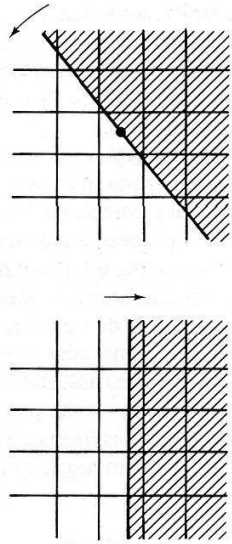


Fig. 3.11 Edge models for orientation and displacement sensitivity analyses.

$$\phi' = \begin{cases} \phi & \text{if } 0 \leqslant \phi \leqslant \tan^{-1}\left(\frac{1}{3}\right) \\ \tan^{-1}\left(\dfrac{7\tan^2\phi + 6\tan\phi - 1}{-9\tan^2\phi + 22\tan\phi - 1}\right) & \text{if } \tan^{-1}\left(\frac{1}{3}\right) \leqslant 0 \leqslant \phi \leqslant \pi/4 \end{cases} \quad (3.25)$$

Arguments from symmetry show that only the $0 \leqslant \phi < \pi/4$ cases need be examined. Similar studies could be made using ramp edge models.

A rather specialized kind of gradient is that taken "between pixels." This scheme is shown in Fig. 3.12. Here a pixel may be thought of as having four *crack edges* surrounding it, whose directions of are fixed by the pixel to be multiples of $\pi/2$. The magnitude of the edge is determined by $|f(\mathbf{x}) - f(\mathbf{y})|$, where $\mathbf{x}$ and $\mathbf{y}$ are the coordinates of the pixels that have the edge in common. One advantage of this formulation is that it provides an effective way of separating regions and their boundaries. The disadvantage is that the edge orientation is crude.

The *Laplacian* is an edge detection operator that is an approximation to the mathematical Laplacian $\partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$ in the same way that the gradient is an approximation to the first partial derivatives. One version of the discrete Laplacian is given by

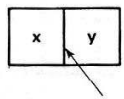


Fig. 3.12 "Crack" edge representation.

$$L(x, y) = f(x, y) - \tfrac{1}{4}[f(x, y+1) + f(x, y-1) + f(x+1, y) + f(x-1, y)] \tag{3.26}$$

The Laplacian has two disadvantages as an edge measure: (1) useful directional information is not available, and (2) the Laplacian, being an approximation to the second derivative, doubly enhances any noise in the image. Because of these disadvantages, the Laplacian has fallen into disuse, although some authors have used it as an adjunct to the gradient [Wechsler and Sklansky 1977; Akatsuka 1974] in the following manner: There is an edge at $\mathbf{x}$ with magnitude $g(\mathbf{x})$ and direction $\phi(\mathbf{x})$ if $g(\mathbf{x}) > T_1$ and $L(\mathbf{x}) > T_2$.

Edge Templates

The *Kirsch operator* [Kirsch 1971] is related to the edge gradient and is given by

$$S(\mathbf{x}) = \max\left[1, \max_k \sum_{k-1}^{k+1} f(\mathbf{x}_k)\right] \tag{3.27}$$

where $f(\mathbf{x}_k)$ are the eight neighboring pixels to $\mathbf{x}$ and where subscripts are computed modulo 8. A 3-bit direction can also be extracted from the value of k that yields the maximum in (3.27). In practice, "pure" template matching has replaced the use of (3.27). Four separate templates are matched with the image and the operator reports the magnitude and direction associated with the maximum match. As one might expect, the operator is sensitive to the magnitude of $f(\mathbf{x})$, so that in practice variants using large templates are generally used. Figure 3.13 shows Kirsch-motivated templates with different spans.

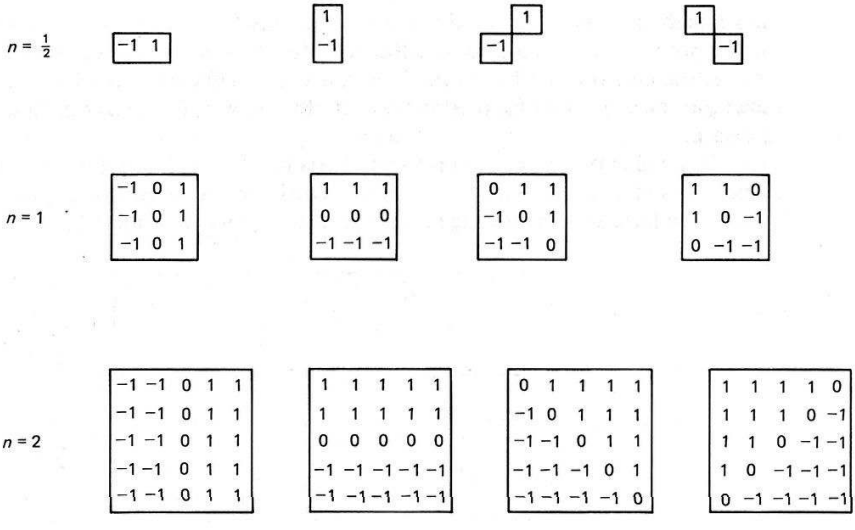


Fig. 3.13 Kirsch templates.

This brief discussion of edge templates should not be construed as a comment on their appropriateness or popularity. In fact, they are widely used, and the template-matching concept is the essence of the other approaches. There is also evidence that the mammalian visual system responds to edges through special low-level template-matching edge detectors [Hubel and Wiesel 1979].

3.3.2 Edge Thresholding Strategies

For most images there will be but few places where the gradient magnitude is equal to zero. Furthermore, in the absence of any special context, small magnitudes are most likely to be due to random fluctuations, or noise in the image function f. Thus in practical cases one may use the expedient of only reporting an edge element at $\mathbf{x}$ if $g(\mathbf{x})$ is greater than some threshold, in order to reduce these noise effects.

This strategy is computationally efficient but may not be the best. An alternative thresholding strategy [Frei and Chen 1977] views difference operators as part of a set of orthogonal basis functions analogous to the Fourier basis of Section 2.2.4. Figure 3.14 shows the nine Frei–Chen basis functions. Using this basis, the image near a point $\mathbf{x}_0$ can be represented as

$$f(\mathbf{x}) = \sum_{k=1}^{8} (f, h_k) h_k(\mathbf{x} - \mathbf{x}_0) / (h_k, h_k) \tag{3.28}$$

where the (f, h_k) is the correlation operation given by

$$(f, h_k) = \sum_{D} f(\mathbf{x}_0) h_k(\mathbf{x} - \mathbf{x}_0) \tag{3.29}$$

and D is the nonzero domain of the basis functions. This operation is also regarded as the *projection* of the image into the basis function h_k. When the image can be reconstructed from the basis functions and their coefficients, the basis functions span the space. In the case of a smaller set of functions, the basis functions span a subspace.

The value of a projection into any basis function is highest when the image function is identical to the basis function. Thus one way of measuring the "edgeness" of a local area in an image is to measure the relative projection of the image

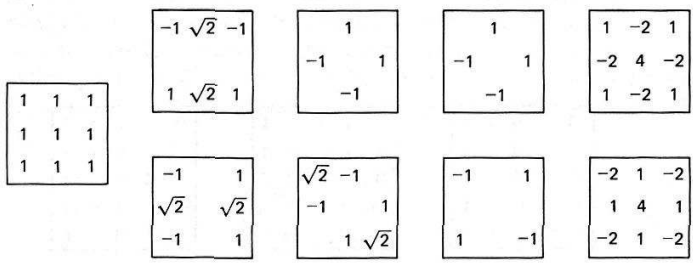


Fig. 3.14 Frei-Chen orthogonal basis.

into the edge basis functions. The relative projection into the particular "edge subspace" is given by

$$\cos\theta = \left(\frac{E}{S}\right)^{1/2} \tag{3.30}$$

where

$$E = \sum_{k=1}^{2} (f, h_k)^2$$

and

$$S = \sum_{k=0}^{8} (f, h_k)^2$$

Thus if $\theta < T$, report an edge; otherwise, not. Figure 3.15 shows the potential advantage of this technique compared to the technique of thresholding the gradient magnitude, using two hypothetical projections B_1 and B_2. Even though B_2 has a small magnitude, its relative projection into edge subspace is large and thus would be counted as an edge with the Frei–Chen criterion. This is not true for B_1.

Under many circumstances it is appropriate to use model information about the image edges. This information can affect the way the edges are interpreted after they have been computed or it may affect the computation process itself. As an example of the first case, one may still use a gradient operator, but vary the threshold for reporting an edge. Many versions of the second, more extreme strategies of using special spatially variant detection methods have been tried [Pingle and Tenenbaum 1971; Griffith 1973; Shirai 1975]. The basic idea is illustrated in Fig. 3.16. Knowledge of the orientation of an edge allows a special orientation-sensitive operator to be brought to bear on it.

3.3.3 Three-Dimensional Edge Operators

In many imaging applications, particularly medicine, the images are three-dimensional. Consider the examples of the reconstructed planes described in Sections 1.1 and 2.3.4. The medical scanner that acquires these data follows several parallel image planes, effectively producing a three-dimensional volume of data.

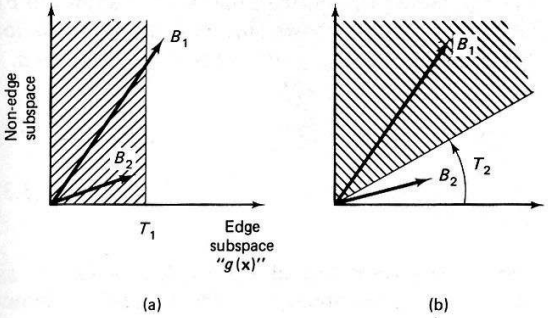
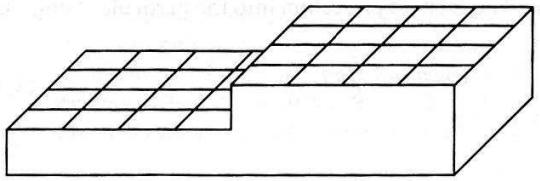
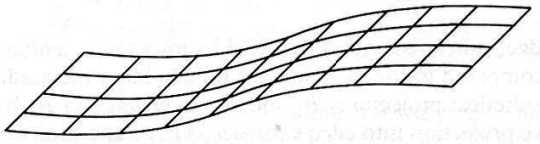


Fig. 3.15 Comparison of thresholding techniques.



(a)



(b)

Fig. 3.16 Model-directed edge detection.

In three-dimensional data, boundaries of objects are surfaces. Edge elements in two dimensions become surface elements in three dimensions. The two-dimensional image gradient, when generalized to three dimensions, is the local surface normal. Just as in the two-dimensional case, many different basis operators can be used [Liu 1977; Zucker and Hummel 1979]. That of Zucker and Hummel uses an optimal basis assuming an underlying continuous model. We shall just describe the operator here; the proof of its correctness given the continuous image model may be found in the reference. The basis functions for the three-dimensional operator are given by

$$g_1(x, y, z) = \frac{x}{r} \tag{3.31}$$

$$g_2(x, y, z) = \frac{y}{r}$$

$$g_3(x, y, z) = \frac{z}{r}$$

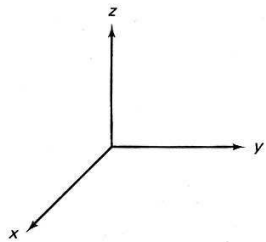
where $r = (x^2 + y^2 + z^2)^{1/2}$. The discrete form of these operators is shown in Fig. 3.17 for a $3 \times 3 \times 3$ pixel domain D. Only g_1 is shown since the others are obvious by symmetry. To apply the operator at a point x_0, y_0, z_0 compute projections a, b, and c, where

$$a = (g_1, f) = \sum_D g_1(\mathbf{x}) f(\mathbf{x}-\mathbf{x}_0)$$

$$b = (g_2, f) \tag{3.32}$$

$$c = (g_3, f)$$

The result of these computations is the surface normal $\boldsymbol{n} = (a, b, c)$ at (x_0, y_0, z_0). Surface thresholding is analogous to edge thresholding: Report a surface element



$-\frac{\sqrt{3}}{3}$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{3}}{3}$
$-\frac{\sqrt{2}}{2}$	-1	$-\frac{\sqrt{2}}{2}$
$-\frac{\sqrt{3}}{3}$	$-\frac{\sqrt{2}}{2}$	$-\frac{\sqrt{3}}{3}$

0	0	0
0	0	0
0	0	0

$\frac{\sqrt{3}}{3}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{3}$
$\frac{\sqrt{2}}{2}$	1	$\frac{\sqrt{2}}{2}$
$\frac{\sqrt{3}}{3}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{3}$

Fig. 3.17 The $3 \times 3 \times 3$ edge basis function $g_1(x, y, z)$.

only if $s(x, y, z) = |\boldsymbol{n}|$ exceeds some threshold. Figure 3.18 shows the results of applying the operator to a synthetic three-dimensional image of a torus. The display shows small detected surface patches.

3.3.4 How Good are Edge Operators?

The plethora of edge operators is very difficult to compare and evaluate. For example, some operators may find most edges but also respond to noise; others may be

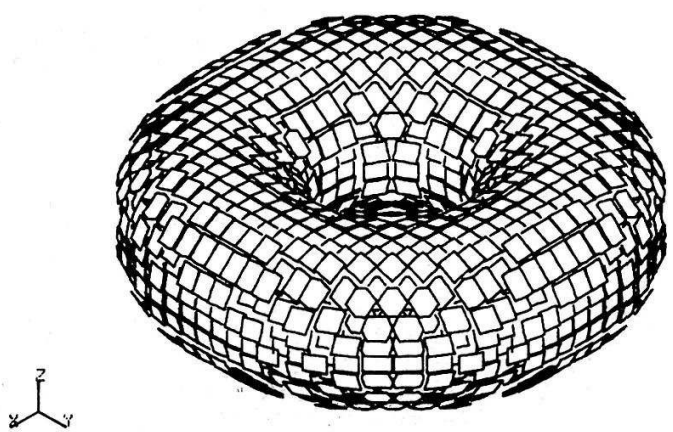


Fig. 3.18 Results of applying the Zucker-Hummel 3-D operator to synthetic image data in the shape of a torus.

noise-insensitive but miss some crucial edges. The following figure of merit [Pratt 1978] may be used to compare edge operators:

$$F = \frac{1}{\max(N_A, N_I)} \sum_{i=1}^{N_A} \frac{1}{1 + (ad_i^2)} \tag{3.33}$$

where N_A and N_I represent the number of actual and ideal edge points, respectively, a is a scaling constant, and d is the signed separation distance of an actual edge point normal to a line of ideal edge points. The term ad_i^2 penalizes detected edges which are offset from their true position; the penalty can be adjusted via a. Using this measure, all operators have surprisingly similar behaviors. Unsurprisingly, the performance of each deteriorates in the presence of noise [Abdou 1978]. (Pratt defines a signal-to-noise ratio as the square of the step edge amplitude divided by the standard deviation of Gaussian white noise.) Figure 3.19 shows some typical curves for different operators. To make this figure, the threshold for reporting an edge was chosen independently for each operator so as to maximize Eq. (3.33).

These comparisons are important as they provide a gross measure of differences in performance of operators even though each operator embodies a specific edge model and may be best in special circumstances. But perhaps the more important point is that since all real-world images have significant amounts of noise, all edge operators will generally produce imperfect results. This means that in considering the overall computer vision problem, that of building descriptions of objects, the efforts are usually best spent in developing methods that can use or improve the measurements from unreliable edges rather than in a search for the ideal edge detector.

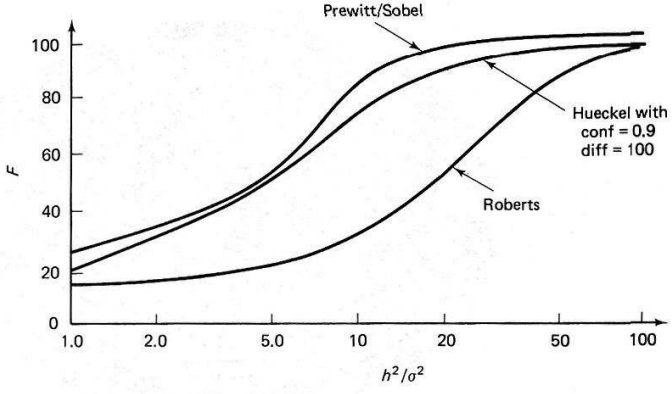


Fig. 3.19 Edge operator performance using Pratt's measure (Eq. 3.33).

3.3.5 Edge Relaxation

One way to improve edge operator measurements is to adjust them based on measurements of neighboring edges. This is a natural thing to want to do: If a weak horizontal edge is positioned between two strong horizontal edges, it should gain credibility. The edges can be adjusted based on local information using parallel-iterative techniques. This sort of process is related to more global analysis and is complementary to sequential approaches such as edge tracking (Chapter 4).

Early cooperative edge detection techniques used pairwise measurements between pixels [Zucker et al. 1977]. A later version [Prager 1980] allows for more complicated adjustment formulas. In describing the edge relaxation scheme, we essentially follow Prager's development and use the crack edges described at the end of the discussion on gradients (Sec. 3.31). The development can be extended to the other kinds of edges and the reader is invited to do just this in the Exercises.

The overall strategy is to recognize local edge patterns which cause the confidence in an edge to be modified. Prager recognizes three groups of patterns: patterns where the confidence of an edge can be increased, decreased, or left the same. The overall structure of the algorithm is as follows:

Algorithm 3.1 Edge Relaxation

0. Compute the initial confidence of each edge $C^0(e)$ as the normalized gradient magnitude normalized by the maximum gradient magnitude in the image.

1. $k = 1$;

2. Compute each edge type based on the confidence of edge neighbors;

3. Modify the confidence of each edge $C^k(e)$ based on its edge type and its previous confidence $C^{k-1}(e)$;

4. Test the $C^k(e)$'s to see if they have all converged to either 0 or 1. If so, stop; else, increment k and go to 2.

The two important parts of the algorithm are step 2, computing the edge type, and step 3, modifying the edge confidence.

The edge-type classification relies on the notation for edges (Fig. 3.20). The edge type is a concatenation of the left and right vertex types. Vertex types are computed from the strength of edges emanating from a vertex. Vertical edges are handled in the same way, exploiting the obvious symmetries with the horizontal case. Besides the central edge e, the left vertex is the end point for three other possible edges. Classifying these possible edges into "edge" and "no-edge" provides the underpinnings for the vertex types in Fig. 3.21.

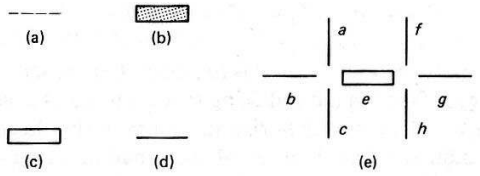


Fig. 3.20 Edge notation. (a) Edge position with no edge. (b) Edge position with edge. (c) Edge to be updated. (d) Edge of unknown strength. (e) Configuration of edges around a central edge e.

To compute vertex type, choose the maximum confidence vertex, i.e., the vertex is type j where j maximizes conf(j)

and

$$\begin{aligned} \text{conf}(0) &= (m-a)(m-b)(m-c) \\ \text{conf}(1) &= a(m-b)(m-c) \\ \text{conf}(2) &= ab(m-c) \\ \text{conf}(3) &= abc \end{aligned}$$

where

$m = \max(a, b, c, q)$
q is a constant (0.1 is about right)

and a, b, and c are the normalized gradient magnitudes for the three edges. Without loss of generality, $a \geqslant b \geqslant c$. The parameter m adjusts the vertex classification so that it is relative to the local maximum. Thus $(a, b, c) = (0.25, 0.01, 0.01)$ is a type 1 vertex. The parameter q forces weak vertices to type zero [e.g., (0.01, 0.001, 0.001) is type zero].

Once the vertex type has been computed, the edge type is simple. It is merely the concatenation of the two vertex types. That is, the edge type is (ij), where i and j are the vertex types. (From symmetry, only consider $i \geqslant j$.)

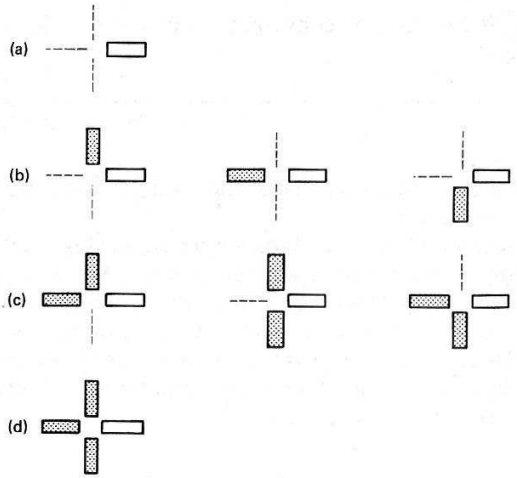


Fig. 3.21 Classification of vertex type of left-hand endpoint of edge e, Fig. 3.20.

Decisions in the second step of modifying edge confidence based on edge type appear in Table 3.1. The updating formula is:

increment: $C^{k+1}(e) = \min(1, C^k(e) + \delta)$
decrement: $C^{k+1}(e) = \max(0, C^k(e) - \delta)$
leave as is: $C^{k+1}(e) = C^k(e)$

where δ is a constant (values from 0.1 to 0.3 are appropriate). The result of using the relaxation scheme is shown in Fig. 3.22. The figures on the left-hand side show

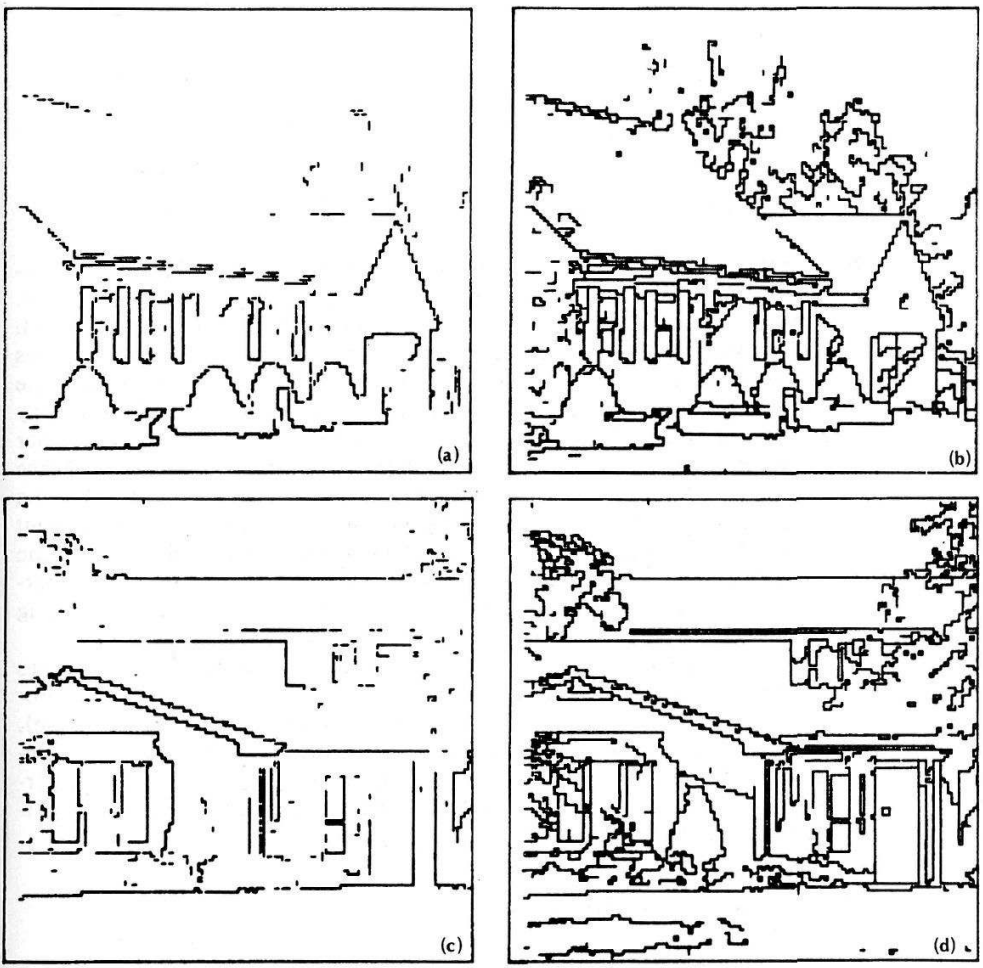


Fig. 3.22 Edge relaxation results. (a) Raw edge data. Edge strengths have been thresholded at 0.25 for display purposes only. (b) Results after five iterations of relaxation applied to (a). (c) Different version of (a). Edge strengths have been thresholded at 0.25 for display purposes only. (d) Results after five iterations of relaxation applied to (c).

the edges with normalized magnitudes greater than 0.25. Weak edges cause many gaps in the boundaries. The figures on the right side show the results of five iterations of edge relaxation. Here the confidence of the weak edges has been increased owing to the proximity of other edges, using the rules in Table 3.1.

Table 3.1

Decrement	*Increment*	*Leave as is*
0-0	*1-1*	*0-1*
0-2	*1-2*	*2-2*
0-3	*1-3*	*2-3*
		3-3

3.4 RANGE INFORMATION FROM GEOMETRY

Neither the perspective or orthogonal projection operations, which take the three-dimensional world to a two-dimensional image, is invertible in the usual sense. Since projection maps an infinite line onto a point in the image, information is lost. For a fixed viewpoint and direction, infinitely many continuous and discontinuous three-dimensional configurations of points could project on our retina in an image of, say, our grandmother. Simple cases are grandmothers of various sizes cleverly placed at varying distances so as to project onto the same area. An astronomer might imagine millions of points distributed perhaps through light-years of space which happen to line up into a "grandmother constellation." All that can be mathematically guaranteed by imaging geometry is that the image point corresponds to one of the infinite number of points on that three-dimensional line of sight. The "inverse perspective" transformation (Appendix 1) simply determines the equation of the infinite line of sight from the parameters of the imaging process modeled as a point projection.

However, a line and a plane not including it intersect in just one point. Lines of sight are easy to compute, and so it is possible to tell where any image point projects on to any known plane (the supporting ground or table plane is a favorite). Similarly, if two images from different viewpoints can be placed in correspondence, the intersection of the lines of sight from two matching image points determines a point in three-space. These simple observations are the basis of light-striping ranging (Section 2.3.3) and are important in stereo imaging.

3.4.1. Stereo Vision and Triangulation

One of the first ideas that occurs to one who wants to do three-dimensional sensing is the biologically motivated one of stereo vision. Two cameras, or one camera from two positions, can give relative depth or absolute three-dimensional location, depending on the elaboration of the processing and measurement. There has been

considerable effort in this direction [Moravec 1977; Quam and Hannah 1974; Binford 1971; Turner 1974; Shapira 1974]. The technique is conceptually simple:

1. Take two images separated by a baseline.
2. Identify points between the two images.
3. Use the inverse perspective transform (Appendix 1) or simple triangulation (Section 2.2.2) to derive the two lines on which the world point lies.
4. Intersect the lines.

The resulting point is in three-dimensional world coordinates.

The hardest part of this method is step 2, that of identifying corresponding points in the two images. One way of doing this is to use correlation, or template matching, as described in Section 3.2.1. The idea is to take a patch of one image and match it against the other image, finding the place of best match in the second image, and assigning a related "disparity" (the amount the patch has been displaced) to the patch.

Correlation is a relatively expensive operation, its naive implementation requiring $0(n^2m^2)$ multiplications and additions for an $m \times m$ patch and $n \times n$ image. This requirement can be drastically improved by capitalizing on the idea of variable resolution; the improved technique is described in Section 3.7.2.

Efficient correlation is of technological concern, but even if it were free and instantaneous, it would still be inadequate. The basic problems with correlation in stereo imaging have to do with the fact that things can look significantly different from different points of view. It is possible for the two stereo views to be sufficiently different that corresponding areas may not be matched correctly. Worse, in scenes with much obscuration, very important features of the scene may be present in only one view. This problem is alleviated by decreasing the baseline, but of course then the accuracy of depth determinations suffers; at a baseline length of zero there is no problem, but no stereo either. One solution is to identify world features, not image appearance, in the two views, and match those (the nose of a person, the corner of a cube). However, if three-dimensional information is sought as a help in perception, it is unreasonable to have to do perception first in order to do stereo.

3.4.2 A Relaxation Algorithm for Stereo

Human *stereopsis*, or fusing the inputs from the eyes into a stereo image, does not necessarily involve being aware of features to match in either view. Most human beings can fuse quite efficiently stereo pairs which individually consist of randomly placed dots, and thus can perceive three-dimensional shapes without recognizing monocular clues in either image. For example, consider the stereo pair of Fig. 3.23. In either frame by itself, nothing but a randomly speckled rectangle can be perceived. All the stereo information is present in the relative displacement of dots in the two rectangles. To make the right-hand member of the stereo pair, a patch of

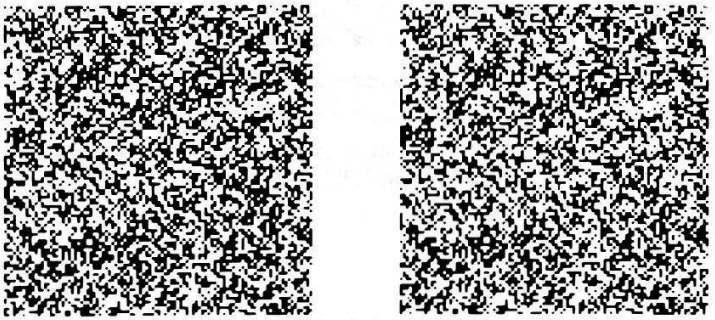


Fig. 3.23 A random-dot stereogram.

the randomly placed dots of the left-hand image is displaced sideways. The dots which are thus covered are lost, and the space left by displacing the patch is filled in with random dots.

Interestingly enough, a very simple algorithm [Marr and Poggio 1976] can be formulated that computes disparity from random dot stereograms. First consider the simpler problem of matching one-dimensional images of four points as depicted in Fig. 3.24. Although only one depth plane allows all four points to be placed in correspondence, lesser numbers of points can be matched in other planes.

The crux of the algorithm is the rules, which help determine, on a local basis, the appropriateness of a match. Two rules arise from the observation that most images are of opaque objects with smooth surfaces and depth discontinuities only at object boundaries:

1. Each point in an image may have only one depth value.
2. A point is almost sure to have a depth value near the values of its neighbors.

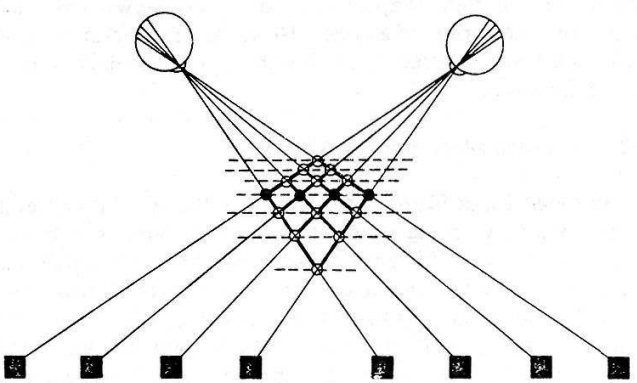


Fig. 3.24 The stereo matching problem.

Figure 3.24 can be viewed as a binary network where each possible match is represented by a binary state. Matches have value 1 and nonmatches value 0. Figure 3.25 shows an expanded version of Fig. 3.24. The connections of alternative matches for a point inhibit each other and connections between matches of equal depth reinforce each other. To extend this idea to two dimensions, use parallel arrays for different values of y where equal depth matches have reinforcing connections. Thus the extended array is modeled as the matrix $C(x, y, d)$ where the point x, y, d corresponds to a particular match between a point (x_1, y_1) in the right image and a point (x_2, y_2) in the left image. The stereopsis algorithm produces a series of matrices C_n which converges to the correct solution for most cases. The initial matrix $C_0(x, y, d)$ has values of one where x, y, d correspond to a match in the original data and has values of zero or otherwise.

Algorithm 3.2 [Marr and Poggio 1976]

Until C satisfies some convergence criterion, do

$$C_{n+1}(x, y, d) = \left\{ \sum_{x',y',d' \in S} C_n(x', y', d') - \sum_{x',y',d' \in \theta} C_n(x', y', d') + C_0(x, y, d) \right\} \quad (3.34)$$

where the term in braces is handled as follows:

$$\{t\} = \begin{cases} 1 & \text{if } t > T \\ 0 & \text{otherwise} \end{cases}$$

S = set of points x', y', d' such that $|\mathbf{x} - \mathbf{x}'| \leqslant 1$ and $d = d'$

θ = set of points x', y', d' such that $|\mathbf{x} - \mathbf{x}'| \leqslant 1$ and $|d - d'| = 1$

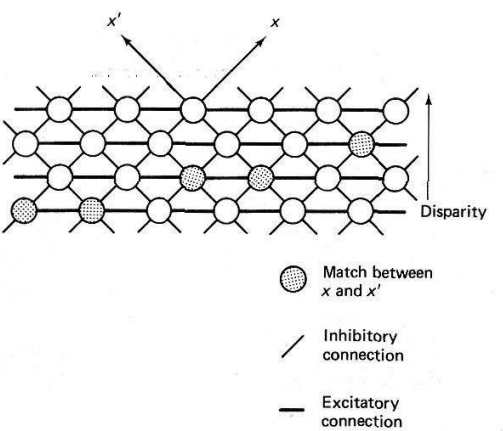


Fig. 3.25 Extension of stereo matching.

One convergence criterion is that the number of points modified on an iteration must be less than some threshold T. Fig. 3.26 shows the results of this computation; the disparity is encoded as a gray level and displayed as an image for different values of n.

A more general version of this algorithm matches image features such as edges rather than points (in the random-dot stereogram, the only features are

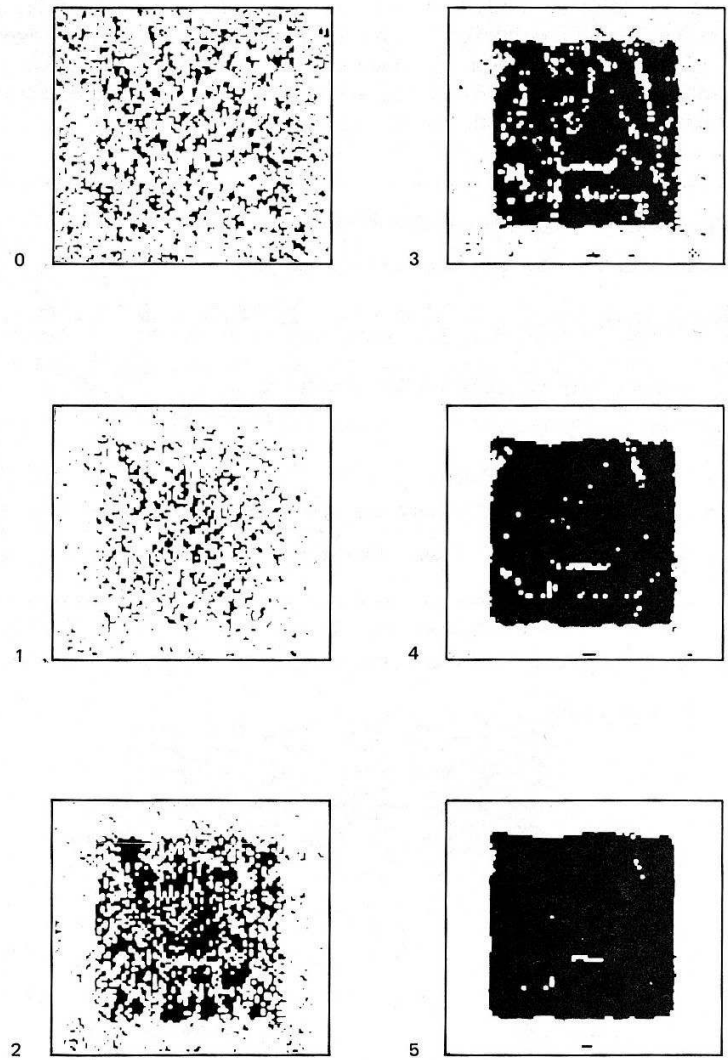


Fig. 3.26 The results of relaxation computations for stereo.

points), but the principles are the same. The extraction of features more complicated than edges or points is itself a thorny problem and the subject of Part II. It should be mentioned that Marr and Poggio have refined their stereopsis algorithm to agree better with psychological data [Marr and Poggio 1977].

3.5 SURFACE ORIENTATION FROM REFLECTANCE MODELS

The ordinary visual world is mostly composed of opaque three-dimensional objects. The intensity (gray level) of a pixel in a digital image is produced by the light reflected by a small area of surface near the corresponding point on the object.

It is easiest to get consistent shape (orientation) information from an image if the lighting and surface reflectance do not change from one scene location to another. Analytically, it is possible to treat such lighting as uniform illumination, a point source at infinity, or an infinite linear source. Practically, the human shape-from-shading transform is relatively robust. Of course, the perception of shape may be manipulated by changing the surface shading in calculated ways. In part, cosmetics work by changing the reflectivity properties of the skin and misdirecting our human shape-from-shading algorithms.

The recovery transformation to obtain information about surface orientation is possible if some information about the light source and the object's reflectivity is known. General algorithms to obtain and quantify this information are complicated but practical simplifications can be made [Horn 1975; Woodham 1978; Ikeuchi 1980]. The main complicating factor is that even with mathematically tractable object surface properties, a single image intensity does not uniquely define the surface orientation. We shall study two ways of overcoming this difficulty. The first algorithm uses intensity images as input and determines the surface orientation by using multiple light source positions to remove ambiguity in surface orientation. The second algorithm uses a single source but exploits constraints between neighboring surface elements. Such an algorithm assigns initial ranges of orientations to surface elements (actually to their corresponding image pixels) on the basis of intensity. The neighboring orientations are "relaxed" against each other until each converges to a unique orientation (Section 3.5.4).

3.5.1 Reflectivity Functions

For all these derivations, consider a distant point source of light impinging on a small patch of surface; several angles from this situation are important (Fig. 3.27).

A surface's reflectance is the fraction of a given incident energy flux (irradiance) it reflects in any given direction. Formally, the *reflectivity function* is defined as $r = \frac{dL}{dE}$, where L is exitant radiance and E is incident flux. In general, for anisotropic reflecting surfaces, the reflectivity function (hence L) is a function of all three angles i, e, and g. The quantity of interest to us is image irradiance, which is proportional to scene radiance, given by $L = \int r\, dE$. In general, the evaluation of this integral can be quite complicated, and the reader is referred to [Horn and

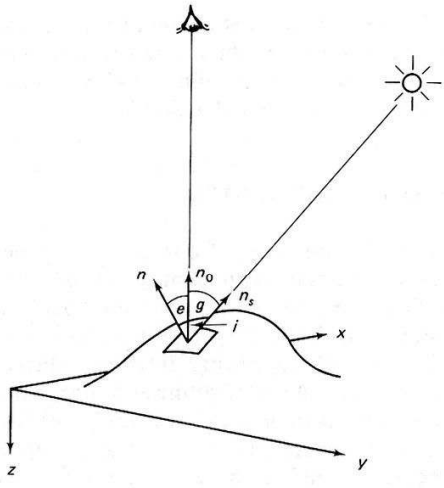


Fig. 3.27 Important reflectance angles: *i*, incidence; *e*, emittance; *g*, phase.

Sjoberg 1978] for a more detailed study. For our purposes we consider surfaces with simple reflectivity functions.

Lambertian surfaces, those with an ideal matte finish, have a very simple reflectivity function which is proportional only to the cosine of the incident angle. These surfaces have the property that under uniform or collimated illumination they look equally bright from any direction. This is because the amount of light reflected from a unit area goes down as the cosine of the viewing angle, but the amount of area seen in any solid angle goes up as the reciprocal of the cosine of the viewing angle. Thus the perceived intensity of a surface element is constant with respect to viewer position. Other surfaces with simple reflectivity functions are "dusty" and "specular" surfaces. An example of a dusty surface is the lunar surface, which reflects in all directions equally. Specular (purely mirror-like) surfaces such as polished metal reflect only at the angle of reflection = angle of incidence, and in a direction such that the incidence, normal, and emittance vectors are coplanar.

Most smooth things have a specular component to their reflection, but in general some light is reflected at all angles in decreasing amounts from the specular angle. One way to achieve this effect is to use the cosine of the angle between the predicted specular angle and the viewing angle, which is given by C where

$$C = 2\cos(i)\cos(e) - \cos(g)$$

This quantity is unity in the pure specular direction and falls off to zero at $\frac{\pi}{2}$ radians away from it. Convincing specular contributions of greater or less sharpness are produced by taking powers of C. A simple radiance formula that allows the simulation of both matte and specular effects is

$$L(i, e, g) = s(C)^n + (1 - s)\cos(i) \tag{3.35}$$

Here s varies between 0 and 1 and determines the fraction of specularly reflected light; n determines the sharpness of specularity peaks. As n increases, the specular peak gets sharper and sharper. Computer graphics research is constantly extending the frontiers of realistic and detailed reflectance, refractance, and illumination calculations [Blinn 1978; Phong 1975; Whitted 1980].

3.5.2 Surface Gradient

The reflectance functions described above are defined in terms of angles measured with respect to a local coordinate frame. For our development, it is more useful to relate the reflectivity function to surface gradients measured with respect to a viewer-oriented coordinate frame.

The concept of *gradient space*, which is defined in a viewer-oriented frame [Horn 1975], is extremely useful in understanding the recovery transformation algorithm for the surface normal. This gradient refers to the orientation of a physical surface, *not* to local intensities. It must not be confused with the *intensity* gradients discussed in Section 3.3 and elsewhere in this book.

Gradient space is a two-dimensional space of slants of scene surfaces. It measures a basic "intrinsic" (three-dimensional) property of surfaces. Consider the point-projection imaging geometry of Fig. 2.2, with the viewpoint at infinity (far from the scene relative to the scene dimensions). The image projection is then orthographic, nor perspective.

The surface gradient is defined for a surface expressed as $-z = f(x, y)$. The gradient is a vector (p, q), where

$$p = \frac{\partial(-z)}{\partial x} \qquad q = \frac{\partial(-z)}{\partial y} \tag{3.36}$$

Any plane in the image (such as the face plane of a polyhedral face) may be expressed in terms of its gradient. The general plane equation is

$$Ax + By + Cz + D = 0 \tag{3.37}$$

Thus

$$-z = \frac{A}{C}x + \frac{B}{C}y + \frac{D}{C} \tag{3.38}$$

and from (3.36) the gradient may be related to the plane equation:

$$-z = px + qy + K \tag{3.39}$$

Gradient space is thus the two-dimensional space of (p, q) vectors. The p and q axes are often considered to be superimposed on the x and y image plane coordinate axes. Then the (p, q) vector is "in the direction" of the surface slant of imaged surfaces. Any plane perpendicular to the viewing direction has a (p, q) vector of (0,0). Vectors on the q (or y) axis correspond to planes tilted about the x axis in an "upward" or "downward" ("$y$ward") direction (like the tilt of a dressing table

mirror). The direction arctan (q/p) is the direction of fastest change of surface depth $(-z)$ as x and y change. $(p^2 + q^2)^{1/2}$ is the rate of this change. For instance, a vertical plane "edge on" to the viewer has a (p, q) of $(I\infty, 0)$.

The *reflectance map* $R(p, q)$ represents this variation of perceived brightness with surface orientation. $R(p, q)$ gives scene radiance (Section 2.2.3) as a function of surface gradient (in our usual viewer-centered coordinate system). (Figure 3.27 showed the situation and defined some important angles.) $R(p, q)$ is usually shown as contours of constant scene radiance (Fig. 3.28). The following are a few useful cases.

In the case of a Lambertian surface with the source in the direction of the viewer $(i = e)$, the gradient space image looks like Fig. 3.28. Remember that Lambertian surfaces have constant intensity for constant illumination angle; these constant angles occur on the concentric circles of Fig. 3.28, since the direction of tilt does not affect the magnitude of the angle. The brightest surfaces are those illuminated from a normal direction—they are facing the viewer and so their gradients are $(0, 0)$.

Working this out from first principles, the incident angle and emittance angle are the same in this case, since the light is near the viewer. Both are the angle between the surface normal and the view vector. Looking at the x–y plane means a vector to the light source of $(0, 0, -1)$, and at a gradient point (p, q), the surface normal is $(p, q, -1)$. Also,

$$R = r_0 \cos i \tag{3.40}$$

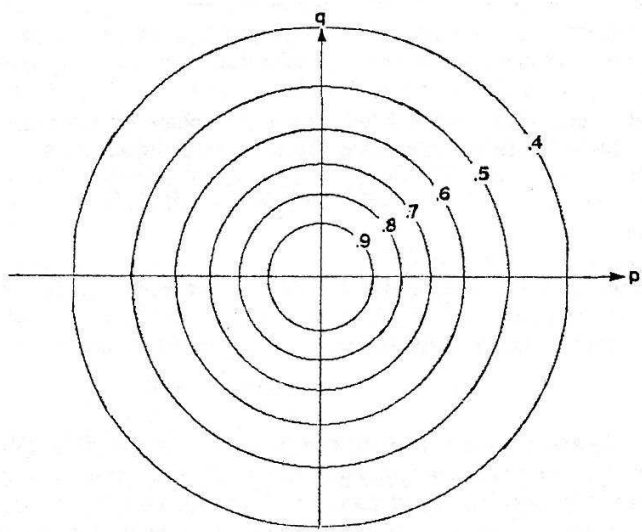


Fig. 3.28 Contours of constant radiance in gradient space for Lambertian surfaces; single light source near the viewpoint.

where r_o is a proportionality constant, and we conventionally use R to denote radiance in a viewer-centered frame. Let $\mathbf{n}_s$ and $\mathbf{n}$ be unit vectors in the source and surface normal directions. Since $\cos i = \mathbf{n}_s \cdot \mathbf{n}$

$$R = \frac{r_0}{(1 + p^2 + q^2)^{1/2}} \tag{3.41}$$

Thus $\cos(i)$ determines the image brightness, and so a plot of it is the gradient space image (Figs. 3.29 and 3.30).

For a more general light position, the mathematics is the same; if the light source is in the $(p_s, q_s, -1)$ direction, take the dot product of this direction and the surface normal.

$$R = r_o \mathbf{n} \cdot \mathbf{n}_s \tag{3.42}$$

Or, in other words,

$$R = \frac{r_o(p_s p + q_s q + 1)}{[(1 + p^2 + q^2)(1 + p_s^2 + q_s^2)]^{1/2}}$$

The phase angle g is constant throughout gradient space with orthographic projection (viewer distant from scene) and light source distant from scene.

Setting R constant to obtain contour lines gives a second-order equation, producing conic sections. In fact, the contours are produced by a set of cones of varying angles, whose axis is in the direction of the light source, intersecting a plane at unit distance from the origin. The resulting contours appear in Fig. 3.29. Here the dark line is the terminator, and represents all those planes that are edge-

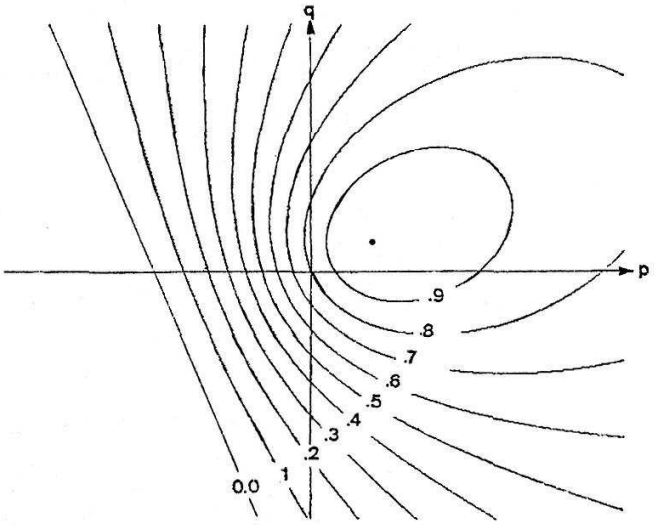


Fig. 3.29 Contours of constant radiance in gradient space. Lambertian surfaces; light not near viewpoint.

on to the light source; gradients on the back side of the terminator represent self-shadowed surfaces (facing away from the light). One intensity determines a contour and so gives a cone whose tangent planes all have that emittance. For a surface with specularity, contours of constant $I(i, e, g)$ could appear as in Fig. 3.30.

The point of specularity is between the matte component maximum brightness gradient and the origin. The brightest matte surface normal points at the light source and the origin points at the viewer. Pure specular reflection can occur if the vector tilts halfway toward the viewer maintaining the direction of tilt. Thus its gradient is on a line between the origin and the light-source direction gradient point.

3.5.3 Photometric Stereo

The reflectance equation (3.42) constrains the possible surface orientation to a locus on the reflectance map. Multiple light-source positions can determine the orientation uniquely [Woodham 1978]. Each separate light position gives a separate value for the intensity (proportional to radiance) at each point $f(\mathbf{x})$. If the surface reflectance r_o is unknown, three equations are needed to determine the reflectance together with the unit normal $\boldsymbol{n}$. If each source position vector is denoted by n_k, $k = 1, \ldots, 3$, the following equations result:

$$I_k(x, y) = r_o(\mathbf{n}_k \cdot \mathbf{n}), \qquad k = 1, \ldots, 3 \tag{3.43}$$

where I is normalized intensity. In matrix form

$$\mathbf{I} = r_o N \mathbf{n} \tag{3.44}$$

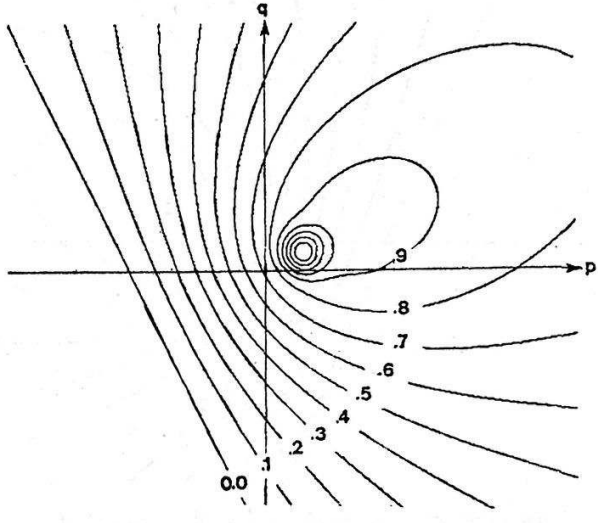


Fig. 3.30 Contours of constant radiance for a specular/matte surface.

where

$$\mathbf{I} = [I_1(x, y), I_2(x, y), I_3(x, y)]^T,$$

and

$$N = \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} \tag{3.45}$$

and $I = fc$ where c is the appropriate normalization constant. If c is not known, it can be regarded as being part of r_o without affecting the normal direction calculation. As long as the three source positions $\mathbf{n}_1$, $\mathbf{n}_2$, $\mathbf{n}_3$ are not coplanar, the matrix N will have an inverse. Then solve for r_o and $\mathbf{n}$ by using (3.44), first using the fact that $\mathbf{n}$ is a unit vector to derive

$$r_0 = |N^{-1}\mathbf{I}| \tag{3.46}$$

and then solving for $\boldsymbol{n}$ to obtain

$$\mathbf{n} = \frac{1}{r_o} N^{-1}\mathbf{I} \tag{3.47}$$

Examples of a particular solution are shown in Fig. 3.31. Of course, a prerequisite for using this method is that the surface point not be in shadows for any of the sources.

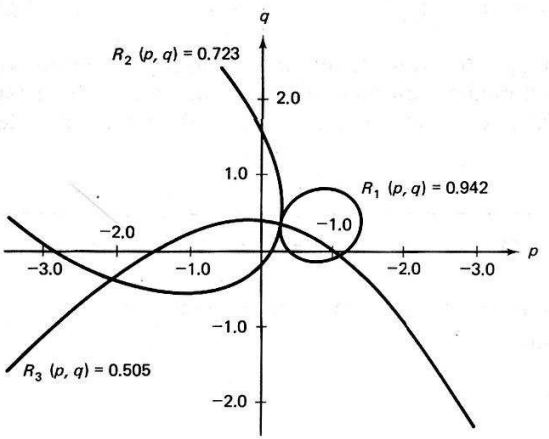


Fig. 3.31 A particular solution for photometric stereo.

3.5.4 Shape from Shading by Relaxation

Combining local information allows improved estimates for edges (Section 3.3.5) and for disparity (Section 3.4.2). In a similar manner local information can help in computing surface orientation [Ikeuchi 1980]. Basically, the reflectance equation

provides one constraint on the surface orientation and another is provided by the heuristic requirement that the surface be smooth.

Suppose there is an estimate of the surface normal at a point $(p(x, y), q(x, y))$. If the normal is not accurate, the reflectivity equation $I(x, y) = R(p, q)$ will not hold. Thus it seems reasonable to seek p and q that minimize $(I - R)^2$. The other requirement is that $p(x, y)$ and $q(x, y)$ be smooth, and this can be measured by their Laplacians $\nabla^2 p$ and $\nabla^2 q$. For a smooth curve both of these terms should be small. The goal is to minimize the error at a point,

$$E(x, y) = [I(x, y) - R(p, q)]^2 + \lambda[(\nabla^2 p)^2 + (\nabla^2 q)^2] \tag{3.48}$$

where the Lagrange multiplier λ [Russell 1976] incorporates the smoothness constraint. Differentiating $E(x, y)$ with respect to p and q and approximating derivatives numerically gives the following equations for $p(x, y)$ and $q(x, y)$:

$$p(x, y) = p_{av}(x, y) + T(x, y, p, q)\frac{\partial R}{\partial q} \tag{3.49}$$

$$q(x, y) = q_{av}(x, y) + T(x, y, p, q)\frac{\partial R}{\partial q} \tag{3.50}$$

where

$$T(x, y, p, q) = (1/\lambda)[I(x, y) - R(p, q)]$$

using

$$p_{av}(x, y) = \frac{1}{4}[p(x + 1, y) + p(x - 1, y) + p(x, y + 1) + p(x,y - 1)] \tag{3.51}$$

and a similar expression for q_{av}. Now Eqs. (3.49) and (3.50) lend themselves to solution by the Gauss-Seidel method: calculate the left-hand sides with an estimate for p and q and use them to derive a new estimate for the right-hand sides. More formally,

---

Algorithm 3.3: Shape from Shading [Ikeuchi 1980].

Step 0. k = 0. Pick an initial $p^0(x, y)$ and $q^0(x, y)$ near boundaries.

Step 1. k = k + 1; compute

$$p^k = p_{av}^{k-1} + T\frac{\partial R}{\partial p}$$

$$q^k = q_{av}^{k-1} + T\frac{\partial R}{\partial p}$$

Step 2. If the sum of all the E's is sufficiently small, stop. Else, go to step 1.

A loose end in this algorithm is that boundary conditions must be specified. These are values of p and q determined a priori that remain constant throughout each iteration. The simplest place to specify a surface gradient is at an occluding contour (see Fig. 3.32) where the gradient is nearly 90° to the line of sight. Unfortunately, p and q are infinite at these points. Ikeuchi's elegant solution to this is to use a different coordinate system for gradient space, that of a Gaussian sphere (Appendix 1). In this system, the surface normal is described relative to where it intersects the sphere if the tail of the normal is at the sphere's origin. This is the point at which a plane perpendicular to the normal would touch the sphere if translated toward it (Fig. 3.32b).

In this system the radiance may be described in terms of the spherical coordinates θ, ϕ. For a Lambertian surface

$$R(\theta, \phi) = \cos\theta \cos\theta_s + \sin\theta \sin\theta_s \cos(\phi - \phi_s) \tag{3.52}$$

At an occluding contour $\phi = \pi/2$ and θ is given by $\tan^{-1}(\partial y / \partial x)$, where the derivatives are calculated at the occluding contour (Fig. 3.32c).

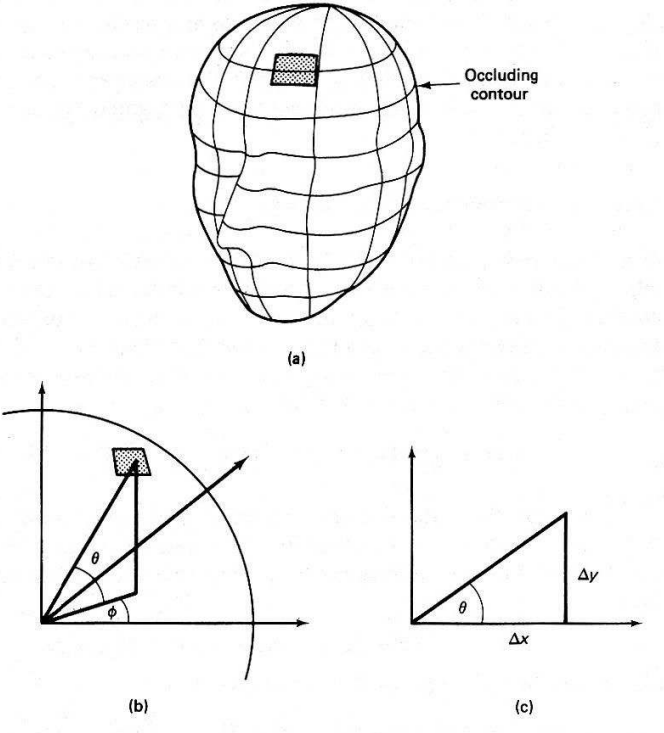


Fig. 3.32 (a) Occluding contour. (b) Gaussian sphere. (c) Calculating θ from occluding contour.

To use the (θ, ϕ) formulation instead of the (p, q) formulation is an easy matter. Simply substitute θ for p and ϕ for q in all instances of the formula in Algorithm 3.3.

3.6 OPTICAL FLOW

Much of the work on computer analysis of visual motion assumes a stationary observer and a stationary background. In contrast, biological systems typically move relatively continuously through the world, and the image projected on their retinas varies essentially continuously while they move. Human beings perceive smooth continuous motion as such.

Although biological visual systems are discrete, this quantization is so fine that it is capable of producing essentially continuous outputs. These outputs can mirror the continuous flow of the imaged world across the retina. Such continuous information is called *optical flow*. Postulating optical flow as an input to a perceptual system leads to interesting methods of motion perception.

The optical flow, or instantaneous velocity field, assigns to every point on the visual field a two-dimensional "retinal velocity" at which it is moving across the visual field. This section describes how approximations to instantaneous flow may be computed from the usual input situation in a sequence of discrete images. Methods of using optical flow to compute the observer's motion, a relative depth map, surface normals of his or her surroundings, and other useful information are given in Chapter 7.

3.6.1 The Fundamental Flow Constraint

One of the important features of optical flow is that it can be calculated simply, using local information. One way of doing this is to model the motion image by a continuous variation of image intensity as a function of position and time, then expand the intensity function $f(x, y, t)$ in a Taylor series.

$$f(x + dx, y + dy, t + dt) = \tag{3.53}$$
$$f(x, y, t) + \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy + \frac{\partial f}{\partial t} dt + \text{higher-order terms}$$

As usual, the higher-order terms are henceforth ignored. The crucial observation to be exploited is the following: If indeed the image at some time $t + dt$ is the result of the original image at time t being moved translationally by dx and dy, then in fact

$$f(x + dx, y + dy, t + dt) = f(x, y, t) \tag{3.54}$$

Consequently, from Eqs. (3.53) and (3.54),

$$-\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} \tag{3.55}$$

Now $\frac{\partial f}{\partial t}$, $\frac{\partial f}{\partial x}$, and $\frac{\partial f}{\partial y}$ are all measurable quantities, and $\frac{dx}{dt}$ and $\frac{dy}{dt}$ are estimates of what we are looking for—the velocity in the x and y directions. Writing

$$\frac{dx}{dt} = u, \qquad \frac{dy}{dt} = v$$

gives

$$-\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}u + \frac{\partial f}{\partial y}v \tag{3.56}$$

or equivalently,

$$-\frac{\partial f}{\partial t} = \nabla f \cdot \mathbf{u} \tag{3.57}$$

where ∇f is the spatial gradient of the image and $\mathbf{u} = (u, v)$ the velocity.

The implications of (3.57) are interesting. Consider a fixed camera with a scene moving past it. The equations say that the *time* rate of change in intensity of a point in the image is (to first order) explained as the *spatial* rate of change in the intensity of the scene multiplied by the *velocity* that points of the scene move past the camera.

This equation also indicates that the velocity (u, v) must lie on a line perpendicular to the vector (f_x, f_y) where f_x and f_y are the partial derivatives with respect to x and y, respectively (Fig. 3.33). In fact, if the partial derivatives are very accurate the magnitude component of the velocity in the direction (f_x, f_y) is (from 3.57):

$$\frac{-f_t}{[(f_x^2 + f_y^2)]^{1/2}}$$

3.6.2 Calculating Optical Flow by Relaxation

Equation (3.57) constrains the velocity but does not determine it uniquely. The development of Section 3.5.4 motivates the search for a solution that satisfies Eq.

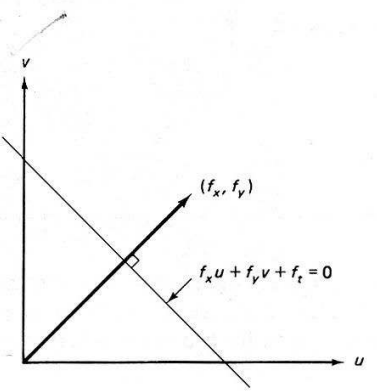


Fig. 3.33 Relation between (u, v) and (f_x, f_y).

(3.57) as closely as possible and also is locally smooth [Horn and Schunck 1980]. In this case as well, the Laplacians of the two velocity components, $\nabla^2 u$ and $\nabla^2 v$, can measure local smoothness.

Again using the method of Lagrange multipliers, minimize the flow error

$$E^2(x, y) = (f_x u + f_y v + f_t)^2 + \lambda^2[(\nabla^2 u)^2 + (\nabla^2 v)^2] \tag{3.58}$$

Differentiating this equation with respect to u and v provides equations for the change in error with respect to u and v, which must be zero for a minimum. Writing $\nabla^2 u$ as $u - u_{av}$ and $\nabla^2 v$ as $v - v_{av}$, these equations are

$$(\lambda^2 + f_x^2)u + f_x f_y v = \lambda^2 u_{av} - f_x f_t \tag{3.59}$$

$$f_x f_y u + (\lambda^2 + f_y^2)v = \lambda^2 v_{av} - f_y f_t \tag{3.60}$$

These equations may be solved for u and v, yielding

$$u = u_{av} - f_x \frac{P}{D} \tag{3.61}$$

$$v = v_{av} - f_y \frac{P}{D} \tag{3.62}$$

where

$$P = f_x u_{av} + f_y v_{av} + f_t$$

$$D = \lambda^2 + f_x^2 + f_y^2$$

To turn this into an iterative equation for solving $u(x, y)$ and $v(x, y)$, again use the Gauss-Seidel method.

Algorithm 3.4: Optical Flow [Horn and Schunck 1980].

$k = 0$.
Initialize all u^k and v^k to zero.
Until some error measure is satisfied, do

$$u^k = u_{av}^{k-1} - f_x \frac{P}{D}$$

$$v^k = v_{av}^{k-1} - f_y \frac{P}{D}$$

As Horn and Schunck demonstrate, this method derives the flow for two time frames, but it can be improved by using several time frames and using the final solution after one iteration at one time for the initial solution at the following time frame. That is:

Algorithm 3.5: Multiframe Optical Flow.

$t = 0$.
Initialize all $u(x, y, 0)$, $v(x, y, 0)$
for $t = 1$ *until* maxframes *do*

$$u(x, y, t) = u_{av}(x, y, t-1) - f_x \frac{P}{D}$$

$$v(x, y, t) = v_{av}(x, y, t-1) - f_y \frac{P}{D}$$

The results of using synthetic data from a rotating checkered sphere are shown in Fig. 3.34.

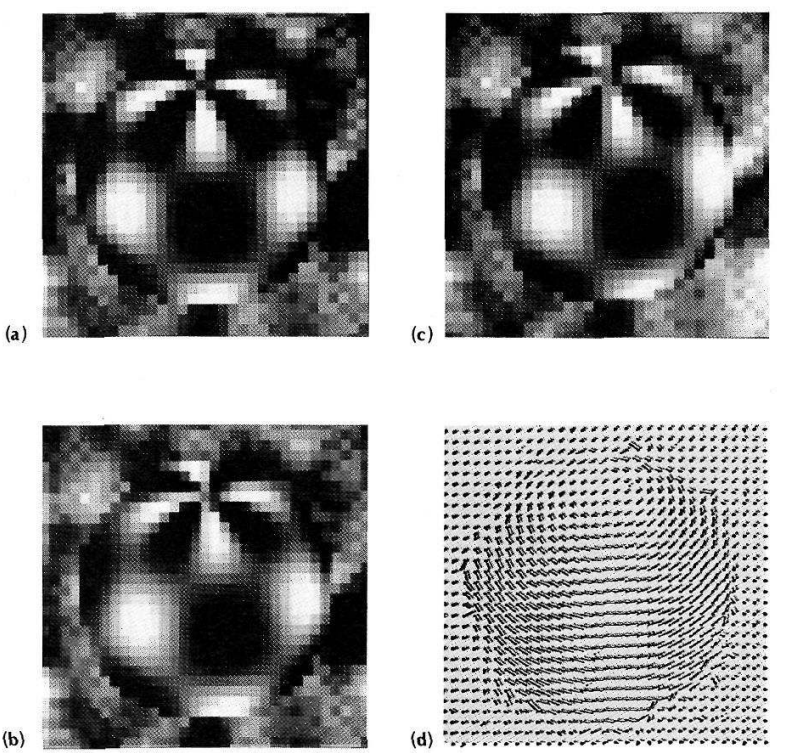


Fig. 3.34 Optical flow results. (a), (b) and (c) are three frames from the rotating sphere, (d) is the derived three-dimensional flow after 32 such time frames.

3.7 RESOLUTION PYRAMIDS

What is the best spatial resolution for an image? The sampling theorem states that the maximum spatial frequency in the image data must be less than half the sampling frequency in order that the sampled image represent the original unambiguously. However, the sampling theorem is not a good predictor of how easily objects can be recognized by computer programs. Often objects can be more easily recognized in images that have a very low sampling rate. There are two reasons for this. First, the computations are fewer because of the reduction in dimensionality. Second, confusing detail present in the high-resolution versions of the images may not appear at the reduced resolution. But even though some objects are more easily found at low resolutions, usually an object description needs detail only revealed at the higher resolutions. This leads naturally to the notion of a *pyramidal* image data structure in which the search for objects is begun at a low resolution, and refined at ever-increasing resolutions until one reaches the highest resolution of interest. Figure 3.35 shows the correspondence between pixels for the pyramidal structure.

In the next three sections, pyramids are applied to gray-level images and edge images. Pyramids, however, are a very general tool and can be used to represent any image at varying levels of detail.

3.7.1 Gray-level Consolidation

In some applications, redigitizing the image with a different sampling rate is a way to reduce the number of samples. However, most digitizer parameters are difficult to change, so that often computational means of reduction are needed. A straightforward method is to partition the digitized image into nonoverlapping

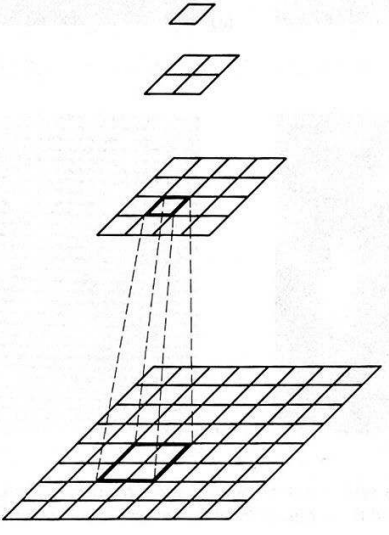


Fig. 3.35 Pyramidal image structure.

neighborhoods of equal size and shape and to replace each of those neighborhoods by the average pixel densities in that neighborhood. This operation is *consolidation.* For an $n \times n$ neighborhood, consolidation is equivalent to averaging the original image over the neighborhood followed by sampling at intervals n units apart.

Consolidation tends to offset the aliasing that would be introduced by sampling the sensed data at a reduced rate. This is due to the effects of the averaging step in the consolidation process. For the one-dimensional case where

$$f'(x) = \frac{1}{2}[f(x) + f(x + \Delta)] \tag{3.63}$$

the corresponding Fourier transform [Steiglitz 1974] is

$$H(u) = \frac{1}{2}\left[1 + e^{-j2\pi\Delta}\right]F(u) \tag{3.64}$$

which has magnitude $|H(u)| = \cos[\pi(u/u_o)]$ and phase $-\pi(u/u_o)$. The sampling frequency $u_o = 1/\Delta$ where Δ is the spacing between samples. Thus the averaging step has the effect of attenuating the higher frequencies of $F(u)$ as shown in Fig. 3.36. Since the higher frequencies are involved in aliasing, attenuating these frequencies reduces the aliasing effects.

3.7.2 Pyramidal Structures in Correlation

With correlation matching, the use of multiple resolution techniques can sometimes provide significant functional and computational advantages [Moravec 1977]. Binary search correlation uses pyramids of the input image and reference

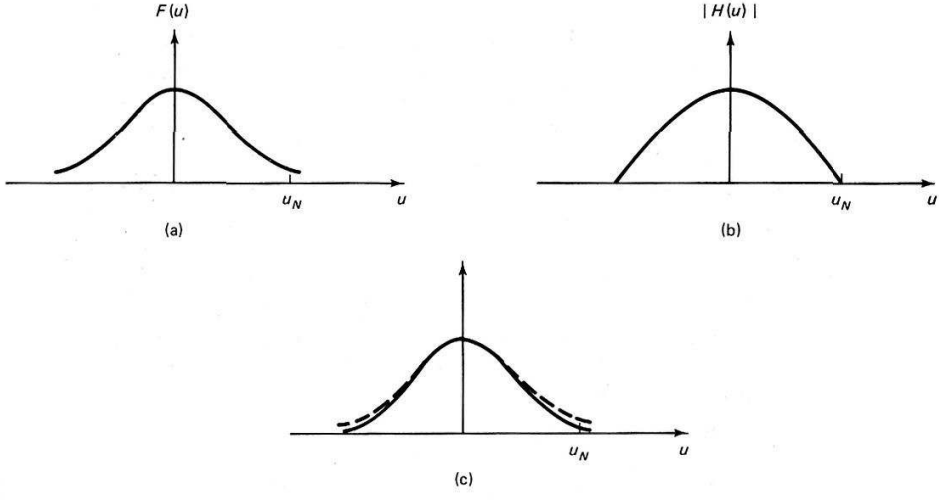


Fig. 3.36 Consolidation effects viewed in the spatial frequency domain. (a) Original transform. (b) Transform of averaging operator. (c) Transform of averaged image.

patterns. The algorithm partakes of the computational efficiency of binary (as opposed to linear) search [Knuth 1973]. Further, the low-resolution correlation operations at high levels in the pyramid ensure that the earlier correlations are on gross image features rather than details.

In binary search correlation a feature to be located is at some unknown location in the input image. The reference version of the feature originates in another image, the reference image. The feature in the reference image is contained in a window of $n \times n$ pixels. The task of the correlator is to find an $n \times n$ window in the input image that best matches the reference image window containing the feature. The details of the correlation processes are given in the following algorithm.

Algorithm 3.6: Binary Search Correlation Control Algorithm

Definitions

OrigReference: an $N \times N$ image containing a feature centered at (FeatureX, FeatureY).

OrigInput: an $M \times M$ array in which an instance of the Feature is to be located. For simplicity, assume that it is at the same resolution as OrigReference.

n: a window size; an $n \times n$ window in OrigReference is large enough to contain the Feature.

Window: an $n \times n$ array containing a varying-resolution subimage of OrigReference centered on the Feature.

Input: a $2n \times 2n$ array containing a varying-resolution subimage of OrigInput, centered on the best match for the Feature.

Reference: a temporary array.

Algorithm

1. Input := Consolidate OrigInput by a factor of $2n/M$ to size $2n \times 2n$.
2. Reference := Consolidate OrigReference by the same factor $2n/M$ to size $2nN/M \times 2nN/M$. This consolidation takes the Feature to a new (Feature$X$, Feature$Y$).
3. Window := $n \times n$ window from Reference centered on the new (Feature$X$, Feature$Y$).
4. Calculate the match metric of the window at the $(n + 1)^2$ locations in Input at which it is wholly contained. Say that the best match occurs at (BestMatch$X$, BestMatch$Y$) in Input.

5. Input := $n \times n$ window from Input centered at (BestMatch$X$, BestMatch$Y$), enlarged by a factor of 2.
6. Reference := Reference enlarged by a factor of 2. This takes Feature to a new (Feature$X$, Feature$Y$).
7. *Go to* 3.

Through time, the algorithm uses a reference image for matching that is always centered on the feature to be matched, but that homes in on the feature by being increased in resolution and thus reduced in linear image coverage by a factor of 2 each time. In the input image, a similar homing-in is going on, but the search area is usually twice the linear dimension of the reference window. Further, the center of the search area varies in the input image as the improved resolution refines the point of best match.

Binary search correlation is for matching features with context. The template at low resolution possibly corresponds to much of the area around the feature, while the feature may be so small in the initial consolidated images as to be invisible. The coarse-to-fine strategy is perfect for such conditions, since it allows gross features to be matched first and to guide the later high-resolution search for best match. Such matching with context is less useful for locating several instances of a shape dotted at random around an image.

3.7.3 Pyramidal Structures in Edge Detection

As an example of the use of pyramidal structures in processing, consider the use of such structures in edge detection. This application, after [Tanimoto and Pavlidis 1975], uses two pyramids, one to store the image and another to store the image edges. The idea of the algorithm is that a neighborhood in the low-resolution image where the gray-level values are the same is taken to imply that in fact there is no gray-level change (edge) in the neighborhood. Of course, the low-resolution levels in the pyramid tend to blur the image and thus attenuate the gray-level changes that denote edges. Thus the starting level in the pyramid must be picked judiciously to ensure that the important edges are detected.

Algorithm 3.7: Hierarchical Edge Detection

```
recursive procedure refine (k, x, y)
  begin
  if k < MaxLevel then
    for dx = 0 until 1 do
    for dy = 0 until 1 do
      if EdgeOp (k, x + dx, y + dy) > Threshold(x)
      then refine (k + 1, x + dx, y + dy)
  end;
```

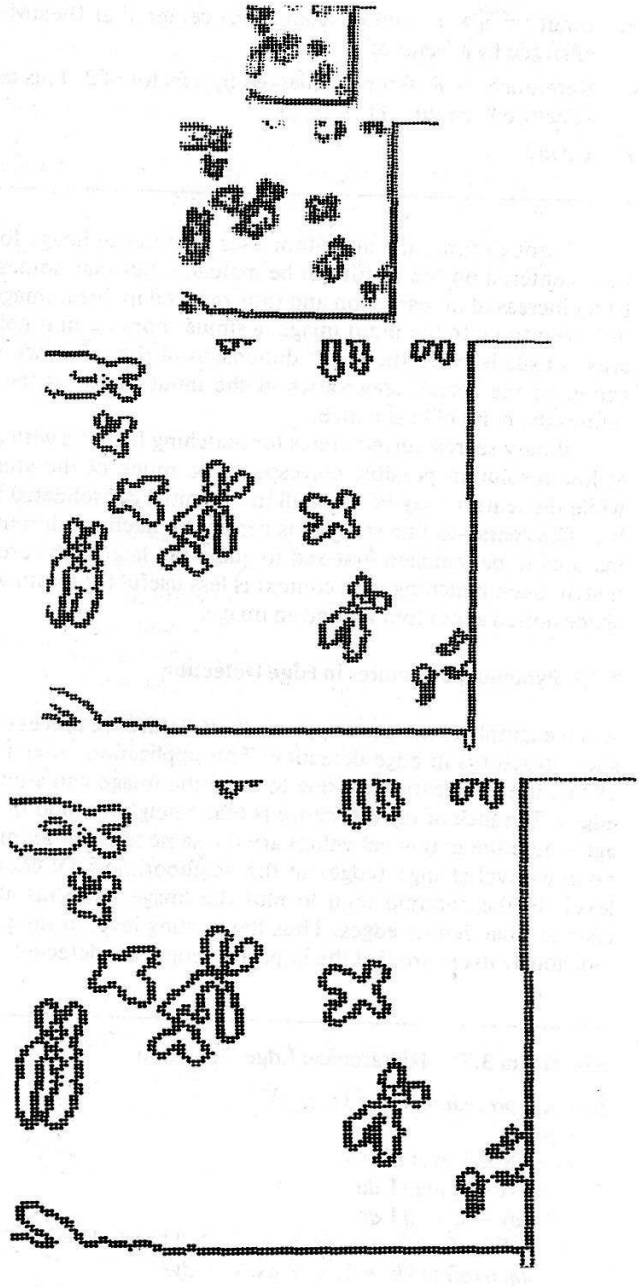


Fig. 3.37 Pyramidal edge detection.

```
procedure FindEdges:
  begin
  comment apply operator to every pixel in the
      starting level s, refining where necessary;
    for x := 0 until 2^S − 1 do
    for y := 0 until 2^S − 1 do
      if EdgeOp (s, x, y) > Threshold(s)
      then refine (s, x, y);
  end;
```

Figure 3.37 shows Tanimoto's results for a chromosome image. The table inset shows the computational advantage in terms of the calls to the edge operator as a function of the starting level s.

Similar kinds of edge detection strategies based on pyramids have been pursued by [Levine 1978; Hanson and Riseman 1978]. The latter effort is a little different in that processing within the pyramid is bidirectional; information from edges detected at a high-resolution level is projected to low-resolution levels of the pyramid.

EXERCISES

3.1 Derive an analytical expression for the response of the Sobel operator to a vertical step edge as a function of the distance of the edge to the center of the operator.

3.2 Use the formulas of Eqs. (3.31) to derive the digital template function for g_1 in a 5^3 pixel domain.

3.3 Specify a version of Algorithm 3.1 that uses the gradient edge operator instead of the "crack" edge operator.

3.4 In photometric stereo, three or more light source positions are used to determine a surface orientation. The dual of this problem uses surface orientations to determine light source position. What is the usefulness of the latter formulation? In particular, how does it relate to Algorithm 3.3?

3.5 Using any one of Algorithms 3.1 through 3.4 as an example, show how it could be modified to use pyramidal data structures.

3.6 Write a reflectance function to capture the "grazing incidence" phenomenon—surfaces become more mirror-like at small angles of incidence (and reflectance).

3.7 Equations 3.49 and 3.50 were derived by minimizing the local error. Show how these equations are modified when total error [i.e., $\sum_{x,y} E(x, y)$] is minimized.

REFERENCES

ABDOU, I. E. "Quantitative methods of edge detection." USCIPI Report 830, Image Processing Institute, Univ. Southern California, July 1978.

AKATSUKA, T., T. ISOBE, and O. TAKATANI. "Feature extraction of stomach radiograph." *Proc.*, 2nd IJCPR, August 1974, 324-328.

ANDREWS, H. C. and B. R. HUNT. *Digital Image Restoration.* Englewood Cliffs, NJ: Prentice-Hall, 1977.

ATTNEAVE, F. "Some informational aspects of visual perception." *Psychological Review 61*, 1954.

BARROW, H. G. and J. M. TENENBAUM. "Computational Vision." *Proc. IEEE 69*, 5, May 1981, 572-595

BARROW, H. G. and J. M. TENENBAUM. "Recovering intrinsic scene characteristics from images." Technical Note 157, AI Center, SRI International, April 1978.

BINFORD, T. O. "Visual perception by computer." *Proc.*, IEEE Conf. on Systems and Control, Miami, December 1971.

BLINN, J. E. "Computer display of curved surfaces." Ph.D. dissertation, Computer Science Dept., Univ. Utah, 1978.

FREI, W. and C. C. CHEN. "Fast boundary detection: a generalization and a new algorithm." *IEEE Trans. Computers 26*, 2, October 1977, 988-998.

GONZALEZ, R. C. and P. WINTZ. *Digital Image Processing.* Reading, MA: Addison-Wesley, 1977.

GRIFFITH, A. K. "Edge detection in simple scenes using a priori information." *IEEE Trans. Computers 22*, 4, April 1973.

HANSON, A. R. and E. M. RISEMAN (Eds.). *Computer Vision Systems (CVS).* New York: Academic Press, 1978.

HORN, B. K. P. "Determining lightness from an image." *CGIP 3*, 4, December 1974, 277-299.

HORN, B. K. P. "Shape from shading." In *PCV*, 1975.

HORN, B. K. P. and B. G. SCHUNCK. "Determining optical flow." AI Memo 572, AI Lab, MIT, April 1980.

HORN, B. K. P. and R. W. SJOBERG. "Calculating the reflectance map." *Proc.*, DARPA IU Workshop, November 1978, 115-126.

HUBEL, D. H. and T. N. WIESEL. "Brain mechanisms of vision." *Scientific American*, September 1979, 150-162.

HUECKEL, M. "An operator which locates edges in digitized pictures." *J. ACM 18*, 1, January 1971, 113-125.

HUECKEL, M. "A local visual operator which recognizes edges and lines." *J. ACM 20*, 4, October 1973, 634-647.

IKEUCHI, K. "Numerical shape from shading and occluding contours in a single view." AI Memo 566, AI Lab, MIT, revised February 1980.

KIRSCH, R. A. "Computer determination of the constituent structure of biological images." *Computers and Biomedical Research 4*, 3, June 1971, 315-328.

KNUTH, D. E. *The Art of Computer Programming.* Reading, MA: Addison-Wesley, 1973.

LEVINE, M. D. "A knowledge-based computer vision system." In *CVS*, 1978.

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

MARR, D. and T. POGGIO. "Cooperative computation of stereo disparity." *Science 194*, 1976, 283-287.

MARR, D. and T. POGGIO. "A theory of human stereo vision." AI Memo 451, AI Lab, MIT, November 1977.

MERO, L. and Z. VASSY. "A simplified and fast version of the Hueckel operator for finding optimal edges in pictures." *Proc.*, 4th IJCAI, September 1975, 650-655.

MORAVEC, H. P. "Towards automatic visual obstacle avoidance." *Proc.*, 5th IJCAI, August 1977, 584.

NEVATIA, R. "Evaluation of a simplified Hueckel edge-line detector." Note, *CGIP 6*, 6, December 1977, 582-588.

PHONG, B-T. "Illumination for computer generated pictures." *Commun. ACM 18*, 6, June 1975, 311-317.

PINGLE, K. K. and J. M. TENENBAUM. "An accommodating edge follower." *Proc.*, 2nd IJCAI, September 1971, 1-7.

PRAGER, J. M. "Extracting and labeling boundary segments in natural scenes." *IEEE Trans. PAMI 2*, 1, January 1980, 16-27.

PRATT, W. K. *Digital Image Processing.* New York: Wiley-Interscience, 1978.

PREWITT, J. M. S. "Object enhancement and extraction." In *Picture Processing and Psychopictorics*, B. S. Lipkin and A. Rosenfeld (Eds.). New York: Academic Press, 1970.

QUAM, L. and M. J. HANNAH. "Stanford automated photogrammetry research." AIM-254, Stanford AI Lab, November 1974.

ROBERTS, L. G. "Machine perception of three-dimensional solids." In *Optical and Electro-optical Information Processing*, J. P. Tippett et al. (Eds.). Cambridge, MA: MIT Press, 1965.

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

ROSENFELD, A., R. A. HUMMEL, and S. W. ZUCKER. "Scene labelling by relaxation operations." *IEEE Trans. SMC 6*, 1976, 430.

RUSSELL, D. L. (Ed.). *Calculus of Variations and Control Theory.* New York: Academic Press, 1976.

SHAPIRA, R. "A technique for the reconstruction of a straight-edge, wire-frame object from two or more central projections." *CGIP 3*, 4, December 1974, 318-326.

SHIRAI, V. "Analyzing intensity arrays using knowledge about scenes." In *PCV*, 1975.

STEIGLITZ, K. *An Introduction to Discrete Systems.* New York: Wiley, 1974.

STOCKHAM, T. J., Jr. "Image processing in the context of a visual model." *Proc. IEEE 60*, 7, July 1972, 828-842.

TANIMOTO, S. and T. PAVLIDIS. "A hierarchical data structure for picture processing." *CGIP 4*, 2, June 1975, 104-119.

TRETIAK, O. J. "A parameteric model for edge detection." *Proc.*, 3rd COMPSAC, November 1979, 884-887.

TURNER, K. J. "Computer perception of curved objects using a television camera." Ph.D. dissertation, Univ. Edinburgh, 1974.

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

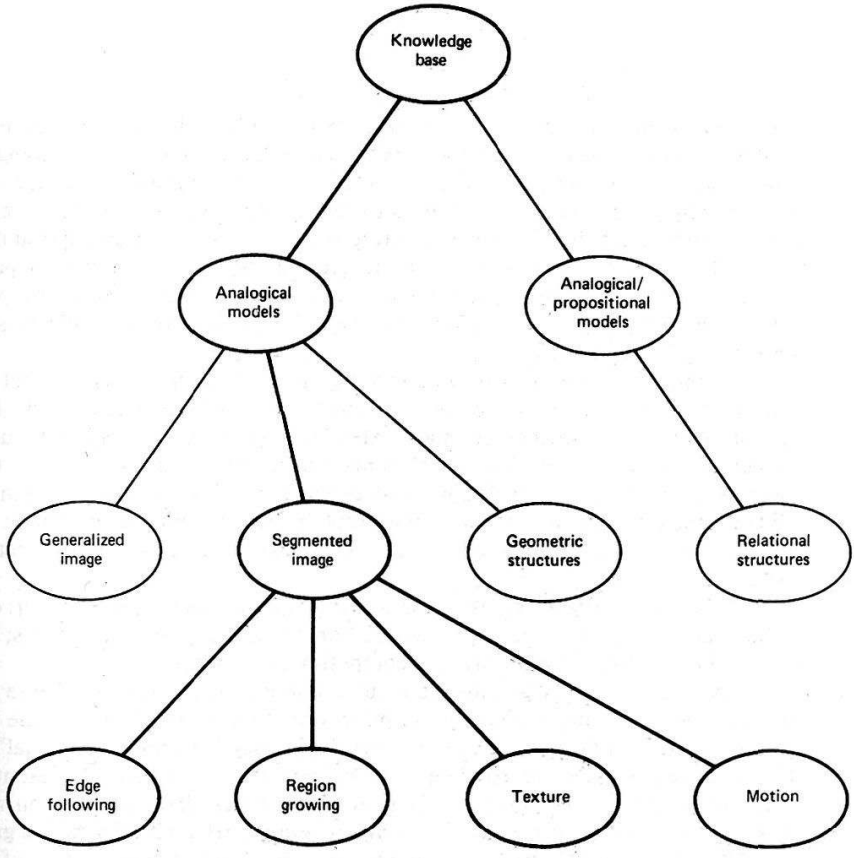
WHITTED, T. "An improved illumination model for shaded display." *Comm. ACM 23*, 6, June 1980, 343-349.

WOODHAM, R. J. "Photometric stereo: A reflectance map technique for determining surface orientation from image intensity." *Proc.*, 22nd International Symp., Society of Photo-optical Instrumentation Engineers, San Diego, CA, August 1978, 136-143.

ZUCKER, S. W. and R. A. HUMMEL. "An optimal three-dimensional edge operator." Report 79-10, McGill Univ., April 1979.

ZUCKER, S. W., R. A. HUMMEL, and A. ROSENFELD. "An application of relaxation labeling to line and curve enhancement." *IEEE Trans. Computers 26*, 1977. Apral N°4 pp 394

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.

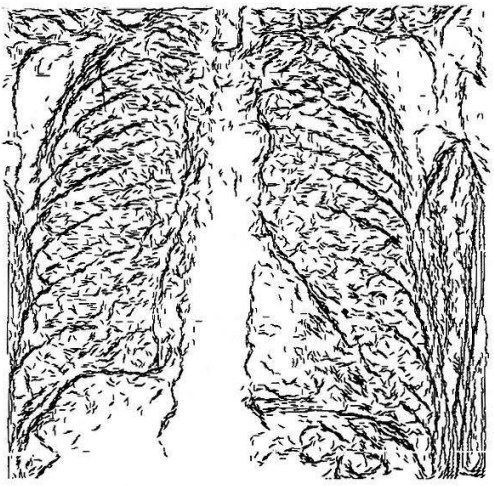


Fig. 4.1 Edge elements in a chest radiograph.

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.

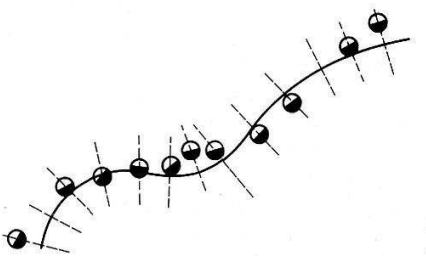


Fig. 4.2 Search orientations from an approximate boundary location.

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

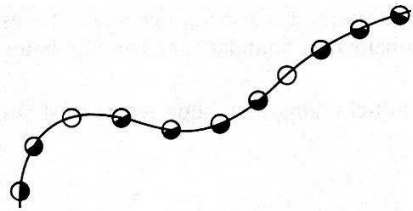


Fig. 4.3 A template for edge-operator application.

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].

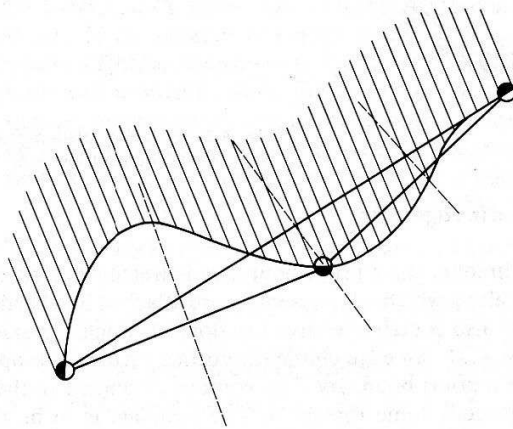


Fig. 4.4 Divide and conquer technique.

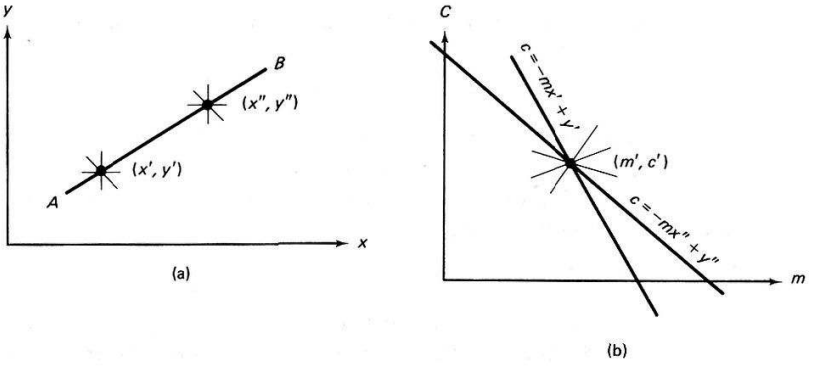


Fig. 4.5 A line (a) in image space; (b) in parameter space.

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

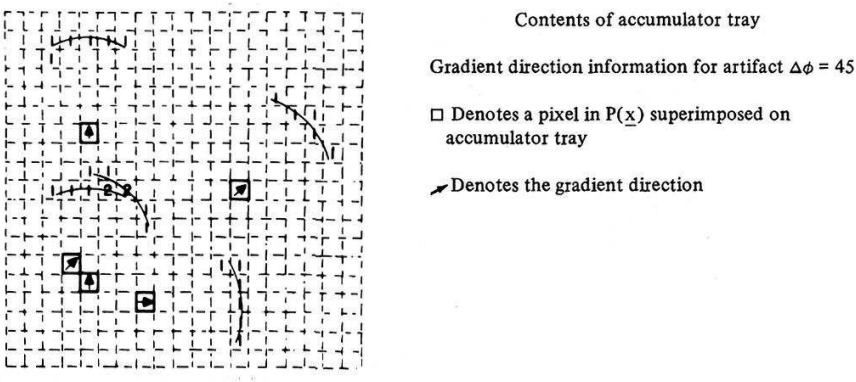


Fig 4.6 Reduction in computation with gradient information

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

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

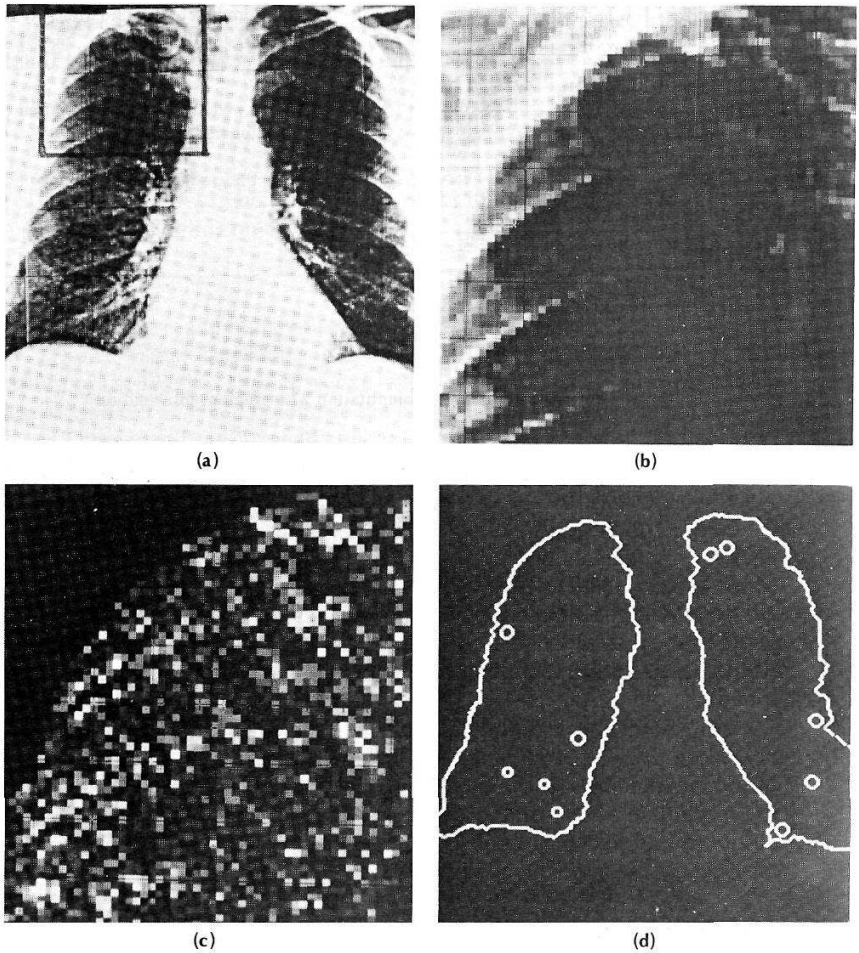


Fig. 4.7 Using the Hough technique for circular shapes. (a) Radiograph. (b) Window. (c) Accumulator array for $r = 3$. (d) Results of maxima detection.

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.

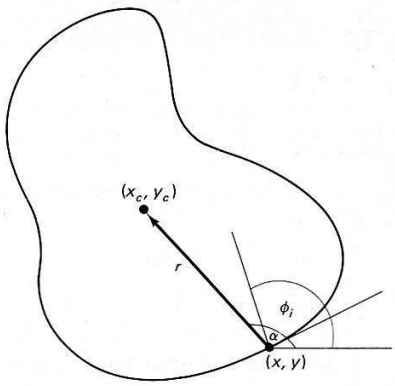


Fig. 4.8 Geometry used to form the R-Table.

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 ϕ, 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

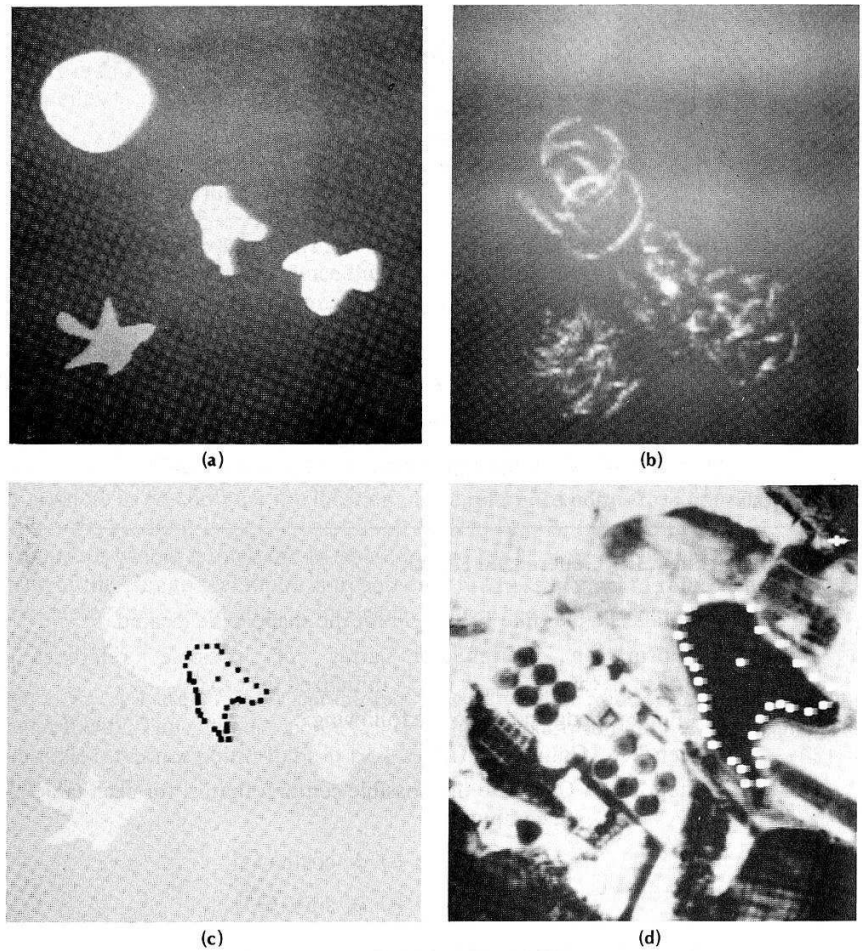


Fig. 4.9 Applying the Generalized Hough technique. (a) Synthetic image. (b) Hough Transform $A(x_c, y_c)$ for middle shape. (c) Detected shape. (d) Same shape in an aerial image setting.

$A(x_c, y_c)$ 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 ϕ do

for each S and θ

$$x_c := x + r(\phi) S \cos[\alpha(\phi) + \theta]$$

$$y_c := y + r(\phi) S \sin[\alpha(\phi) + \theta]$$

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, $g(\mathbf{x}_i)$

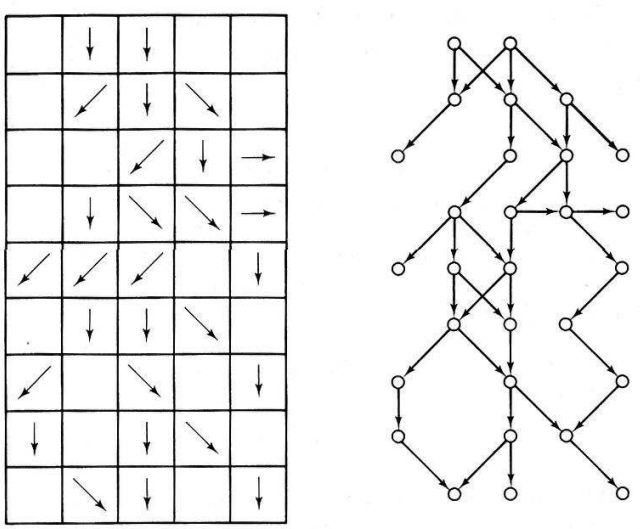


Fig. 4.10 Interpreting a gradient image as a graph (see text).

$> T$, $g(\mathbf{x}_j) > T$, 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
>
> $$k(\mathbf{x}_i, \mathbf{x}_j) \geq h^*(\mathbf{x}_i) - h^*(\mathbf{x}_j)$$

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 $\mathbf{x}$ 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 $d(\mathbf{x}_i, \mathbf{x}_{goal})$, 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

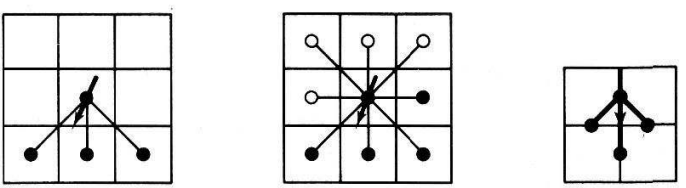


Fig. 4.11 Successor conventions in heuristic search (see text).

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-

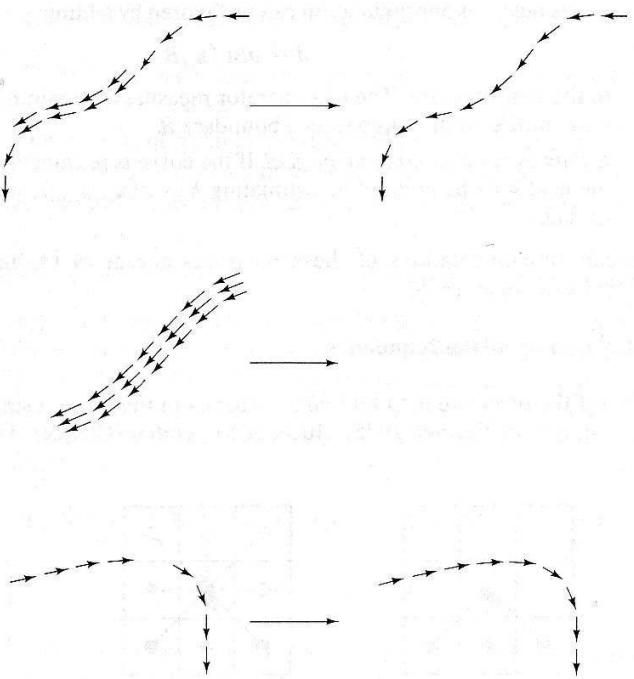


Fig. 4.12 Operations in the creation of prime streaks.

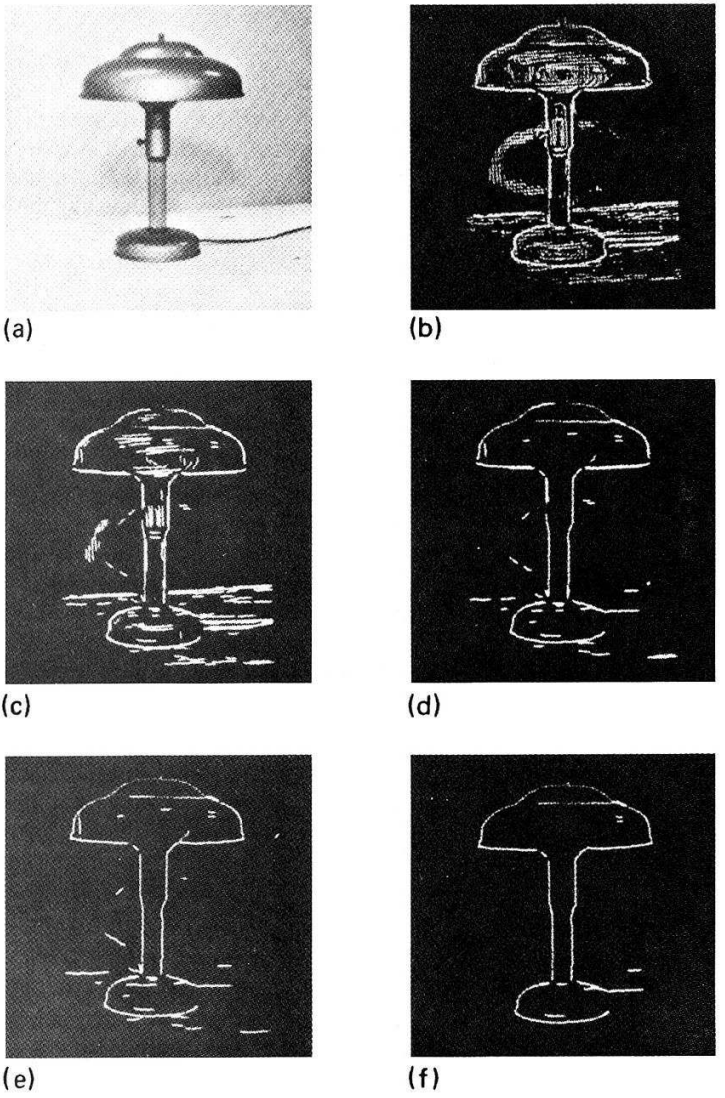


Fig. 4.13 Ramer's results.

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

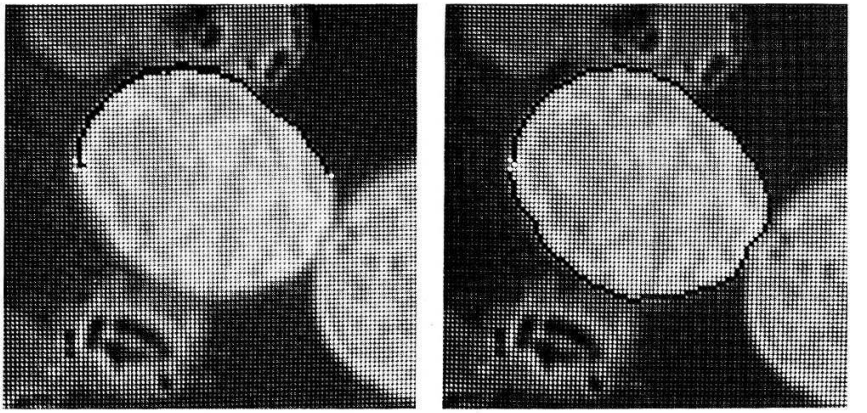
In this elegant idea [Lester 1978], only the maximum-cost arc of each path is kept as an estimate of g. This is like finding a mountain pass at minimum altitude. The advantage is that g does not build up continuously with depth in the search tree, so that good paths may be followed for a long time. This 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.



(a) (b)

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 $\mathbf{x}$ 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 ap-

plied 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

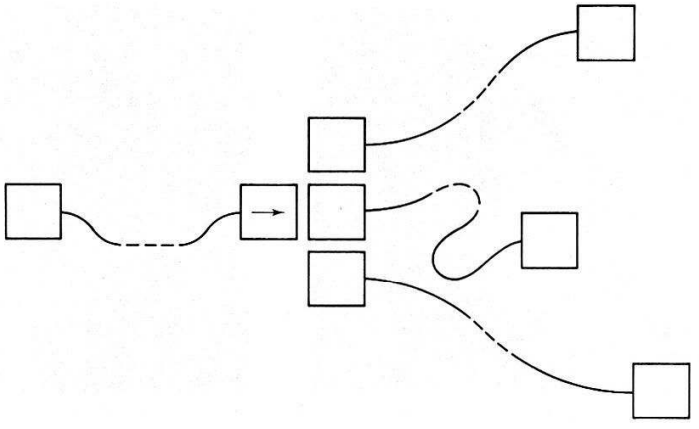


Fig. 4.15 DP optimization for boundary tracing.

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

$$f_l(\mathbf{x}_{k+1}) = \max_{x_k}[s(\mathbf{x}_k) + \alpha q(\mathbf{x}_k, t(\mathbf{x}_{k+1})) + f_{l-1}(\mathbf{x}_k)] \tag{4.22}$$

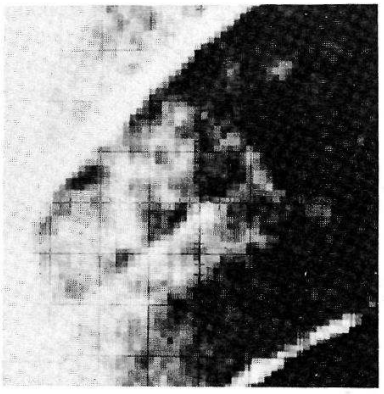
where the curve length n is related to α by a building sequence $n(l)$ such that $n(1) = 1$, $n(L) = N$, and $n(l) - n(l-1)$ is a member of $\{n(k) | k = 1, \ldots, l-1\}$. Also, $t(\mathbf{x}_k)$ is a function that extracts the tail pixel of the curve headed by $\mathbf{x}_k$. Further details may be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

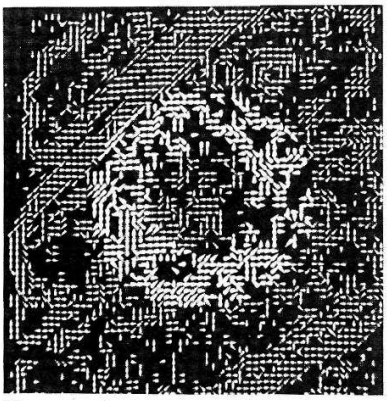
Another application of dynamic programming may be found in the pseudo-parallel road finder of Barrow [Barrow 1976].

4.5.3 Lower Resolution Evaluation Functions

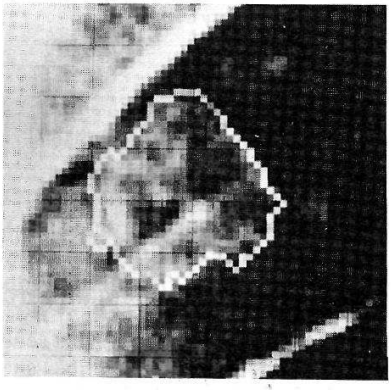
In the dynamic programming formulation just developed, the components $g(\mathbf{x}_k)$ and $q(\mathbf{x}_k, \mathbf{x}_{k+1})$ in the evaluation function are very localized; the variables $\mathbf{x}$ for successive s and q are in fact constrained to be grid neighbors. This need not be the case: The $\mathbf{x}$ 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



(a)



(b)



(c)

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:

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

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

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:

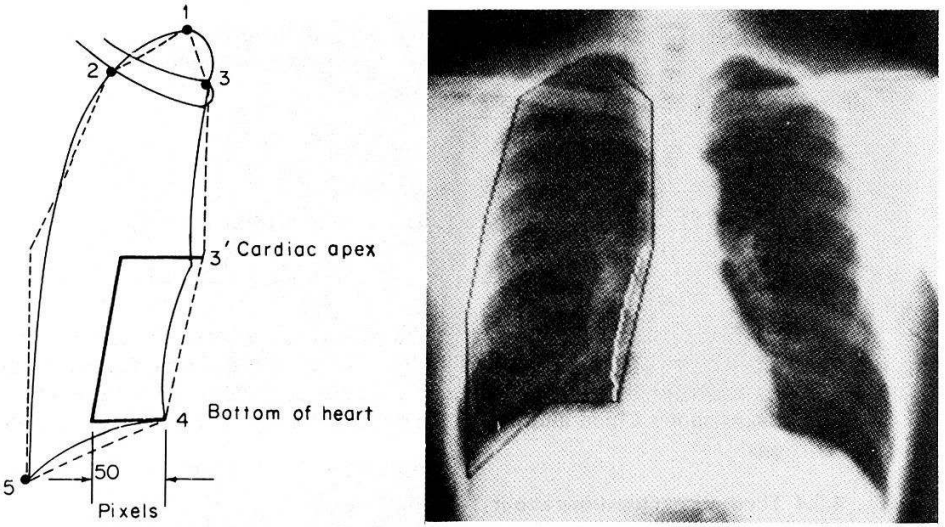


Fig. 4.17 Results of using local templates and global relations. (a) Model. (b) Results.

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-

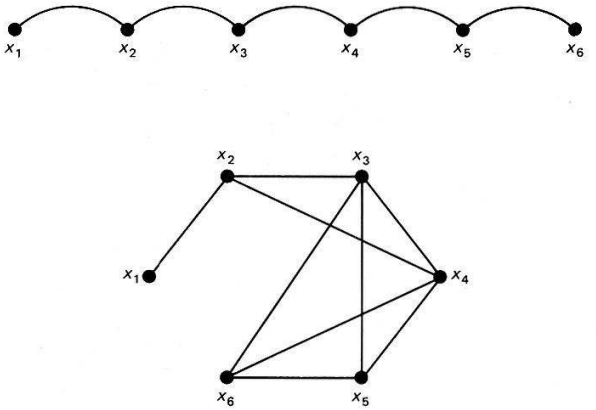


Fig. 4.18 Interaction graphs for DP (see text).

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.

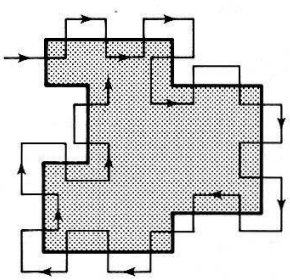


Fig. 4.19 Finding the boundary in a binary image.

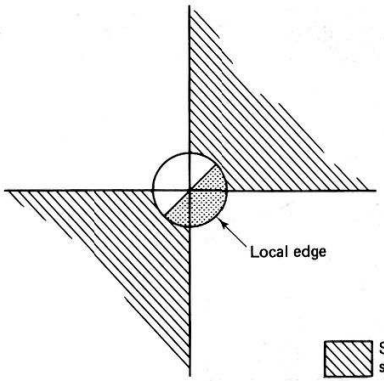


Fig. 4.20 Angular orientations for contour following.

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 sophisti-

cated 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

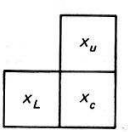


Fig. 5.1 L-shaped template for blob coloring.

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-

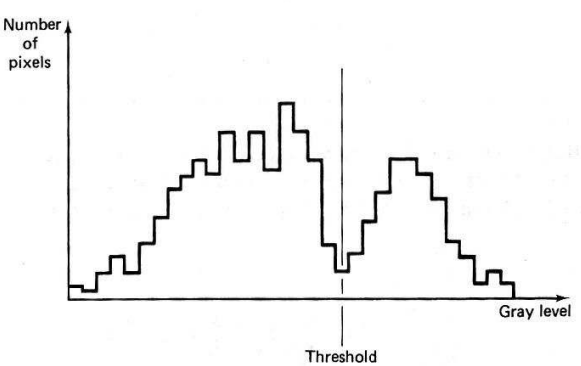


Fig. 5.2 Threshold determination from gray-level histogram.

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.

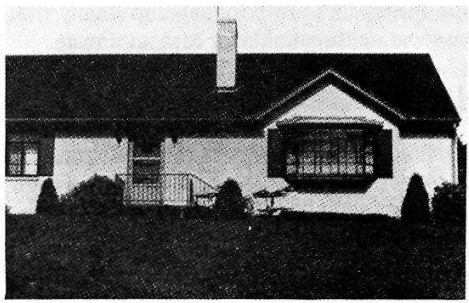
The region partitioning is then as follows:

Algorithm 5.2: Region Growing via Recursive Splitting

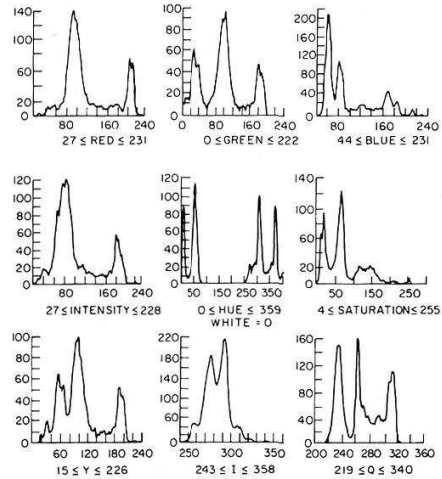
1. Consider the entire image as a region and compute histograms for each of the picture vector components.
2. Apply a peak-finding test to each histogram. If at least one component passes the test, pick the component with the most significant peak and determine two thresholds, one either side of the 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.

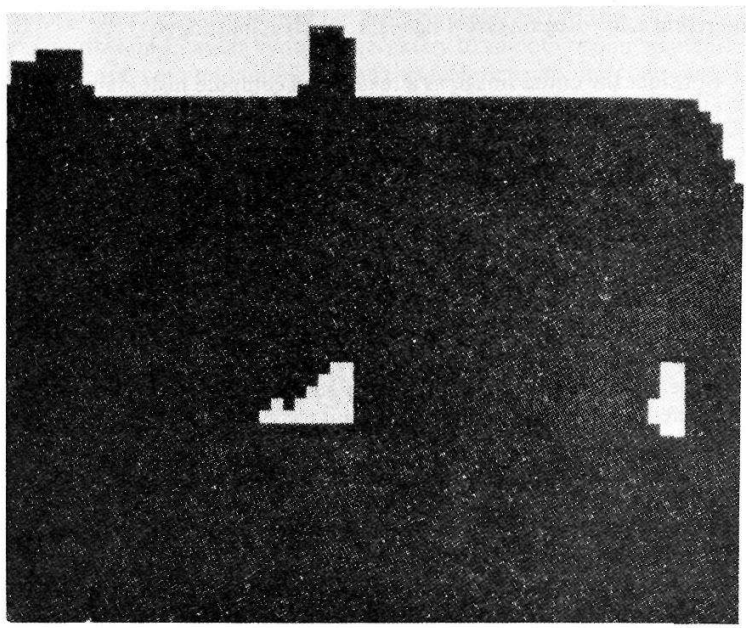
The figure shows the clear separation of peaks in the three-dimensional histogram that is not evident in either of the one-dimensional histograms. How many



(a)

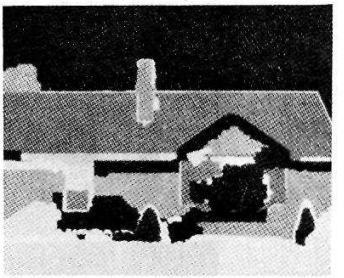


(b)



(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).

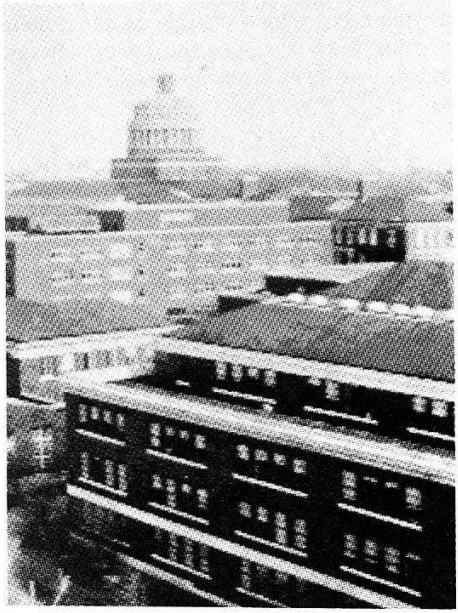
If Eq. (5.5) is not satisfied for some k, it means that that region is inhomogeneous and should be split into subregions. If Eq. (5.6) is not satisfied for some i and j, then regions i and j are collectively homogeneous and should be merged into a single region.

In our previous discussions we used

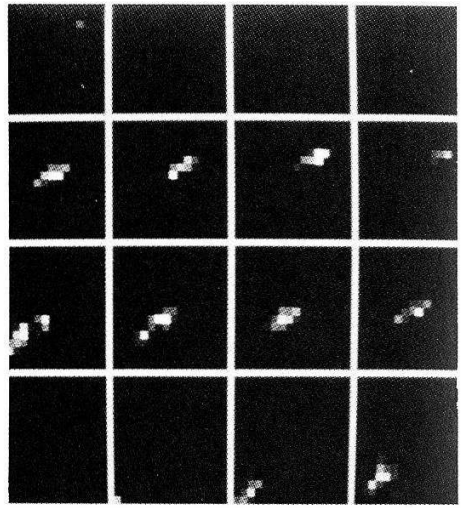
$$H(R) = \begin{cases} \text{true} & \text{if all neighboring pairs of points} \\ & \text{in } R \text{ are such that } f(\mathbf{x}) - f(\mathbf{y}) < T \\ \text{false} & \text{otherwise} \end{cases} \tag{5.7}$$

and

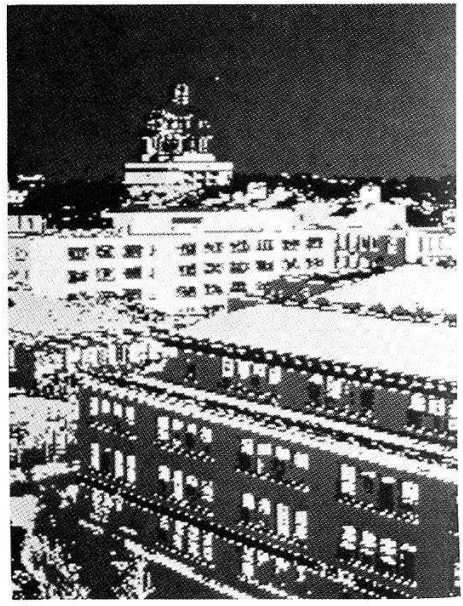
$$H(R) = \begin{cases} \text{true} & \text{if the points in } R \text{ pass a} \\ & \text{bimodality or peak test} \\ \text{false} & \text{otherwise} \end{cases} \tag{5.8}$$



(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.)

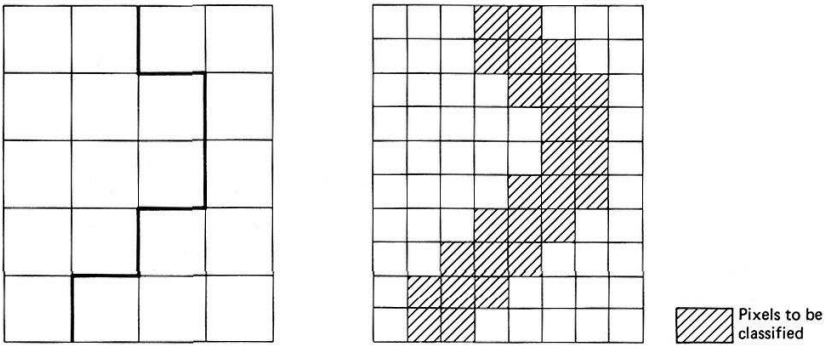


Fig. 5.5 Hierarchical region refinement.

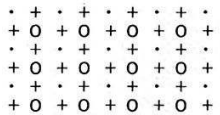
A way of working toward the satisfaction of these homogeneity criteria is the split-and-merge algorithm [Horowitz and Pavlidis 1974]. To use the algorithm it is necessary to organize the image pixels into a pyramidal grid structure of regions. In this grid structure, regions are organized into groups 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 $R_{k1}, \ldots, R_{k4}$, $H(R_{k1} \bigcup R_{k2} \bigcup R_{k3} \bigcup R_{k4}) =$ 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.



· Unassigned
+ Edge data
O Grey level data

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

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

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

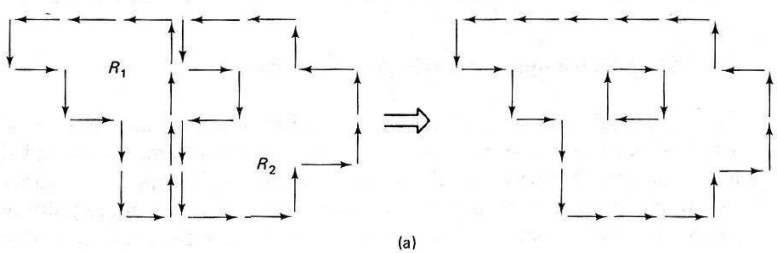


Fig. 5.7 Region operations on the grid structure of Fig. 5.6.

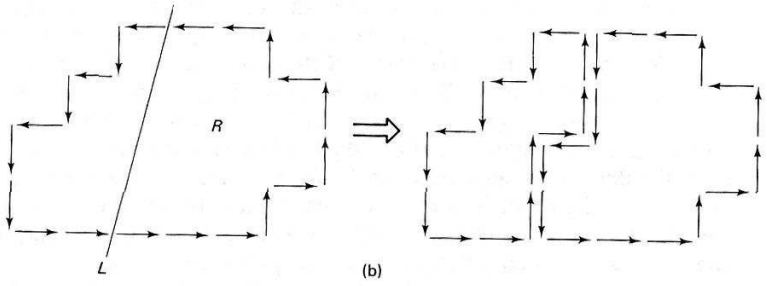


Fig. 5.7 (cont.)

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

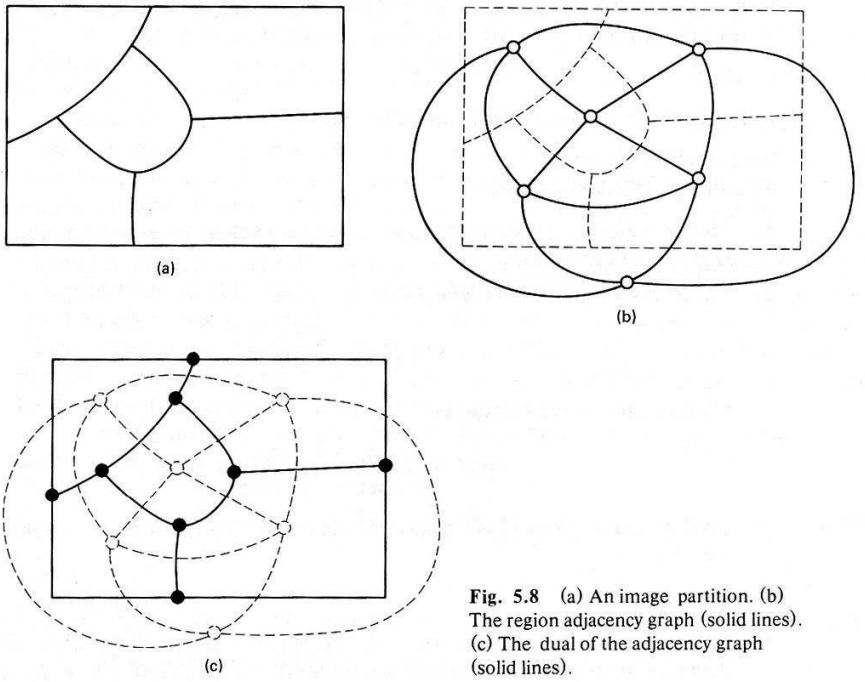


Fig. 5.8 (a) An image partition. (b) The region adjacency graph (solid lines). (c) The dual of the adjacency graph (solid lines).

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-

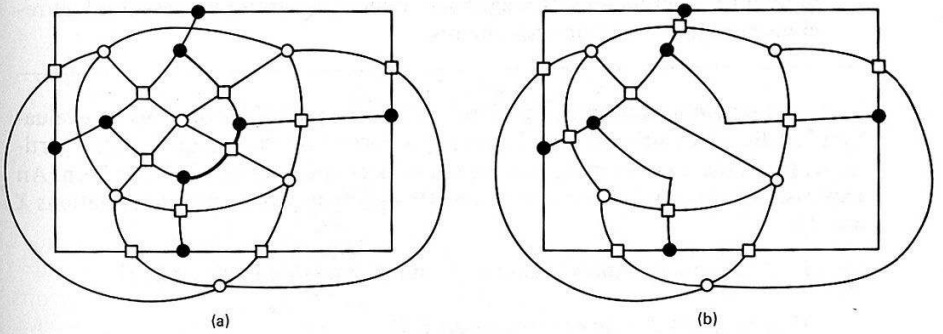


Fig. 5.9 Merging operations using the region adjacency graph and its dual. (a) Before merging regions separated by dark boundary line. (b) After merging.

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):

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

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 $P[R_i \text{ is a CAR} \mid 0 \leqslant i < I, 0 \leqslant h < H_1] =$, and $P[B_{ij}$ divides two car regions $\mid M_k \leqslant m < M_{k+1}, N_l < n \leqslant N_{l+1} =$. 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

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

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

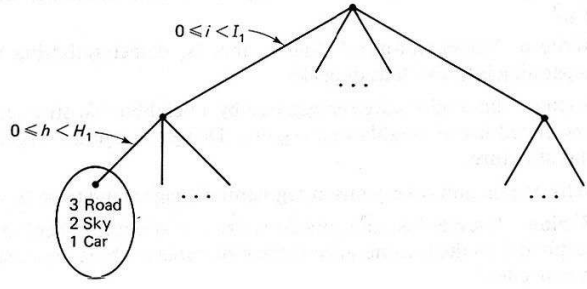


Fig. 5.10 Hypothetical classification tree for region measurements showing a particular branch for specific ranges of intensity and hue.

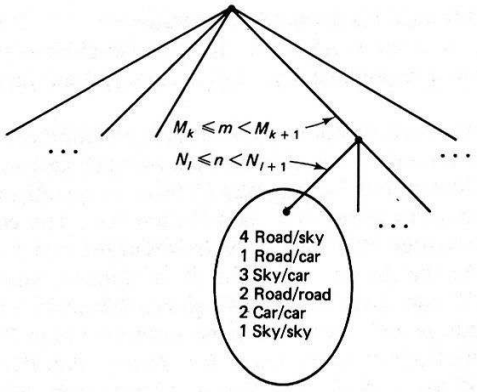


Fig. 5.11 Hypothetical classification tree for boundary measurements showing a specific branch for specific ranges of two measurements m and n.

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

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

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) $IT_2 \geq WT_2 > P_j$

(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.

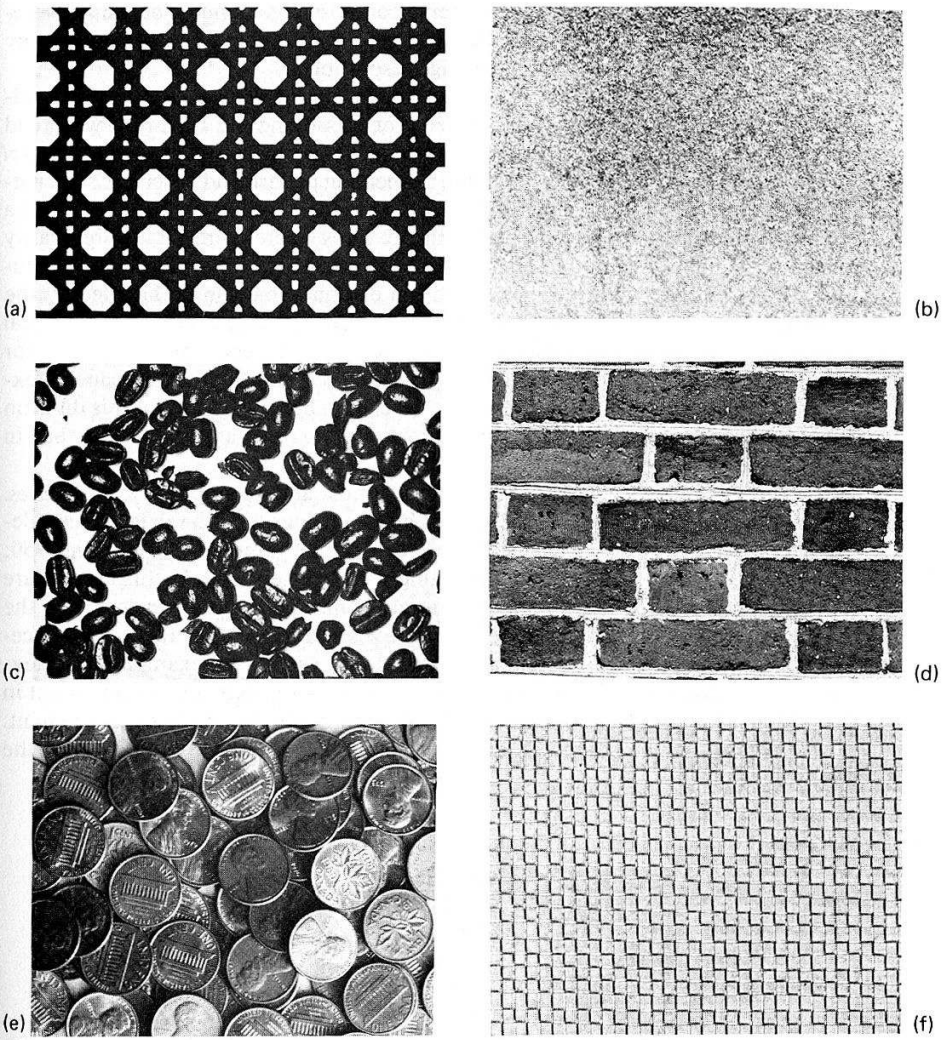


Fig. 6.1 Six examples of texture. (a) Cane. (b) Paper. (c) Coffee beans. (d) Brick wall. (e) Coins. (f) Wire braid.

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.

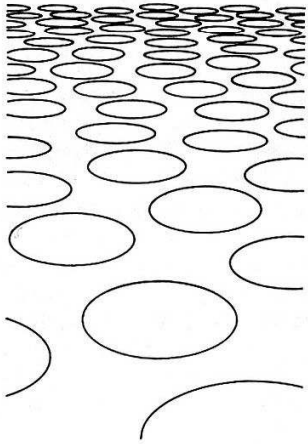


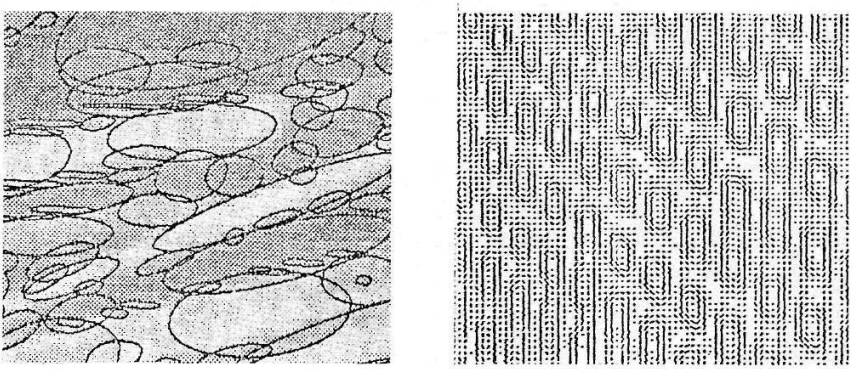
Fig. 6.2 Texture as a surface property.

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

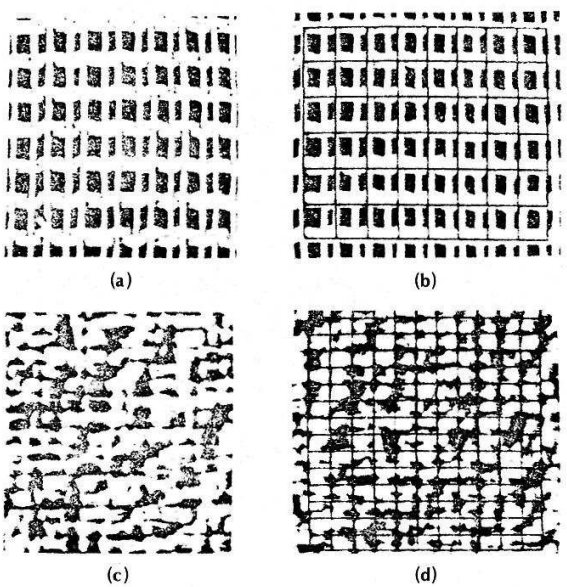


Fig. 6.4 Computing texture resolutions. (a) French canvas. (b) Resolution grid for canvas. (c) Raffia. (d) Grid for raffia.

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-

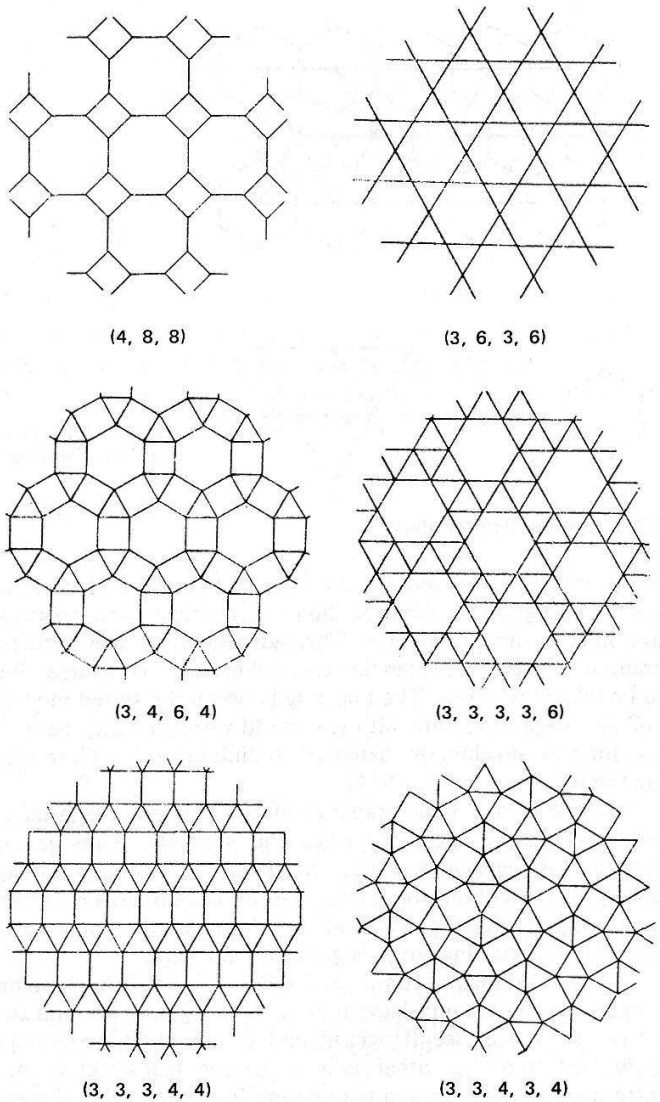


Fig. 6.5 Semiregular tesselations.

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.

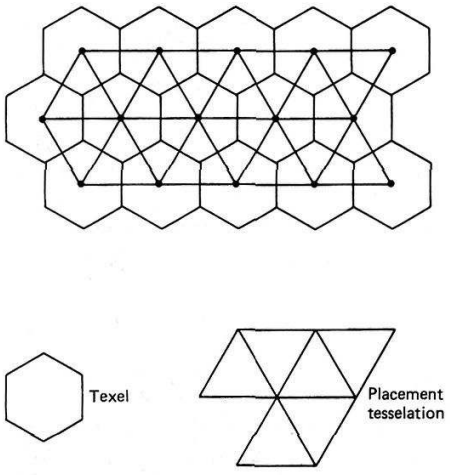


Fig. 6.6 The primitive placement tesselation as the dual of the primitive tesselation.

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