mapped into a single point in $r-\theta$ space. Furthermore, the set of lines L_2 which have the same vanishing point (x_v, y_v) project onto a circle in $r-\theta$ space with the line segment $((0, 0), (x_v, y_v))$ as a diameter. This scheme has two drawbacks: (1) vanishing points at infinity are projected into infinity, and (2) circles require some effort to detect. Hence we are motivated to use the transform $(x, y, \Delta x, \Delta y) \rightarrow \left(\frac{k}{r}, \theta\right)$ for some constant k. Now vanishing points at infinity are projected into the origin and the locus of the set of points L_2 is now a line. This line is perpendicular to the vector $\boldsymbol{x}_v$ and $\frac{k}{\|\mathbf{x}_v\|}$ units from the origin, as shown in Fig. 6.28. It can be detected by a second stage of the Hough transform; each point **a** is mapped into an $r'-\theta'$ space. For every **a**, compute all the r', θ' such that

$$a \cos\theta' + b \sin\theta' = r' \tag{6.17}$$

and increment that location in the appropriate r', θ' accumulator array. In this second space a vanishing point is detected as

$$r' = \frac{k}{\|\mathbf{x}_v\|} \tag{6.18}$$

$$\theta' = \tan^{-1}\left(\frac{y_v}{x_v}\right) \tag{6.19}$$

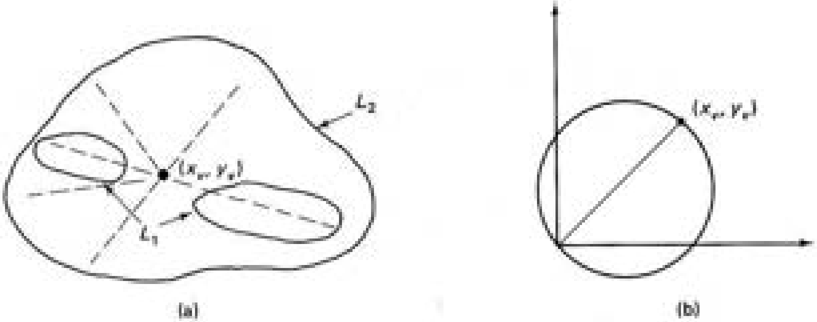


Fig. 6.27 Detecting the vanishing point with the Hough transform.

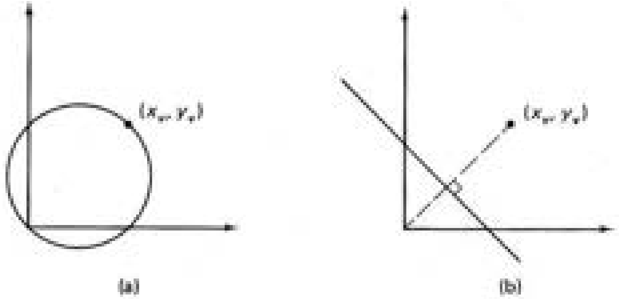


Fig. 6.28 Vanishing point loci.

In Kender's application the texels and their placement tesselation are similar in that the primitives are parallel to arcs in the placement tesselation graph. In a more general application the tesselation could be computed by connecting the centers of primitives.

EXERCISES

6.1 Devise a computer algorithm that, given a set of texels from each of a set of different "windows" of the textured image, checks to see of the resolution is appropriate. In other words, try to formalize the discussion of resolution in Section 6.2.

6.2 Are any of the grammars in Section 6.3 suitable for a parallel implementation (i.e., parallel application of rules)? Discuss, illustrating your arguments with examples or counterexamples from each of the three main grammatical types (shape, tree, and array grammars).

6.3 Are shape, array, and tree grammars context free or context-sensitive as defined? Can such grammars be translated into "traditional" (string) grammars? If not, how are they different; and if so, why are they useful?

6.4 Show how the generalized Hough transform (Section 4.3) could be applied to texel detection.

6.5 In an outdoors scene, there is the problem of different scales. For example, consider the grass. Grass that is close to an observer will appear "sharp" and composed of primitive elements, yet grass distant from an observer will be much more "fuzzy" and homogeneous. Describe how one might handle this problem.

6.6 The texture energy transform (Section 6.4.1) is equivalent to a set of Fourier-domain operations. How do the texture energy features compare with the ring and sector features?

6.7 The texture gradient is presumably a gradient in some aspect of texture. What aspect is it, and how might it be quantified so that texture descriptions can be made gradient independent?

6.8 Write a texture region grower and apply it to natural scenes.

REFERENCES

BAJCSY, R. and L. LIEBERMAN. "Texture gradient as a depth cue." *CGIP 5*, 1, March 1976, 52–67.

BRODATZ, P. *Textures: A Photographic Album for Artists and Designers.* Toronto: Dover Publishing Co., 1966.

Connors, R. "Towards a set of statistical features which measure visually perceivable qualities of textures." *Proc.*, PRIP, August 1979, 382–390.

Cover, T. M. "Estimation by the nearest neighbor rule." *IEEE Trans. Information Theory 14*, January 1968, 50–55.

Fu, K. S. *Sequential Methods in Pattern Recognition and Machine Learning.* New York: Academic Press, 1968.

Fu, K. S. *Syntactic Methods in Pattern Recognition.* New York: Academic Press, 1974.

Fukunaga, K. *Introduction to Statistical Pattern Recognition.* New York, Academic Press, 1972.

Gibson, J. J. *The Perception of the Visual World.* Cambridge, MA: Riverside Press, 1950.

Hall, E. L, R. P. Kruger, S. J. Dwyer III, D. L. Hall, R. W. McLaren, and G. S. Lodwick. "A survey of preprocessing and feature extraction techniques for radiographic images." *IEEE Trans. Computers 20*, September 1971.

Haralick, R. M. "Statistical and structural approaches to texture." *Proc.*, 4th IJCPR, November 1978, 45–60.

Haralick, R. M., R. Shanmugam, and I. Dinstein. "Textural features for image classification." *IEEE Trans. SMC 3*, November 1973, 610–621.

Hopcroft, J. E. and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Reading, MA: Addison-Wesley, 1979.

Jayaramamurthy, S. N. "Multilevel array grammars for generating texture scenes." *Proc.*, PRIP, August 1979, 391–398.

Julesz, B. "Textons, the elements of texture perception, and their interactions." *Nature 290*, March 1981, 91–97.

Kender, J. R. "Shape from texture: a brief overview and a new aggregation transform." *Proc.*, DARPA IU Workshop, November 1978, 79–84.

Kruger, R. P., W. B. Thompson, and A. F. Twiner. "Computer diagnosis of pneumoconiosis." *IEEE Trans. SMC 45*, 1974, 40–49.

Laws, K. I. "Textured image segmentation." Ph.D. dissertation, Dept. of Engineering, Univ. Southern California, 1980.

Lu, S. Y. and K. S. Fu. "A syntactic approach to texture analysis." *CGIP 7*, 3, June 1978, 303–330.

Maleson, J. T., C. M. Brown, and J. A. Feldman. "Understanding natural texture." *Proc.*, DARPA IU Workshop, October 1977, 19–27.

Milgram, D. L. and A. Rosenfeld. "Array automata and array grammars." *Proc.*, IFIP Congress 71, Booklet TA-2. Amsterdam: North-Holland, 1971, 166–173.

Pratt, W. K., O. D. Faugeras, and A. Gagalowicz. "Applications of Stochastic Texture Field Models to Image Processing." *Proc. of the IEEE*, Vol.69, No. 5, May 1981

Rosenfeld, A. "Isotonic grammars, parallel grammars and picture grammars." In *MI6*, 1971.

Stevens, K.A. "Representing and analyzing surface orientation." In *Artificial Intelligence: An MIT Perspective*, Vol. 2, P. H. Winston and R. H. Brown (Eds.). Cambridge, MA: MIT Press, 1979.

Stiny, G. and J. Gips. *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts.* Berkeley, CA: University of California Press, 1972.

Tamura, H., S. Mori, and T. Yamawaki. "Textural features corresponding to visual perception." *IEEE Trans. SMC 8*, 1978, 460–473.

Tou, J. T. and R. C. Gonzalez. *Pattern Recognition Principles.* Reading, MA: Addison-Wesley, 1974.

Weszka, J. S., C. R. Dyer, and A. Rosenfeld. "A comparative study of texture measures for terrain classification." *IEEE Trans. SMC 6*, 4, April 1976, 269–285.

Zucker, S. W. "Toward a model of texture." *CGIP 5*, 2, June 1976, 190–202.

Motion 7

7.1 MOTION UNDERSTANDING

Motion imagery presents many interesting challenges to computer vision, but static scene analysis received more attention in the 1960's and 1970's. In part, this may have been due to a technical problem: With most types of input media and domains, motion vision input is much more voluminous than static vision input. However, we believe that a more basic problem has been the assumption that motion vision could best be understood (or implemented) as many static frames analyzed very quickly, with results linked up in temporal sequence. This characterization of motion vision is extreme but perhaps illuminating. First, it assumes that vision involves processing static scenes. Second, it acknowledges that massive amounts of data may be required. Third, in it motion understanding degenerates to a postprocessing step which is mostly a matching operation—the differences or similarities between (understood) frames are analyzed and recorded. The extreme "static is basic" view is that motion is an unnaturally complex or difficult problem because it is ill suited to the techniques available.

A modified view is that object motion provides good image cues for segmentation, much as color might. This approach leads to the use of motion for segmentation, so that motion gets a more basic role in the understanding process. In this view, motion as such is useful for basic image understanding; a motion image sequence may actually be easier to understand than a static image, because the effects of motion can help in segmentation. Recent examples may be found in [Snyder 1981].

A further departure from the "static is basic" view is that motion understanding is qualitatively different from static vision. A logical extreme of this view is that there are many visual processing operations whose primitives are points in motion, and that in fact static vision is the puzzle, being ill-suited to the needs and mechanisms of biological systems. Serious work in computer motion understanding has begun even more recently than computer vision as a whole, and it is too early to dismiss any approach out of hand. There are domains and applications in which the "static is basic" paradigm seems natural, but it also seems very reasonable that animals have perceptual systems or subsystems for which "motion is basic."

Section 7.2 is concerned with processing and understanding the "flow" of the world image across the retina. Section 7.3 considers several techniques for understanding sequences of static images.

7.1.1 Domain Independent Understanding

Domain independent motion processing extracts information from time-varying images using the weakest possible assumptions about the world. Processing that merely transforms the input data into another image-like structure is in the province of generalized image processing. However, if the motion processing aggregates spatial information on the basis of a common feature, then the processing is a form of segmentation.

The basic visual input for domain-independent work in motion vision understanding is *optical flow.* Although Helmholtz noted the striking immediacy of three-dimensional perception mediated through motion [Helmholtz 1925], Gibson is usually credited with pioneering the theory that a primary visual stimulus for motion is the flow of elements in the optic array, or pattern of luminance in the full sphere of solid angle surrounding the observer [Gibson 1950, 1957, 1965, 1966]. Human beings undoubtedly are sensitive to optical flow, as evidenced by the "looming" reflex [Schiff 1965], the effect of flow on balance [Lee and Lishman 1975], and many other documented phenomena [Nakayama and Loomis 1974]. The basic input to an "optical flow understander" is a continuously changing visual field, which may be considered a field of vectors, each expressing the instantaneous change of position on the optic array of the image of a world point. A field of such vectors is shown in Fig. 7.1. The extraction of the vectors from the changing image is a low-level operation often posited by optical flow research; one computational mechanism was given in Chapter 3. Flow may also be approximated in an image sequence by matching and difference operations (Section 7.3.1).

Computer vision researchers have recently begun to concern themselves with both the geometry and computational mechanisms that might be useful in the understanding of optical flow [Horn and Schunck 1980; Clocksin 1980; Prager 1979; Prazdny 1979; Lawton 1981]. Many formalisms are in use. Cartesian, polar space, and spherical coordinates all have their appeal in different situations; differential vector geometry and simple analytic geometry are both used; even the geometry of the eye or camera varies from one study to another. This chapter does not contain a "unified flow theory;" instead it briefly describes several approaches, each of which uses a different aspect of optical flow.

7.1.2 Domain Dependent Understanding

The use of models, or at least stronger assumptions about the world, is complementary to domain-independent processing. The changing image, or even the field of optical flow, can be treated as input to a model-driven vision process whose goal

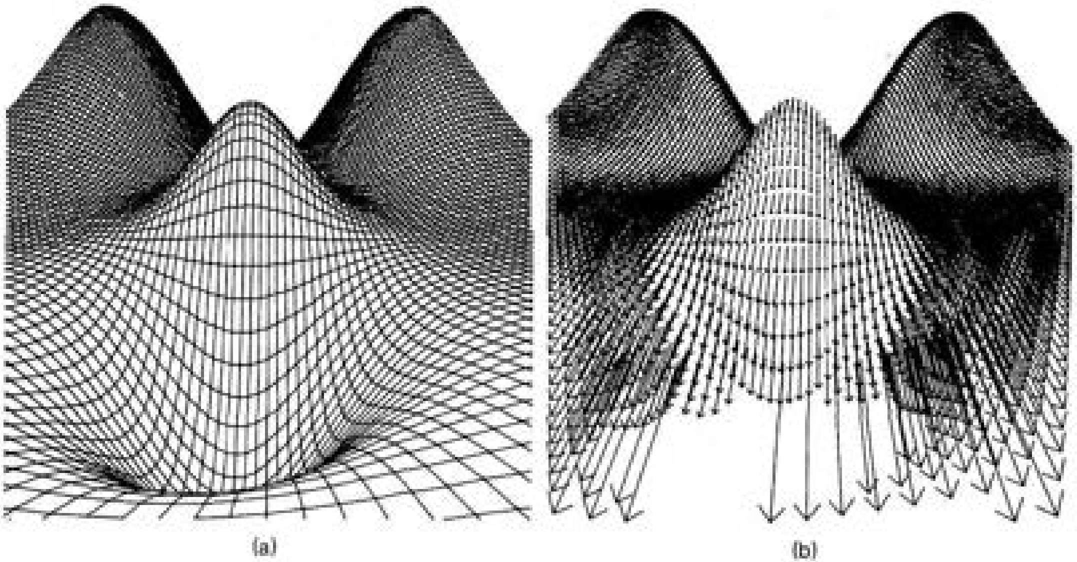


Fig. 7.1 An example of an optical flow field for an approaching "hill." (a) The hill. (b) Flow field.

is typically to segment the input into areas corresponding to meaningful world objects. The optical flow field becomes just another component of the generalized image, together with intensity, texture, or color. Motion often reveals information similar to that from range data; flow and range are discontinuous at object boundaries, surface orientation may be derived, and so forth. Object (or world) motions determine image (or retinal) motions; we shall be explicit about which motion we mean when confusion can occur.

Section 7.3 describes how knowledge of object motion phenomena can help in segmenting the flow field. One useful assumption is that the world contains rigid bodies. Tests for rigid bodies and calculations using data from them are quite useful—for example, the three-dimensional position of four points on a rigid object may be determined uniquely from three views (Section 7.3.2). A weaker object model, that they are assemblies of compound rigid pendula (linkages), is enough to accomplish successful segmentation of very sparse motion input which consists only of images of the end points of links (Section 7.3.3). Section 7.3.4 describes work with a highly specific and detailed model which is used in several ways to restrict low-level image processing and aid in three-dimensional interpretation of human motion images. Section 7.3.5 considers the processing of sequences of segmented images.

The coherence of most three-dimensional objects and their continuity through time are two general principles which, although occasionally violated, guide many segmentation and point-matching heuristics. The assumed correspondence of regions in images with objects is one example. Motion images provide another example; object coherence implies the likelihood of many "continuity" (actually similarity) conditions on the positions and velocities of neighboring image points.

Here are five heuristics for use in matching points from images separated by a small time interval [Prager 1979] (Fig. 7.2).

1. *Maximum velocity.* If a world point is known to have a maximum velocity V with respect to a stationary imaging device, then it can move at most $V\ dt$ between two images made dt time units apart. Thus given the location of the point in one image (and some assumptions about depth), this constraint limits where the point can appear on the second image.
2. *Small velocity change.* Since most visible physical objects have finite mass, this heuristic is a conseqence of physical laws and the assumption of a "small interval" between images. Of course, the definition of "small interval" depends on the definition of the velocitv changes one desires to measure.

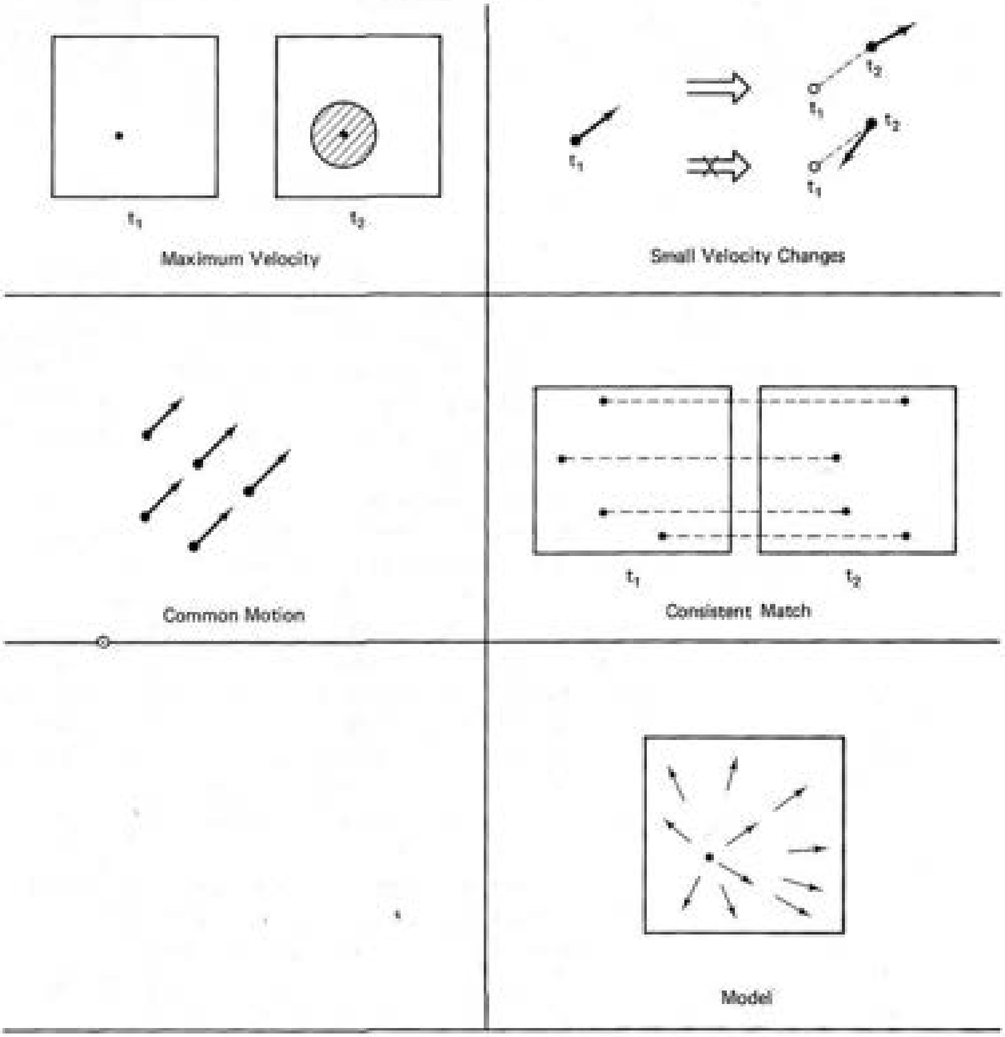


Fig. 7.2 Five heuristics.

3. *Common motion.* Spatially coherent objects often appear in successive images as regions of points sharing a "common motion." It is interesting that such a weak notion as common motion (and the related "common position") actually can serve to segment very sparse scenes of a few points with very complex motion behavior if a long-enough sequence of images is used (Sections 7.3.3 and 7.3.4).
4. *Consistent match.* Two points from one image generally do not match a single point from another image (exceptions arise from occlusions). This is one of the main heuristics in the stereopsis algorithm described in Chapter 3.
5. *Known motion.* If a world model can supply information about object motions, perhaps retinal motions can be derived, predicted, and recognized.

In the discussions to follow these heuristics (and others) are often used or implicitly taken as principles. A careful catalog of the probable behavior of objects in motion is often a useful practical adjunct to a mathematical treatment. The mathematics itself must be based on a set of assumptions, and often these are closely related to the phenomenological heuristics noted above.

7.2 UNDERSTANDING OPTICAL FLOW

This section describes some more direct calculations on optical flow, using no other input information. Information may be obtained from flow that seems useful both for survival in the world and (on a less existential level) for automated image understanding. As with shape from shading research (Chapter 3), the paradigm here is often to see mathematically what information resides in the input and to use this to suggest mechanisms for doing the computation. The flow input is assumed to be known (Chapter 3 showed how to derive optical flow by local analysis of changing intensity in the image).

7.2.1 Focus of Expansion

As one moves through a world of static objects, the visual world as projected on the retina seems to flow past. In fact, for a given direction of translatory motion and direction of gaze, the world seems to be flowing out of one particular retinal point, the *focus of expansion* (FOE). Each direction of motion and gaze induces a unique FOE, which may be a point at infinity if the motion is parallel to the retinal (image) plane.

These aspects of optical flow have been studied by computing the simulated flow pattern an observer would see while moving through a "forest" of vertical cylinders [Prager 1979] or Gaussian hills and valleys [Lawton 1981]. Some sample FOEs are shown in Fig. 7.3. Figure 7.3c shows a second FOE when the field of view contains an object which is itself in motion.

Our first model of the imaging situation is a simplification of the imaging geometry given in Appendix 1. Let the viewpoint be at the origin with the view

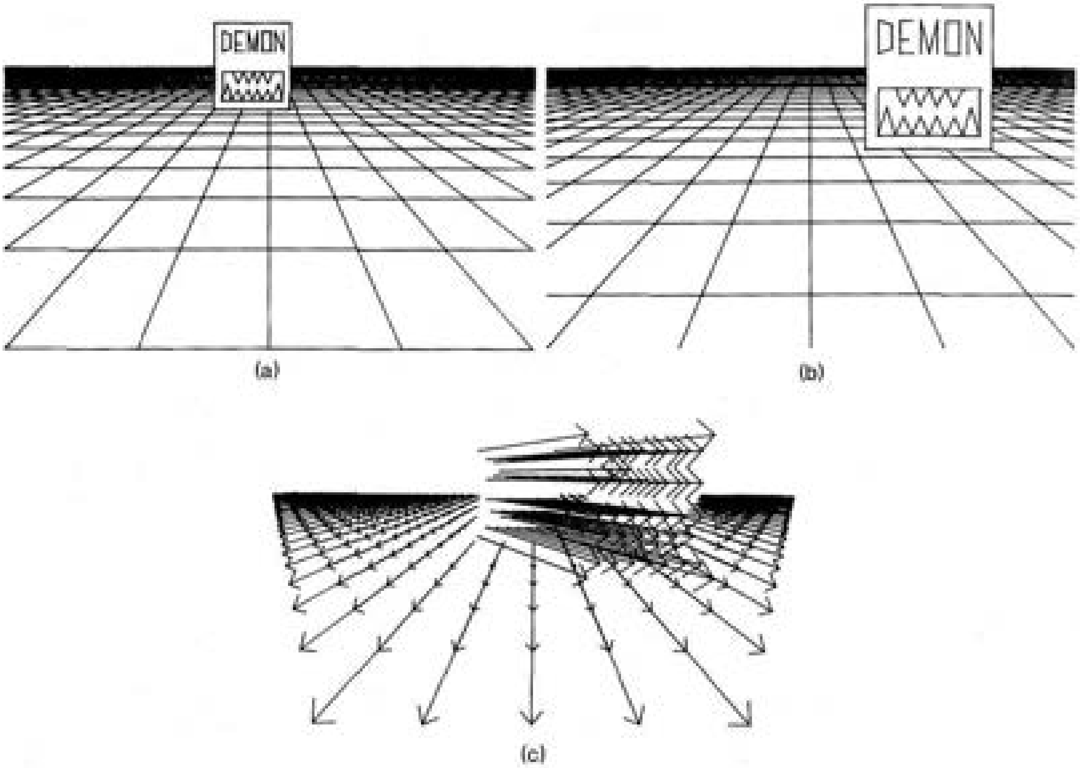


Fig. 7.3 FOE for rectilinear observer motion. (a) An image. (b) Later image. (c) Flow shows different FOEs for static floor and moving object.

direction out along the positive Z axis, and let the focal length $f = 1$. Then the perspective distortion equations simplify to

$$x' = \frac{x}{z} \tag{7.1}$$

$$y' = \frac{y}{z} \tag{7.2}$$

In the next two sections the letters u, v, and w (sometimes written as functions of t) denote world point velocity components, or the time derivatives of world coordinates (x, y, z). Observer motion with instantaneous velocity $(-dx/dt, -dy/dt, -dz/dt) = (-u, -v, -w)$, keeping the coordinate system attached to the viewpoint, gives points in a stationary world a relative velocity (u, v, w). Consider a point located at (x_0, y_0, z_0) at some initial time. After a time interval t, its image will be at

$$(x', y') = \left(\frac{x_0 + ut}{z_0 + wt}, \frac{y_0 + vt}{z_0 + wt}\right) \tag{7.3}$$

As t varies, this parametric "flow-path" equation is that of a straight line; as t goes to minus infinity, the image of the point travels back along the straight line toward a particular point on the image, namely,

$$\text{FOE} = \left[\frac{u}{w}, \frac{v}{w}\right] \tag{7.4}$$

This focus of expansion is where the optical flow originates on the image. If the observer changes direction (or objects in the world change their direction), the FOE changes as well.

7.2.2 Adjacency, Depth, and Collision

The flow path equation of a point moving with a constant velocity reveals information about its depth in z. The information is not provided directly, since all flow paths for points at a given depth do not look alike. However, there is the elegant relation

$$\frac{D(t)}{V(t)} = \frac{z(t)}{w(t)} \tag{7.5}$$

Here again w is dz/dt, and V is dD/dt. D is the distance along the straight flow path from the FOE to the image of the point. Thus the distance/velocity ratio of the point's image is the same as the distance/velocity ratio of the world point. This result is basic, but perhaps not immediately obvious.

The above relation is called the time-to-adjacency relation, because the right-hand side, z/w, is the z-distance of the point from the image plane divided by its velocity toward the plane. It is thus the time until the point passes through the image plane. This basic time interval is clearly useful when dealing with world objects; it changes when the magnitude of the world point's velocity (or the observer's) changes.

Knowing the depth of any point determines the depth of all others of the same velocity w, for it follows from the two time to adjacency equations of the points that

$$z_2(t) = \frac{z_1(t) D_2(t) V_1(t)}{V_2(t) D_1(t)} \tag{7.6}$$

The time-to-adjacency equation allows easy determination of the world coordinates of a point, scaled by its z velocity. If the observer is mobile and in control of his own velocity, and if the world is stationary, such scaled coordinates may be useful. Using the perspective distortion equations,

$$z(t) = \frac{w(t) D(t)}{V(t)} \tag{7.7}$$

$$y(t) = \frac{y'(t) w(t) D(t)}{V(t)} \tag{7.8}$$

$$x(t) = \frac{x'(t) w(t) D(t)}{V(t)} \tag{7.9}$$

As a last example, let us relate optical flow to the sensing of impending collisions with world objects. The focal point of the imaging system, or origin of coordinates, is at any instant headed "toward the focus of expansion," whose image coordinates are $(u/w, v/w)$. It is thus traveling in the direction

$$\mathbf{O} = (\frac{u}{w}, \frac{v}{w}, 1) \tag{7.10}$$

and is following at any instant a path in the environment instantaneously defined by the parametric equation

$$(x, y, z) = t\mathbf{O} = t(\frac{u}{w}, \frac{v}{w}, 1) \tag{7.11}$$

where t acts like a real scalar measure of time. Given this vector expression for the path of the observer, one can apply well-known vector formulas from analytic solid geometry to derive useful information about the relation of this path to world points, which are also vectors.

For example, the position $\mathbf{P}$ along the observer's path at which a world point approaches closest is given by

$$\mathbf{P} = \frac{\mathbf{O}(\mathbf{O} \cdot \mathbf{x})}{(\mathbf{O} \cdot \mathbf{O})} \tag{7.12}$$

where $\mathbf{O}$ is the direction of observer motion and $\mathbf{x}$ the position of the world point. Here the period (.) is the dot product operator. The squared distance Q^2 between the observer and the world point at closest approach is then

$$Q^2 = (\mathbf{x} \cdot \mathbf{x}) - (\mathbf{x} \cdot \mathbf{O})^2 / (\mathbf{O} \cdot \mathbf{O}) \tag{7.13}$$

7.2.3 Surface Orientation and Edge Detection

It is possible to derive surface orientation and to characterize certain types of surface discontinuities (edges) by their motion. A formalism, computer program, and biologically motivated computational mechanism for these calculations was developed in [Clocksin 1980].

This section outlines mainly the surface orientation aspect of this work. As usual, the model is for a monocular observer, whose focal point is the origin of coordinates. An unusual feature of the model is that the observer has a spherical retina. The world is thus projected onto an "image unit sphere" instead of an image plane. World points and surface orientation are represented in an observer-centered Cartesian coordinate system. The image sphere has a spherical coordinate system which may be considered as "longitude" θ and "latitude" ϕ. These coordinates bear no relation to the orientation of the retina. World points are then determined by their image coordinates and a range r. An observer-centered Cartesian coordinate system is also useful; it is related to the sphere as shown in Fig. 7.4, and by the transformations given in Appendix 1.

The flow of the image of a freely moving world point may be found through the following derivation. As before, let the world velocity of the point (possibly induced by observer motion) $(dx/dt, dy/dt, dz/dt)$ be written (u, v, w). Similarly,

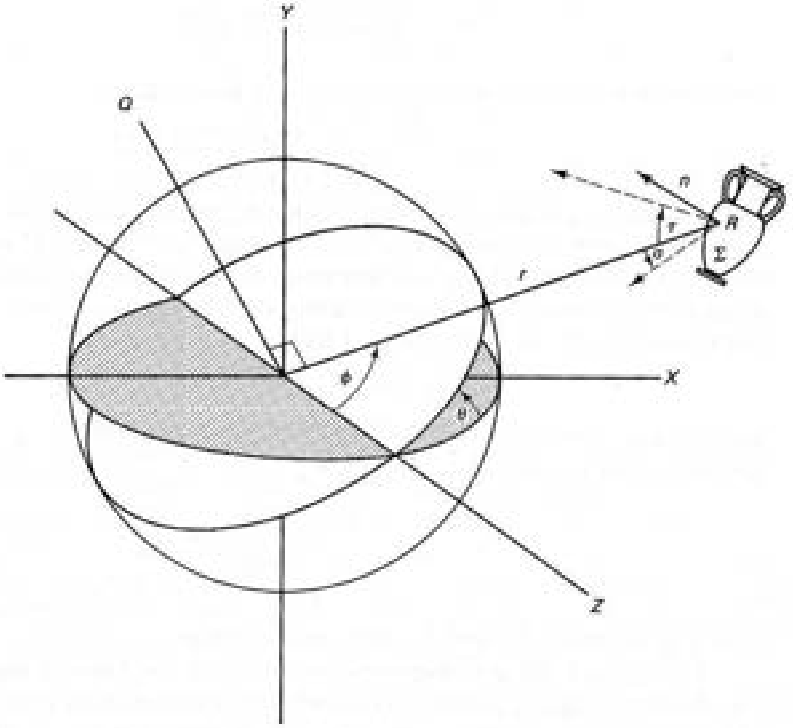


Fig. 7.4 Spherical coordinate system, and the definition of σ and τ.

write the angular velocities of the image point in the θ and ϕ directions as

$$\delta = \frac{d\theta}{dt} \tag{7.14}$$

$$\epsilon = \frac{d\phi}{dt} \tag{7.15}$$

Then from the coordinate transformation equations of Appendix 1,

$$y = x \tan\theta \tag{7.16}$$

Differentiating and solving for $d\theta/dt$ (written as δ) gives

$$\delta = \frac{v - u \tan\theta}{x \sec^2\theta} \tag{7.17}$$

Substituting for x its spherical coordinate expression $r \sin\phi \cos\theta$ and simplifying yields the general expression for flow in the θ direction:

$$\delta = \frac{v \cos\theta - u \sin\theta}{r \sin\phi} \tag{7.18}$$

The derivation of ϵ proceeds from the coordinate transformation equation

$$z = r \cos\phi \tag{7.19}$$

Differentiating, solving for $d\phi/dt$ (written as ϵ), and using

$$\frac{dr}{dt} = \frac{xu + yv + zw}{r} \tag{7.20}$$

yields the general expression for flow in the ϕ direction:

$$\epsilon = \frac{(xu + yv + zw)\cos\phi - rw}{r^2 \sin\phi} \tag{7.21}$$

As usual, general point motions are rather complicated to deal with, and more constraints are needed if the optic flow is to be "inverted" to discover much about the outside world. Let us then make the simplification that the world is stationary and the observer is traveling along the z direction at some speed S (This assumption is briefly discussed below.) Explicitly, suppose that

$$u = 0, \quad v = 0, \quad w = -S$$

Substituting these into the general flow equations (7.18) and (7.21) yields simplified flow equations:

$$\delta = 0 \tag{7.22}$$

$$\epsilon = \frac{S \sin\phi}{r} \tag{7.23}$$

Thus r is a function of θ and ϕ and therefore so is ϵ.

It is this simplified flow equation which forms the basis for surface orientation calculation and edge detection. The goals are to assign to any point in the flow field one of three interpretations: *edge*, *surface*, or *space* and also to derive the type of edge and the orientation of the surface.

To find surface orientation, represent the surface normal of a surface Σ by two angles σ and τ defined as in Fig. 7.4 with the two planes of σ and τ being the RZ and QR planes, respectively. The slant is measured relative to the line of sight, denoted by R in the figure. σ and τ correspond to depth changes in "depth profiles" oriented along lines of constant θ and ϕ, respectively. Thus,

$$\tan\sigma = \left(\frac{1}{r}\right)\frac{\partial r}{\partial\phi} \tag{7.24}$$

$$\tan\tau = \left(\frac{1}{r}\right)\frac{\partial r}{\partial\theta} \tag{7.25}$$

Surface orientation is defined by σ and τ or equivalently by their tangents. A surface perpendicular to the line of sight has $\sigma = \tau = 0$.

Equations (7.24) and (7.25) assume the range r is known. However, one can determine them without knowing r through the simplified flow equation, Eq. (7.23). The latter may be written

$$r = \frac{S \sin\phi}{\epsilon(\theta,\ \phi)}$$

where $\epsilon(\theta, \phi)$ gives the flow in the ϕ direction. Differentiating this with respect to θ and ϕ gives

$$\frac{\partial r}{\partial \phi} = S\,\frac{\epsilon \cos \phi - \sin \phi \,(\partial \epsilon/\partial \phi)}{\epsilon^2} \tag{7.26}$$

$$\frac{\partial r}{\partial \theta} = -\,\frac{S \sin \phi \,(\partial \epsilon/\partial \theta)}{\epsilon^2} \tag{7.27}$$

These last three equations may be substituted into Eqs. (7.24) and (7.25), and the results may then be simplified to the following surface orientation equations:

$$\tan \sigma = \cot \phi - \frac{\partial}{\partial \phi} \ln \epsilon \tag{7.28}$$

$$\tan \tau = -\frac{\partial}{\partial \theta} (\ln \epsilon) \tag{7.29}$$

These tangents are thus easily computed from optical flow. The result does not depend on velocity, and no depth scaling is required. In fact, absolute depth is not computable unless we know more, such as the observer speed.

Turning briefly to edge perception: Although physical edges are a depth phenomenon, in flow they are mirrored by ϵ, the flow measure that allows determination of orientation without depth. In particular, it is possible to demonstrate that the Laplacian of ϵ has singularities where the Laplacian of depth has singularities. An arc on the sphere projects out onto a "depth profile" in the world, along which depth may vary. If the arc is parameterized by α, relations among the depth profile, flow profile, and the singularities in flow are shown in Fig. 7.5. Thus the Laplacian of ϵ provides information about edge type but not about edge depth.

The formal derivations are at an end. Implementing them in a computer program or in a biological system requires solutions to several technical problems. More details on the implementation of this model on a computer and a possible

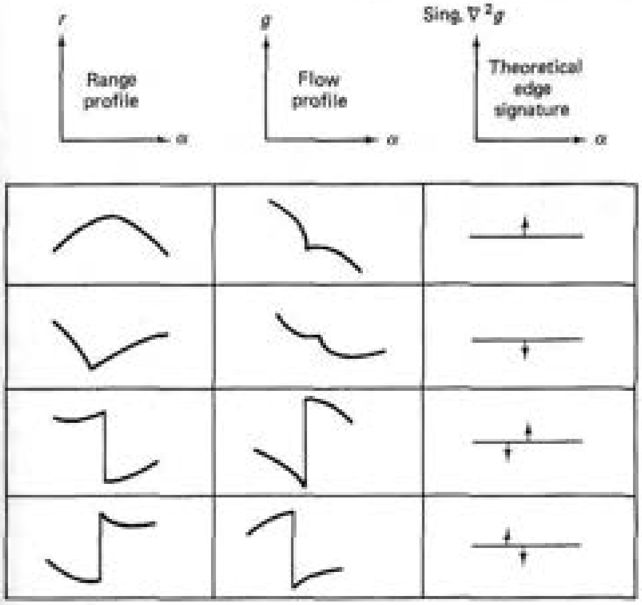


Fig. 7.5 The singularities of the second derivative of the flow profile inform about the type of edge.

implementation using low-level physiological vision primitives appear in [Clocksin 1980]. There are some data on human performance for the types of tasks attempted by the program. The assumption of a fixed environment basically implies that flow motions in the environment are likely to be interpreted as observer motions. This view is rather strikingly borne out by "swaying room" experiments [Lee and Lishman 1975], in which a subject stands in a swayable visual environment. (A large, low-mass bottomless box suspended from above may be lowered around the subject, giving him a room-like visual environment.) When the hanging "room" is made to sway, the subject inside tends to lose balance. Further, moving surfaces in the real world are quite often objects of interest, such as animals.

A survey of depth perception experiments [Braunstein 1976] points to motion as the dominant indicator of surface orientation perception. Random-dot displays of monocular flow patterns [Rogers and Graham 1979] evoke striking perceptions of solid oriented surfaces; flow may be adequate for shape and depth perception even with no other depth information. The experiments on perception of "edges," or discontinuities in flow caused by discontinuities in depth of textured surfaces, are less common. However, there have been enough to provide some confirmation of the model.

The computational model is consistent with and has correctly predicted psychological data on human thresholds for slant and edge perception in optical flow fields. (The thresholds are on the amount of slant to the surface and the depth difference of the edge sides.) The computational model can be used to determine range, but only to poor accuracy; this happens to correspond with the human trait that orientation is much more accurately determined by flow than is range. Quantitatively, the accuracy of orientation and range determinations are the same for the model and for human beings under similar conditions.

7.2.4 Egomotion

It is possible to extract information about complex observer motions from optical flow, although at considerable computational cost. In one formulation [Prazdny 1979], a model observer is allowed to follow any space curve in an environment of stationary objects, while at the same time turning its head. It is possible to derive formulae that determine the observer's instantaneous velocity vector and head rotational vector from a small number (six) of flow vectors in the image on a (standard flat) retina.

The equations that describe flow given observer motion and head rotation can be quite compactly written by using vector operators and a polar coordinate system (similar to that of the last section). The inherent elegance and power of the vector operations is well displayed in these calculations. Inverting the equations results in a system of three cubic equations of 20 terms each. Such a system can be solved by normal methods for simultaneous nonlinear equations, but the solutions tend to be relatively sensitive to noise. In the noise-free case, the method seems to perform quite adequately.

The calculation yields a method for deriving relative depth, or the ratio of the distances of points from the observer. An approximation to surface orientation may be obtained using several relative depth measurements in a small area and assuming that the surface normal varies slowly in tne area.

7.3 UNDERSTANDING IMAGE SEQUENCES

An image sequence is an ordered set of images. The image sequences of interest here are samplings of four-dimensional space-time. Commonly, as in a movie, the images are two-dimensional projections of a three-dimensional physical world, sequenced through time. Sometimes the sequence consists of two-dimensional images of essentially two-dimensional slices of the three-dimensional world, sequenced through the third spatial dimension. Some of the techniques in this section are useful in interpreting the three-dimensional nature of objects from such spatial image sequences, but the main concern here is with temporal image sequences. In many practical applications, the input must be such a sequence, and continuous motion must be inferred from discrete location differences of image points. The thrust of work under these assumptions is often to extend static image understanding by making models that incorporate or explain objects in motion, extending segmentation to work across time [Thompson 1979, Tsotsos 1980].

When asked why he was listening to a metronome ticking, Ezra Pound is said to have replied that he did not listen to the ticks, but to the "spaces between them." Like Pound, we take the ticks, or images, as given, and are really interested in what goes on "between the ticks." We usually want to determine and describe how the images are related to each other. This information must be derived from the static images, and two approaches immediately present themselves: broadly, the first is to look for differences between the images, and the second is to look for similarities.

These two approaches are complementary, and are often used together. A general paradigm for object-oriented motion analysis is the following:

1. Segment (describe) the individual images. This process may be complex, yielding a relational structure or a segmentation into regions or edges. An important special case is the one in which the description (segmentation) process is null and the description is just the image itself. For example, an initial high-level static description is impossible if motion is to be used as an aid to segmentation.
2. Compute and describe the differences or similarities between the descriptions (or undescribed images).
3. Build a description of the sequence as a whole from the single-frame primitives and descriptions of difference or similarity that are relevant to the purpose at hand.

7.3.1 Calculating Flow from Discrete Images

This method is a form of disparity calculation that is not only used for flow calculations, but may also be used for stereo matching or tracking applications. The computations are implemented with "relaxation" techniques.

The flow calculations have so far assumed an underlying continuous image which was densely sampled. With those assumptions and a few more the fundamental motion equation allows the calculation of flow (Chapter 3). The approach of this section is to identify discrete points in the image that are very different from their surround. Given such discrete points from each of two images at different times, the problem becomes one of matching a point in one image with the right point (if it exists) in the other image. This matching problem is known as the *correspondence* problem [Duda and Hart 1973, Aggarwal et al 1981]. The solution to the correspondence problem in the case of motion is, of course, the optic flow.

One algorithm for matching distinct points from two different frames [Barnard and Thompson 1979] breaks the matching problem into two steps. The first is the identification of candidate match points in each of the two frames. The second is an iterative algorithm which adjusts match probabilities for pairs of match points. After successful termination of the algorithm, correct matches have high probabilities and incorrect matches have very low probabilities.

The Moravec interest operator ([Moravec 1977]; Section 3.2) produces candidate match points by measuring the distinctness of a local piece of the image from its surround. Each frame is analyzed separately so that the end result is two sets of points S_1 and S_2, one from each frame, which are candidates to be matched. Candidates in S_1 are indexed by i and those in S_2 by j.

The iterative part of the algorithm is initialized with a data structure for the possible matches that exploits the heuristic that a point in the world does not move large distances between frames. Potential matches for a given point $\mathbf{x}_i$ in S_1, the first image, are all points $\mathbf{y}_j$ in S_2 such that

$$\|\boldsymbol{x}_i - \boldsymbol{y}_j\| \leqslant v_{\max} \tag{7.30}$$

where $v_{\max}$ is the maximum disparity allowed between points. All points that are selected by the Moravec operator have a given disparity vector $\mathbf{v}_{ij}$ and are kept as possible matches. Each disparity has an associated probability P_{ij} which changes through time as the most likely disparities are found. The information kept for each point $\mathbf{x}_i$ in S_1 looks like

$$(\mathbf{x}_i\ (\mathbf{v}_{ij_1},\ P_{ij_1})(\mathbf{v}_{ij_2},\ P_{ij_2}) \cdots (V^*,\ P^*)) \tag{7.31}$$

where V^* is a special symbol that denotes "no match," and all the j_k are members of S_2. Storing the flow vectors $\mathbf{v}$ implicitly stores the corresponding point in S_2 since $\mathbf{y}_j = \mathbf{x}_i + \mathbf{v}_{ij}$. Since the probabilities are adjusted iteratively, one final index is needed to denote the iteration value so that P_{ij} actually becomes P_{ij}^n for $n \geqslant 0$.

The initial approximation for the probabilities P_{ij}^0 takes advantage of the "common motion" heuristic: If $\mathbf{y}_j$ is the correct match point for $\mathbf{x}_i$, the image near $\mathbf{y}_j$ should look like the image near $\mathbf{x}_i$. Thus P_{ij}^0 can be defined by

$$P_{ij}^0 = \frac{1}{1 + cw_{ij}} \qquad \text{for } x \text{ in } S_1 \tag{7.32}$$

where

$$w_{ij} = \sum_{|d\mathbf{x}| \leqslant k} [(f(\mathbf{x}_i + d\mathbf{x},\ t_1) - f(\mathbf{y}_j + d\mathbf{x}, t_2)]^2 \tag{7.33}$$

and c is constant. The updating formula is complex in form but basically is a weighted sum of neighboring match probabilities where the neighboring match is consistent (i.e., has nearly the same velocity). A neighboring match k is consistent if

$$\|\mathbf{v}_{ij} - \mathbf{v}_{kl}\| \leqslant dV_{\max} \tag{7.34}$$

The goodness of a particular match is measured by q_{ij}, where

$$q_{ij}^{n-1} = \sum_{k \text{ a neighbor of } i} \quad \sum_{l \text{ s.t. } kl \text{ satisfies (7.34)}} P_{kl}^{n-1} \tag{7.35}$$

and the probabilities are updated by

$$\tilde{P}_{ij}^{n} = P_{ij}^{n-1}(A + Bq_{ij}) \tag{7.36}$$

$$P_{ij}^{n} = \frac{\tilde{P}_{ij}^{n}}{\sum\limits_{j \text{ s.t. } ij \text{ is a match}} \tilde{P}_{ij}^{n}} \tag{7.37}$$

where the function of Eq. (7.36) is to renormalize the probabilities and A and B are constants.

The following simplified example makes these ideas more concrete.

Consider the situation given in Fig. 7.6, where the points in (a) are from S_1 and the points in (b) are from S_2. Using hypothetical values for P^0, an initial match data structure is, in terms of Eq. (7.31):

((4, 10) ((5, 0), 0.7) ((4, −5), 0.25) ((2, −8), 0.05))
((4, 6) ((5, 4), 0.5) ((4, −1), 0.3) ((2, −4), 0.2))
((2, 3) ((7, 7), 0.3) ((6, 2), 0.35) ((4, −1, 0.2))

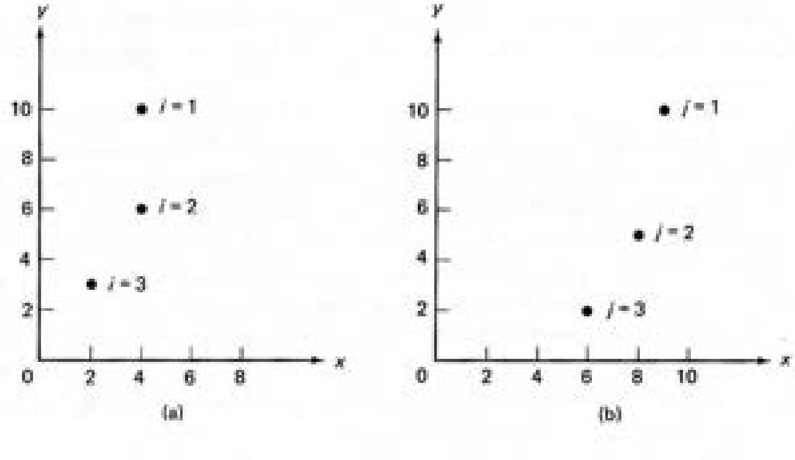


Fig. 7.6 Discrete matching: a concrete example.

Also, $Dv_{max} = 1$, using the chessboard norm. Using the updating formula (7.35), the first set of q_{ij}'s is given by

$$[q_{ij}^1] = \begin{bmatrix} 0.3 & 0.2 & 0 \\ 0 & 0.9 & 0.25 \\ 0 & 0 & 0.3 \end{bmatrix}$$

and the corresponding unnormalized probabilities, with $A = 0.3$ and $B = 3$, are

$$[\tilde{P}_{ij}^1] = \begin{bmatrix} 1.11 & 0.875 & 0.015 \\ 0.15 & 2.79 & 0.80 \\ 0.09 & 0.105 & 0.65 \end{bmatrix}$$

which are normalized to be

$$[P_{ij}^1] = \begin{bmatrix} 0.55 & 0.44 & 0.01 \\ 0.04 & 0.75 & 0.21 \\ 0.11 & 0.12 & 0.74 \end{bmatrix}$$

So after one iteration the match structure is already starting to converge to the best match of $P_{ii} = 1$, $P_{ij} = 0$ for $i \neq j$. Note that in general P_{ij} and q_{ij} are, in matrix form, sparse due to the consistency condition (7.34). To see the results for an example of a more appropriate scale, consult Fig. 7.7.

7.3.2 Rigid Bodies from Motion

The human visual system is predisposed to interpret (perceive) two-dimensional projections of moving three-dimensional rigid objects as just that—moving rigid objects. This facility is an interesting one, since it persists even when all three-dimensions information is removed from any single static view. This sort of result has been known for some time [Wallach and O'Connell 1953; Johansson 1964]. The ability to interpret points as three-dimensional objects demonstrated by Johansson means that the interpretation process does not rely solely on monitoring the changes of angles and length of lines, as suggested by Wallach and O'Connell.

Of course any change between two two-dimensional projections of points in three dimensions can be explained by any number of configurations and motions. Our visual system only accepts a few interpretations, often only one. This one is, in the world of moving objects in which we live, usually correct. This ability to reject unlikely interpretations is consistent with a "rigidity assumption" [Ullman 1979]: Any set of elements undergoing a two-dimensional transformation which has a unique interpretation as a rigid body moving in space should be so interpreted. It seems likely that something like this rigidity assumption is built into our visual system. However, saying that does not tell us much about how it could possibly work. Below we consider the problem of obtaining three-dimensional structure from sets of corresponding two-dimensional points.

One related area of work is the reconstruction of three-dimensional structure when the corresponding points in two dimensions are not known. The reconstruc-

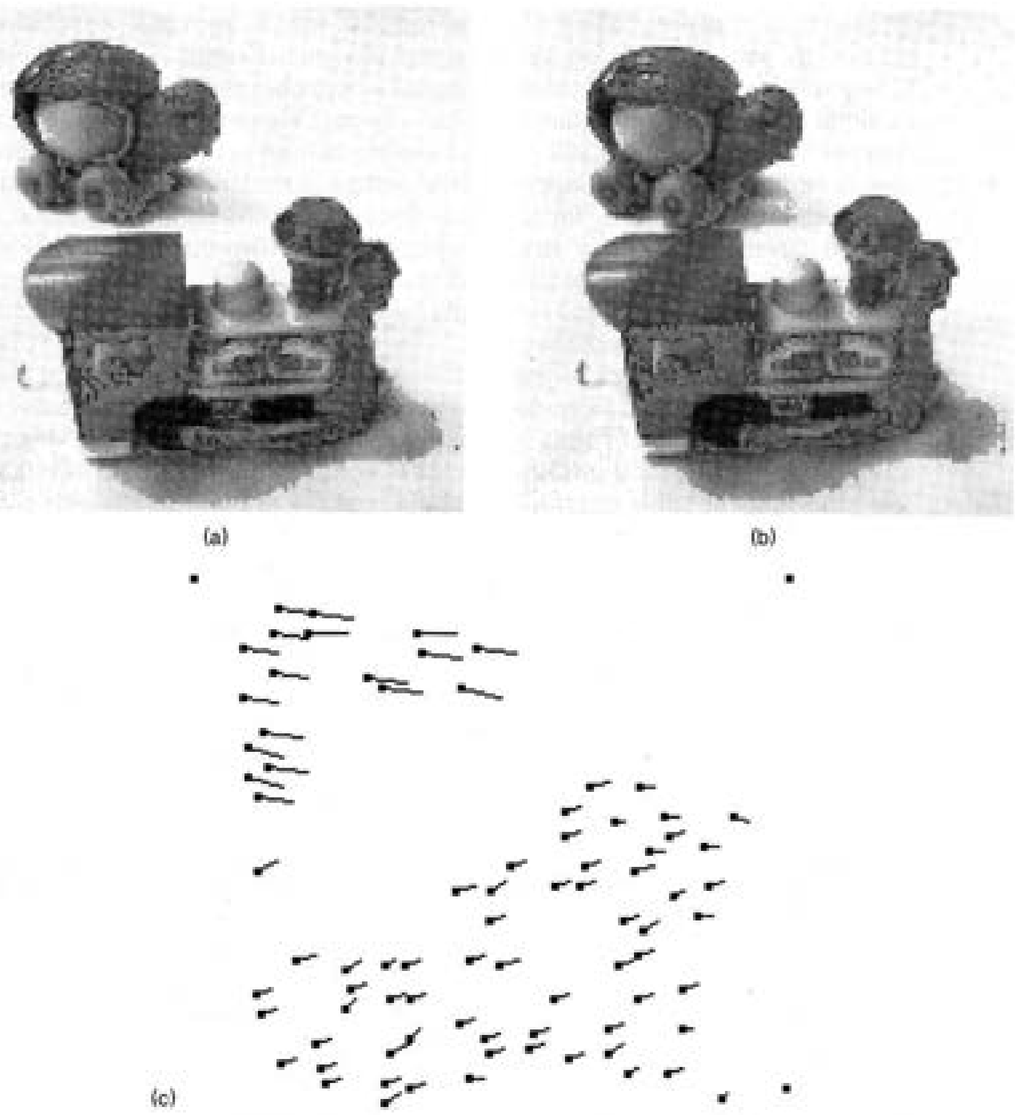


Fig. 7.7 Optical flow from feature point analyses. (a) An image. (b) Later image. (c) Optical flow found by relaxation.

tion procedure must begin by matching points in the several views. It can be shown [Shapira 1974] that general wire-frame objects of straight wires (of which the edges of polyhedra are only a special case) may be reconstructed from a finite number of perspective projections, but that for general wire-frame objects, the number of projections needed may be quite large. In fact, given any set of projections (viewpoints and viewing planes), an object may be constructed that is only ambiguously specified by those projections. Further work on reconstruction from projections is reported in [Shapira and Freeman 1978, Wesley and Markovsky 1981].

If point correspondences are known, it is possible to compute a unique three-dimensional location of four noncoplanar points from just three (orthographic) projections [Ullman 1979]. If the projections result from noncoplanar viewpoints, the recovery of three-dimensional structure is straightforward and is outlined below. If the projections are from coplanar viewpoints, the computations become more complex but still yield a unique result up to reflection. This second case is an important one; it applies if the camera is stationary and the object revolves about a single axis, for instance. Since the reconstruction is unique, the method never gets a wrong structure from accurate two-dimensional evidence about a rigid body. The probability that three views of four nonrigidly connected points can be interpretated as a rigid body is very low. Thus, the method is unlikely to report structure that is not there.

The method may be heuristically extended to multiple objects. Given the capability of describing the three-dimensional structure of four points, one can segment large collections of points by treating them in groups of four, deriving their structure and hence their motion. Groups of points that are not rigid have a very low probability of being interpreted as rigid, and the rest will presumably cluster into sets that share motions associated with rigid objects in the imaged scene. Thus the method to be described may be adaptable for image segmentation.

The calculation may be applied to coplanar points. If a unique result is derived, it is correct; otherwise, the fact that the points are coplanar is revealed. Generally, accuracy of two-dimensional positional information can be sacrificed to some degree if more points or more views are supplied. Perspective projections are more difficult to analyze. Such views can easily be treated approximately by the technique of breaking them into four element groups and treating each group as if it were orthographically projected in a direction depending on its position in the scene. Thus perspective may be dealt with globally, although each group is locally treated as an orthogonal projection. The assumption of orthographic projection implies that the method cannot recover relative depth of objects. The method does not lend itself well to "structure from receding motion" in which the motion information is largely encoded in the perspective effects which render objects larger or smaller as they advance and recede. The method does not serve well to explain human performance on moving images of a few points on nonrigid objects (such as those in Section 7.3.3).

Assume that three orthographic projections of four noncoplanar points are given, and that the correspondence between the points in the projection is known. Translational motion perpendicular to a projection plane is unrecoverable, and translation in a plane parallel to the projection plane is explicitly reproduced in the image by the projection process. The problem thus easily reduces to the case that one of the points is chosen as the origin of coordinates, and stays fixed throughout the process. This treatment follows that of [Ullman 1979].

Let the four points be 0, A, B, and C. Three orthographic views, projections on some planes Π_1, Π_2, and Π_3, are the input to the process. A coordinate system is chosen with origin at 0, and **a**, **b**, and **c** are vectors from 0 to A, B, and C. Then each view has a two-dimensional coordinate system with the image of 0 at its origin. Let $\mathbf{p}_i$ and $\mathbf{q}_i$ be the orthogonal unit basis vectors of the coordinate systems of the Π_i. Let the image coordinates of A, B, and C on Π_i be $(x(a_i), y(a_i))$, $(x(b_i)$,

$y(b_i)$), and $(x(c_i), y(c_i))$ for $i = 1, 2, 3$. The calculations produce vectors $\mathbf{u}_{ij}$, which are unit vectors along the lines of intersection of Π_i with Π_j.

The image coordinates are in fact

$$\begin{aligned} x(a_i) &= \mathbf{a}\cdot\mathbf{p}_i & y(a_i) &= \mathbf{a}\cdot\mathbf{q}_i \\ x(b_i) &= \mathbf{b}\cdot\mathbf{p}_i & y(b_i) &= \mathbf{b}\cdot\mathbf{q}_i \\ x(c_i) &= \mathbf{c}\cdot\mathbf{p}_i & y(c_i) &= \mathbf{c}\cdot\mathbf{q}_i \end{aligned} \tag{7.38}$$

The unit vector $\mathbf{u}_{ij}$ is on both Π_i and Π_j; hence for some r_{ij}, s_{ij}, t_{ij}, and v_{ij},

$$\mathbf{u}_{ij} = r_{ij}\mathbf{p}_i + s_{ij}\mathbf{q}_i \tag{7.39}$$

$$r_{ij}^2 + s_{ij}^2 = 1$$

$$\mathbf{u}_{ij} = t_{ij}\mathbf{p}_j + v_{ij}\mathbf{q}_j \tag{7.40}$$

$$t_{ij}^2 + v_{ij}^2 = 1$$

Equations (7.39) and (7.40) yield

$$r_{ij}\mathbf{p}_i + s_{ij}\mathbf{q}_i = t_{ij}\mathbf{p}_j + v_{ij}\mathbf{q}_j \tag{7.41}$$

Taking the scalar product of **a**, **b**, and **c** with Eq. (7.41) yields three more equations, which are linearly independent. These equations in r_{ij}, s_{ij}, t_{ij}, and v_{ij}, combined with Eqs. (7.39) and (7.40), yield two solutions differing only in sign. But this means that (up to a sign) $\mathbf{u}_{ij}$ is determined in terms of the image coordinate basis vectors $(\mathbf{p}_i, \mathbf{q}_i)$ and $(\mathbf{p}_j, \mathbf{q}_j)$. Two **u** vectors determine one of the planes of orthogonal projection. For instance, $\mathbf{u}_{13}$ and $\mathbf{u}_{23}$ lie in P_3. Given the plane equation for the Π_i, the three-dimensional locations are computed as the intersection of lines perpendicular to the Π_i and through the two-dimensional image points. Of course, because of the ambiguity in sign, the expected mirror image ambiguity of structure exists.

The extension to the case that $\mathbf{u}_{12} = \mathbf{u}_{23} = \mathbf{u}_{31}$, where the three viewpoints are coplanar, is not difficult. It is perhaps a little surprising that coplanar viewpoints still yield a unique interpretation.

An extension of the mathematics to perspective imaging is not difficult to formulate, but the equations are nonlinear and must be solved either conventionally, say by the multidimensional Newton-Raphson technique of Appendix 1, or perhaps by cooperative algorithms of a more artificial intelligence flavor [Lawton 1981].

In geometrically underconstrained situations, plausible interpretations can sometimes be made by using other knowledge to give constraints. For example, one can minimize a second-difference approximation to the acceleration of points in order to use the "constraint" of smooth motion. Such a criterion may find a single "best" location for points. Another example is the use of position and velocity commonality over time to establish rigid members in linkages (Section 7.3.3), a first step to location determination.

To see how the equations might be set up, consider the perspective geometry of Section 7.2.1. In this simplified Cartesian system, Eqs. (7.1) and (7.2) are used as before. Since $z(x', y', 1) = (x, y, z)$, the location of any point is determined (up to a scale factor, since the focal length is not explicit) from its image coordinates and its depth coordinate, z. For $F \geqslant 1$ images and $N \geqslant 3$ points there are $FN - 1$ unknowns (the ability to scale distance allows one point to be placed arbitrarily).

To apply the rigid body constraint, enough pairwise distances between points must be specified to lock them into a rigid configuration. For three points, three distances are necessary. Each additional point requires another three distances, and so for each interframe interval $3(N-2)$ constraints are needed, for a total of $3(F-1)(N-2)$ constraints. Thus, whenever

$$2FN - 6F - 3N + 7 \geqslant 0 \tag{7.42}$$

consistent equations from the constraints can be solved [Lawton 1981]. With two views, five points are needed; with three views, four points. This is not surprising, given the preceding analysis for orthographic projections.

Consider the simple case of two points seen in two frames. If they are rigidly connected, one constraint equation holds. It is equivalent to

$$(\mathbf{x}_{11} - \mathbf{x}_{12})\cdot(\mathbf{x}_{11} - \mathbf{x}_{12}) = (\mathbf{x}_{21} - \mathbf{x}_{22})\cdot(\mathbf{x}_{21} - \mathbf{x}_{22}) \tag{7.43}$$

($\mathbf{x}_{ij}$, $\mathbf{x}'_{ij}$ are, respectively, the world and image coordinate vectors of point j in frame ij). Since $\mathbf{x}_{ij} = z_{ij}\mathbf{x}'_{ij}$, (recall (7.1) and (7.2)) the constraint becomes

$$\begin{aligned} &z_{11}^2(\mathbf{x}'_{11}\cdot\mathbf{x}'_{11}) + z_{12}^2(\mathbf{x}'_{12}\cdot\mathbf{x}'_{12}) - 2z_{11}z_{12}(\mathbf{x}'_{11}\cdot\mathbf{x}'_{12}) \\ &- z_{21}^2(\mathbf{x}'_{21}\cdot\mathbf{x}'_{21}) - z_{22}^2(\mathbf{x}'_{22}\cdot\mathbf{x}'_{22}) + 2z_{21}z_{22}(\mathbf{x}'_{21}\cdot\mathbf{x}'_{22}) = 0 \end{aligned} \tag{7.44}$$

A further constraint that objects only move in the "ground plane," or at a constant y, has the effect of removing two unknowns through substitution in the constraint equation above. Since for arbitrary m and n,

$$y_{im} = z_{im}y'_{im} = y_{in} = z_{in}y'_{in} \tag{7.45}$$

$$z_{in} = \frac{z_{im}y'_{im}}{y'_{in}} \tag{7.46}$$

As a final example, a restriction to purely translational motion of the point configurations yields the constraint

$$(\mathbf{x}_{11} - \mathbf{x}_{21}) - (\mathbf{x}_{12} - \mathbf{x}_{22}) = 0 \tag{7.47}$$

Expanding this as the product of unknown depths (z) and known image positions ($\mathbf{x}'$) yields a vector equation that may be written componentwise as three linear equations in four unknowns. Recall that a focal length must be fixed, effectively setting one unknown: setting one z_{ij} to 1 gives a system of three linear equations in the other three z_{ij}.

7.3.3 Interpretation of Moving Light Displays—A Domain-Independent Approach

One of the domains that provides the purest aspects of motion vision is moving light displays (MLDs). These are sequences of images which track only a few discrete points per frame. A typical way to produce an MLD is to attach small glass bead reflectors to a person's major joints (shoulders, elbows, wrists, hips, knees, ankles), focus a strong light on him or her, and manipulate the contrast of a videotape recorder so as to produce on videotape a record of the movement of the reflective points on the joints. A single frame from such a record is unrecognizable by an inexperienced subject (Fig. 7.8).

However, a sequence of such frames quickly gives (typically in 0.4 second) not only a compelling perception of motion of a three-dimensional body, but allows recognition of the sequence as depicting a walking person, and a description of the type of motion (walking backward, jumping, walking left). Complicated scenes such as several independently moving bodies and couples dancing can be recognized. Sophisticated judgments can be made, such as determining the sex of a subject from an MLD, or recognizing the gait of a friend [Johannson 1964].

MLDs thus present quite a challenge to computer vision. It could be that MLDs of moving people are interpreted by specialized neural mechanisms expressly tailored to the purpose of dealing with any visual input whatever that sug-

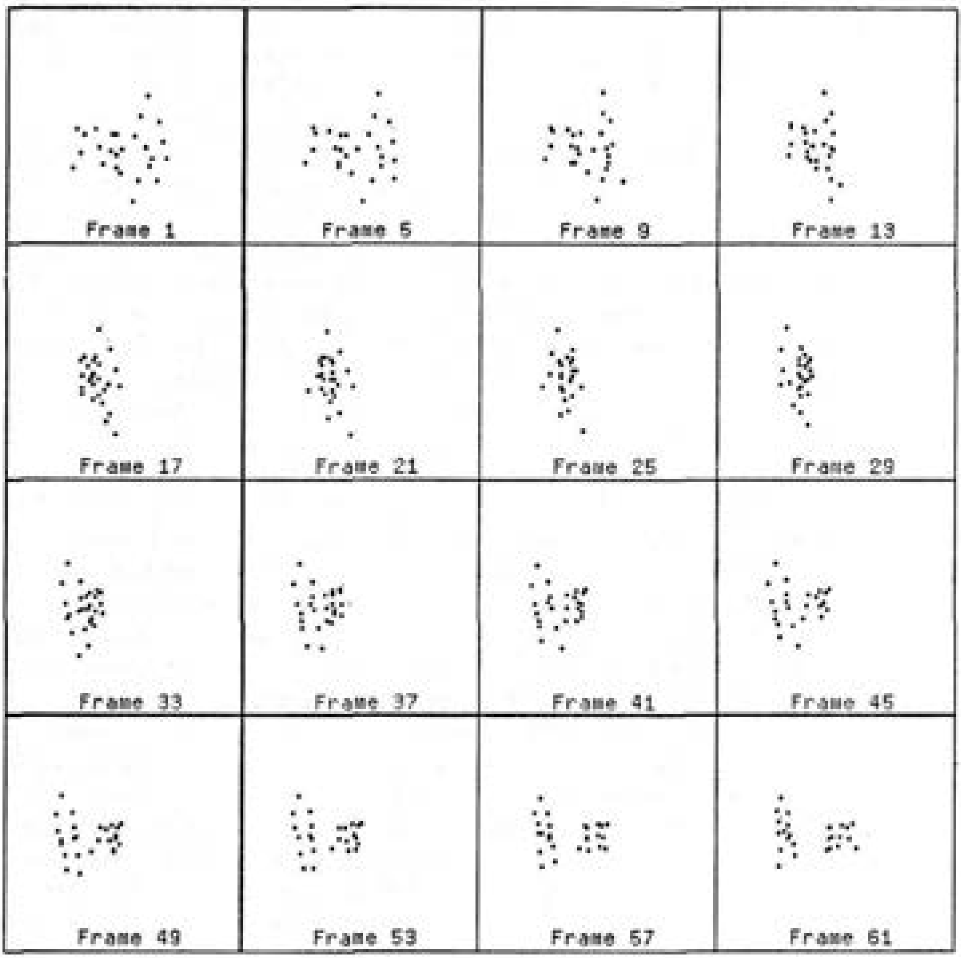


Fig. 7.8 An MLD for a man walking his dog.

gests moving people. MLDs certainly demonstrate that texture, continuous fields of flow, and especially that the interpretability of static versions of the scene are not necessary for human beings to do complex perception of certain three-dimensional objects.

This section is concerned with MLDs of moving human beings, and the interpretation we desire consists of separating images of individuals, in deriving their "connectivity" (i.e., the rigid links that connect the points), and possibly in describing the three-dimensional motion in which the subjects are engaged.

MLDs produced with perspective projection have few of the pleasant properties of the rigid orthographic projection which were used in Section 7.3.1. In particular, both translating and rotating objects are inherently ambiguous in perspective projections [Roache and Aggarwal 1979]. The approximate method outlined in Section 7.3.1, in which local groups of four points are considered rigid and orthographically projected, fails for MLDs of walking people. In many applications, digitization error will limit severely the accuracy returned. Worse, in a typical 12-point MLD of a moving person, there is never a rigid system of four noncoplanar points. The small departures from rigidity occurring in 30 ms of normal walking are enough to render the rigidity assumptions invalid [Rashid 1980].

An algorithm in [Badler 1975] extracts the trajectory of two moving points if they move in parallel paths and are viewed by spherical projection. The projection conditions are approximately met in typical moving-person MLDs, but the lack of points moving in parallel paths is enough to render the algorithm inapplicable.

A good start in the interpretation of MLDs involves solving the point-correspondence problem between frames. Knowing how points move from frame to frame gives at least a start on perceiving the continuity of the objects in the scene. Solving this problem from frame to frame may be attacked in any number of ways; the relaxation approach of Section 7.2.3 is an example.

Another is to predict the location of a point in the two-dimensional image from its velocity in the preceding frame. Velocity is computed from the differences in position of the point in the preceding two frames. Predicting where a point will be in frame 3 implies that one knew which point it was in frames 1 and 2. One way of getting the process started is to associate points in frames 1 and 2 that are nearest neighbors. Evidence suggests that human beings in fact are not infallible trackers of points in MLDs [Rashid 1980]. However, they do not let local inconsistencies in point interpretation (say, if the ankle momentarily "turns into" the knee) detract from their overall perception of a moving person. This is a good example of how inconsistent interpretations arise in human vision.

A program can be given similar resilience by having it suspend judgment on contradictory clues and use succeeding frames to resolve the problem [Rashid 1980; O'Rourke 1980]. Having established local point correspondences, the next problem is to group the points into coherent three-dimensional structures and separate individual bodies moving in the scene. When constraints on the scene are available that make analytic techniques applicable (Section 7.3.1), explicit grouping of points prior to analysis may be unnecessary. In fact, with complex MLDs such as Ullman studied (e.g. two transparent but spotty coaxial cylinders rotating in opposite directions about an axis in the viewing plane), most naive grouping strategies based on two-dimensional motion in the image will fail. Ullman's method chooses four-tuples of points from such a scene; on the average seven-eighths of such groups involve points from both cylinders, but with accurate data the algorithm can identify such nonrigid four-tuples. The remaining one-eighth of the groups have consistent interpretations as rigid rotating groups, and the groups fall into two classes, one for each cylinder.

One straightforward heuristic approach to MLD interpretation enjoys moderate success and does not use domain-dependent models [Rashid 1980]. It has the characteristic that it deals exclusively with two-dimensional motions in order to extract information about three dimensions. The approach is more heuristic than Lawton's and certainly more than Ullman's (Section 7.3.1). It is prey to many of the same pitfalls that threaten any image-based (as opposed to world-based) approach to computer vision. With sparse MLDs of nonrigid objects, clustering algorithms may be used to group points into related structures. Rashid's method computes the minimum spanning tree of points in a four-dimensional space of two-dimensional position and two-dimensional velocity. That is, each point in the MLD is represented at any time t by a four-vector

$$(x(t),\ y(t),\ u(t),\ v(t))$$

where u and v are the velocity in image x and y coordinates. Points may be clustered in this position-velocity space on the basis of a four-dimensional Euclidean metric, modified by information about distances derived from preceding frames. Perspective distortion can affect the usefulness of two-dimensional distances computed in previous frames, and data scaling is useful to establish a reasonable relation between units in the four-dimensional space. Rashid's technique is to scale the data in each dimension to have unit variance and zero mean, and to compute cumulative distances between points in a frame by a function such as

$$D_n(i,j) = d(i,j) + D_{n-1}(i,j) \times 0.95 \tag{7.48}$$

where $D_n(i,j)$ is the cumulative distance between points i and j in frame n, and $d(i,j)$ is their Euclidean distance.

This clustering method can successfully group points on the two cylinders in the rotating-cylinder sequence mentioned above after seven frames. Figure 7.9 gives the results of clustering the data for the MLD of Fig. 7.8. Clustering is stable after some 25 frames (about one-half of a step).

7.3.4 Human Motion Understanding—A Model-Directed Approach

Human motion understanding may be done with a much different approach than the heuristic clustering applied to MLDs in Section 7.3.3. A very detailed model of the domain can help restrict search, make inferences, disambiguate clues, and so forth. A program for understanding images of human motion successfully uses such an approach [O'Rourke 1980; O'Rourke and Badler 1980].

The body model accounts for such factors as relative location of body parts, joint angle ranges, joint angle acceleration limits, collision checking, and gravity. A motion simulation program drives a "bubble man" representation of a person

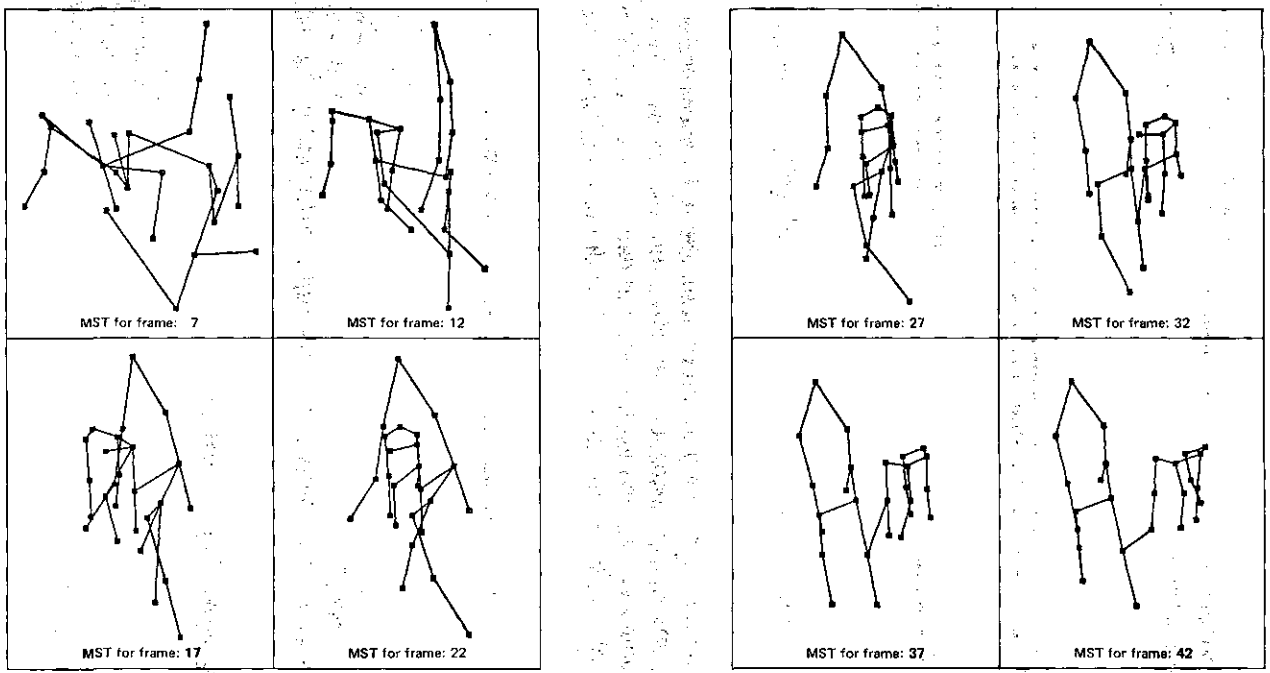


Fig. 7.9 The minimal spanning tree for the man and dog.

(Fig. 7.10a) [Badler and Smoliar 1979]. This representation is used to produce a shaded graphic rendition which serves as input to the motion understanding program (Fig. 7.10b). Knowledge of the imaging process also provides constraints on the configuration of the figure represented. For instance, perspective, the figure/ground distinction, the location of features, and occlusion all have implications for the interpretation of the scene as a configuration of the model.

The system is another example of a cooperative, constraint-satisfying system (Chapter 12), this time one that involves a high-level domain-dependent model.

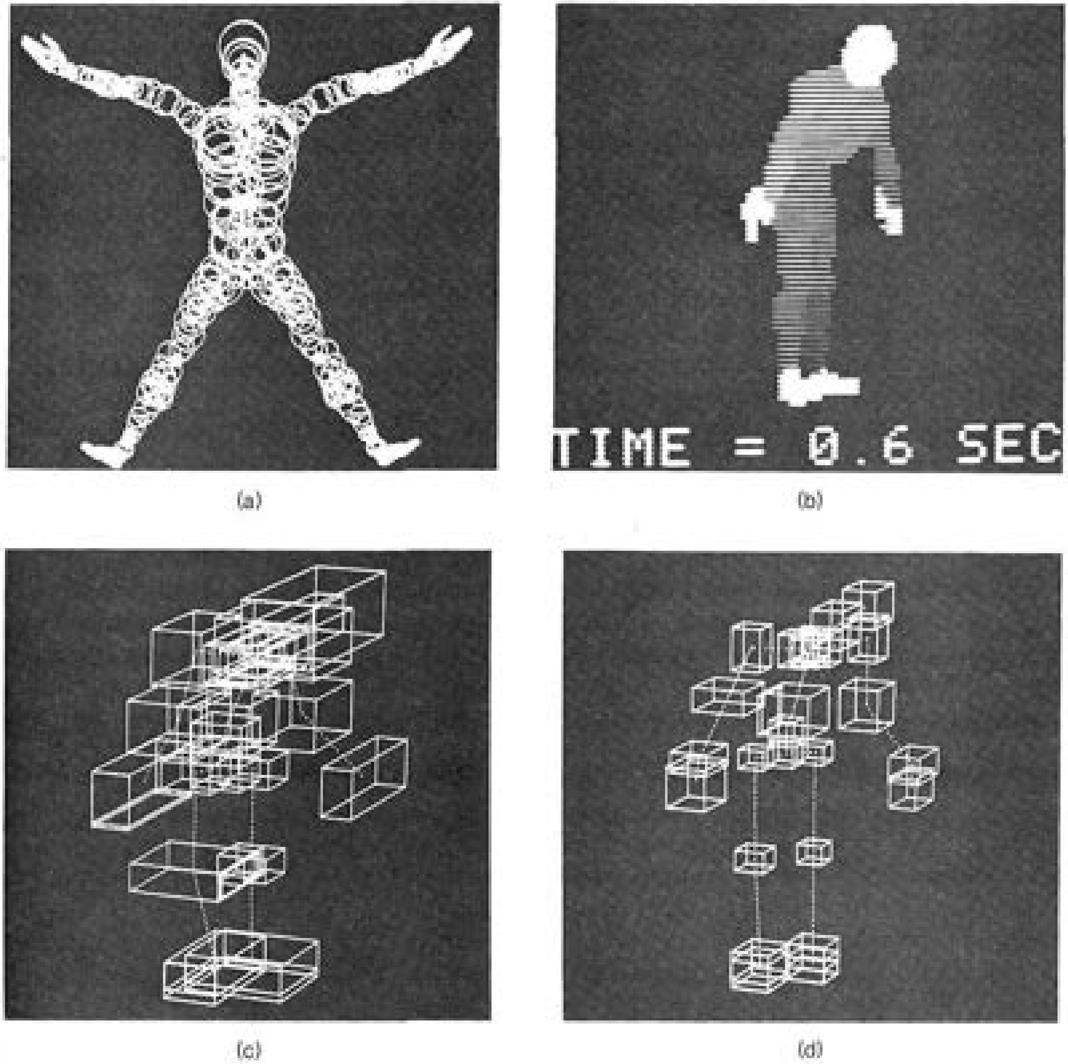


Fig. 7.10 Understanding human motion through the incorporation of many constraints. (a) Bubble Man from simulation program. (b) Input to motion understander; a bowing man. (c, d) Initial and final stages in understanding the motion of the bowing man.

The constraints imposed by the model restrict the application of low-level operators, and their results reduce uncertainty in parts of the model configuration. Through the relations between model parts, improved estimates for part locations are evolved and propagate throughout the model. Figure 7.10c and d show how the image of the bowing man is understood more accurately as time passes and more constraints are propagated through the model. It should be noted that only the hand, foot, and head features are explicitly searched for in the image. The boxes represent possible locations for the obvious body parts. Note how the occlusion has been understood.

7.3.5 Segmented Images

Moving Polygons and Line Drawings

As one step along the way to motion understanding, the analysis of ideal polygonal images was popular for a time [Aggarwal and Duda 1975; Martin and Aggarwal 1978; Potter 1975]. The assumptions are usually that opaque polygons move in parallel planes and may obscure one another (this is often called a 2.5-dimensional situation). The viewpoint is somewhere "above" the collection of moving shapes. The viewer (program) is presented with a sequence of frames either of line drawings or gray level images of the scene (Fig. 7.11). Polygon motion is assumed small between frames. The goal is usually to segment the scenes into polygons, and to extract such information as their direction and speed of motion. The solutions to these problems usually reflect assumptions about the connectivity of the polygons, or restrictions on their motion, and often revolve about the allowable topological and geometrical transformations that can take place in such scenes.

For instance, in a frame with two polygons such as that shown in Fig. 7.12, certain scene vertices belong to primitive polyhedra (they are "true" vertices), whereas others are "false" artifacts of occlusion. The lines impinging at true vertices will not change their angle of meeting through time, but false vertices may change angles if the polygons rotate as they move. False vertices are usually obtuse.

Complex connectivity changes can arise when nonconvex polygons slide past one another. Sorting out a coherent interpretation of a sequence of frames, especially in the presence of noisy vertex positions, is a challenging exercise.

A system was designed in [Badler 1975] which used sequences of line drawings produced by a spherical projection of a three-dimensional world to reconstruct

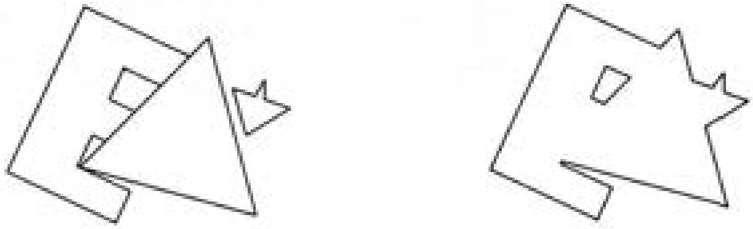


Fig. 7.11 Two frames from a motion image of three moving polygons.