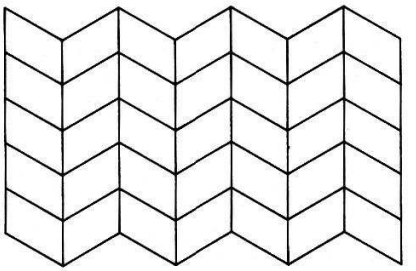


Fig. 6.7 Ambiguous texture.

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

6.3.2 Shape Grammars

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

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

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

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

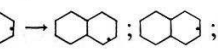
1. Find part of the shape that is geometrically similar to the left side of a rule in terms of both terminal elements and nonterminal elements (markers). There must be a one-to-one correspondence between the terminals and markers in the left side of the rule and the terminals and markers in the part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

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

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

$V_t = \{$  $\}$

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

$R:$  ;etc.

$S = \{$  $\}$

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

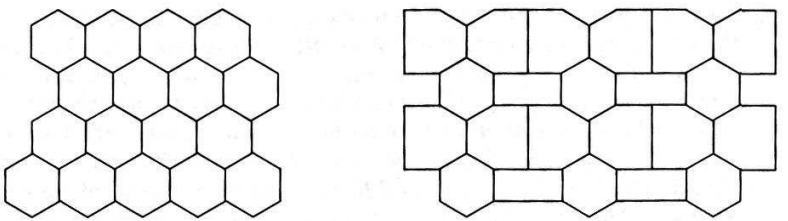


Fig. 6.8 Textures to be recognized (see text).

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

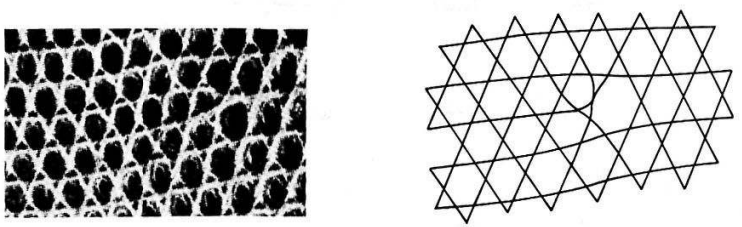


Fig. 6.9 (a) The reptile texture. (b) The reptile texture as a (3, 6, 3, 6) semiregular tesselation with local deformations.

6.3.3 Tree Grammars

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

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

is a tree grammar if

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

$$X_0 \rightarrow x \qquad \text{or} \qquad X_0 \rightarrow x$$
$$\quad\;\; \downarrow$$
$$X_0 \ldots X_{r(x)}$$

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

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

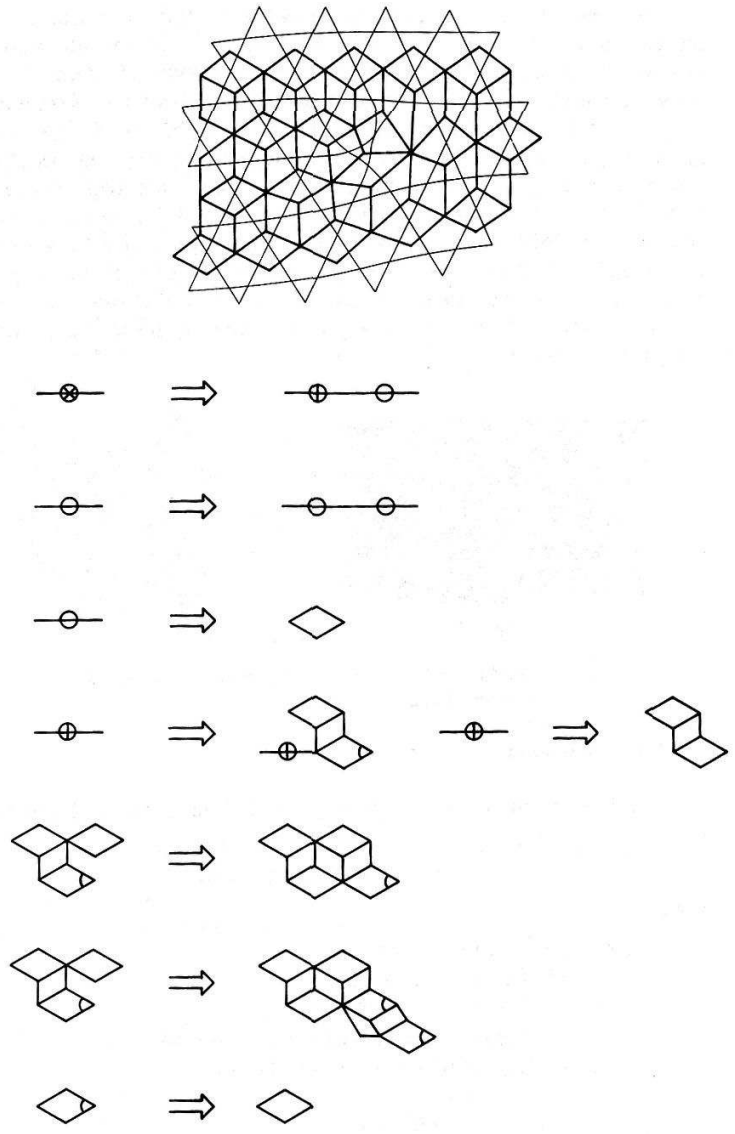


Fig. 6.10 Shape grammar for the reptile texture.

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

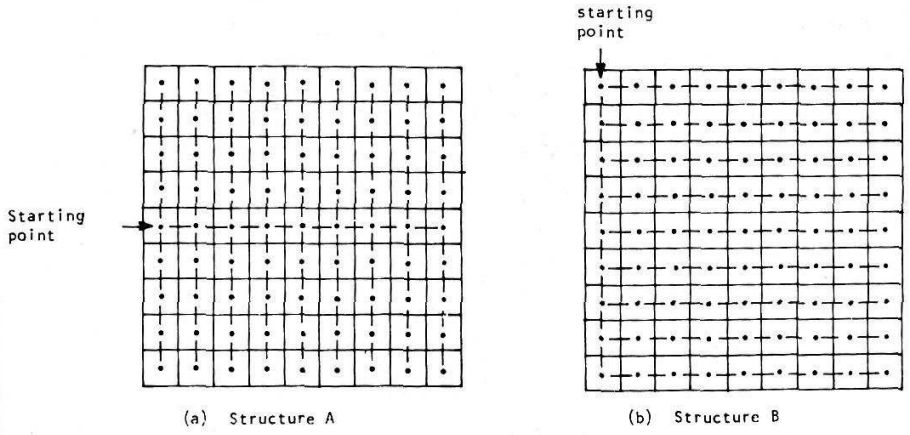


Fig. 6.11 *Two ways of embedding a tree structure in an array.*

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

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

where

$$\begin{aligned} V_t &= \{A_1, C_1\} \\ V_m &= \{X, Y, Z\} \\ r &= \{0, 1, 2\} \end{aligned} \tag{6.2}$$

R :

$$X \rightarrow \begin{array}{c} A_1 \\ X \quad Y \end{array} \quad \text{or} \quad \begin{array}{c} A_1 \\ \downarrow \\ Y \end{array}$$

$$Y \rightarrow \begin{array}{c} C_1 \\ \downarrow \\ Z \end{array} \quad \text{or } C_1$$

$$Z \rightarrow \begin{array}{c} A_1 \\ \downarrow \\ Y \end{array} \quad \text{or } A_1$$

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

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

where

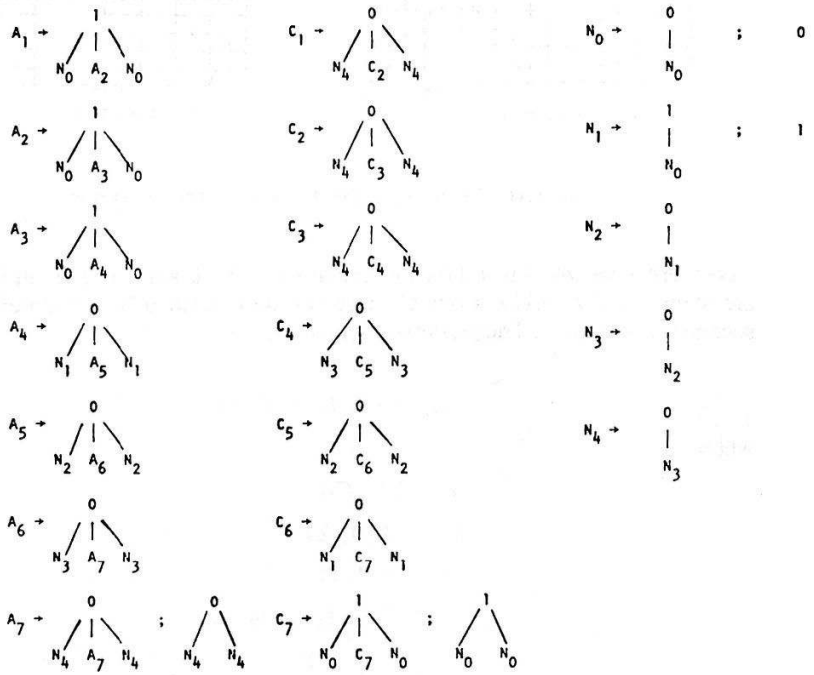
$$V_t = \{1, 0\}$$

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

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

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

R:



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

6.3.4 Array Grammars

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

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

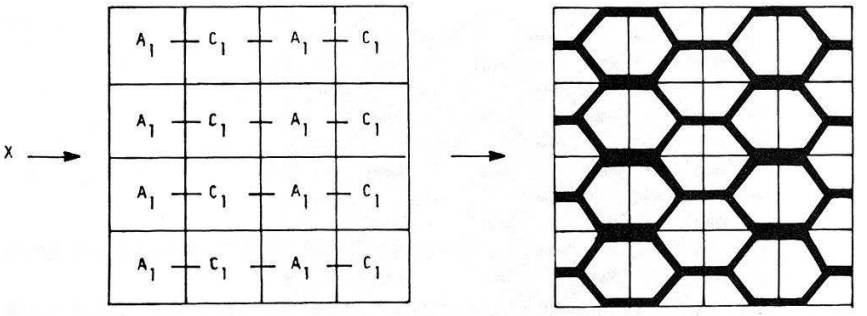


Fig. 6.12 Texture window and grammar (see text).

where

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

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

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

to summarize the four rules

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

Thus the checkerboard rule set is given by

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

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

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

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

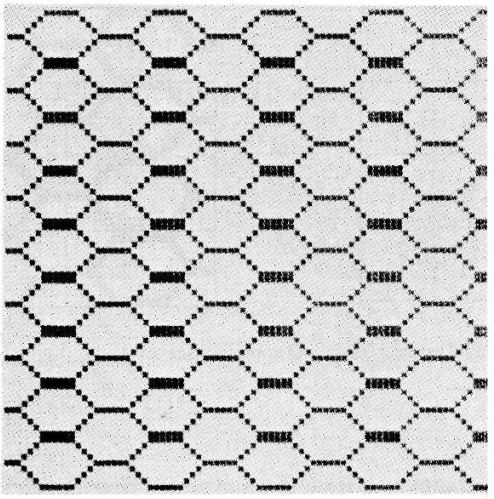


Fig. 6.13 Texture generated by tree grammar.

where

$$V_t = \{a\}$$

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

$$R: S \rightarrow a$$

$$a_x b \rightarrow a_x a \qquad x \text{ in } \{U, D, L, R\}$$

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

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

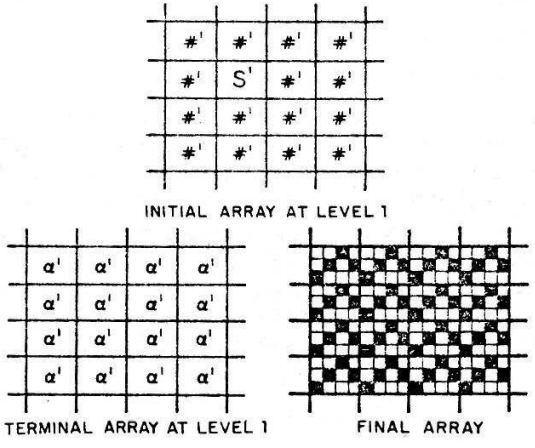


Fig. 6.14 Steps in generating a herringbone texture with an array grammar.

where

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

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

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

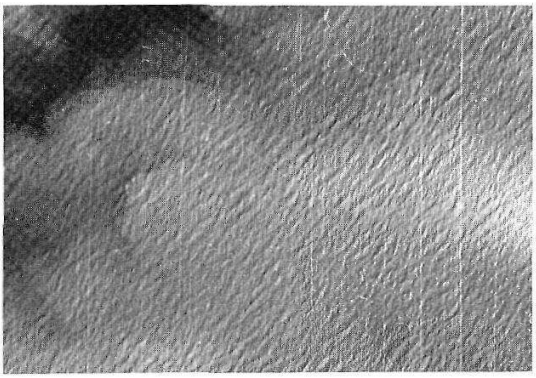
6.4 TEXTURE AS A PATTERN RECOGNITION PROBLEM

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

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

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

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



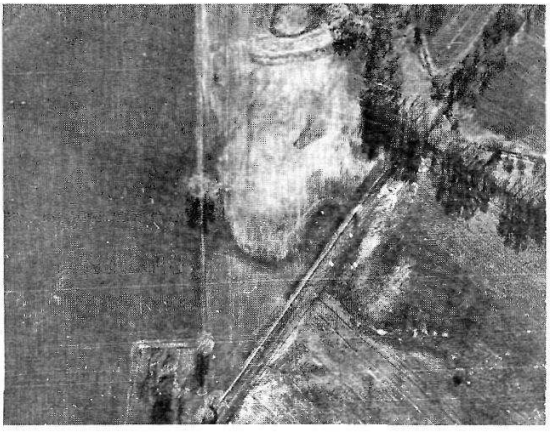


Fig. 6.15 Aerial image textures for discrimination.

Fig. 6.15 (cont.)

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

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

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

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

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

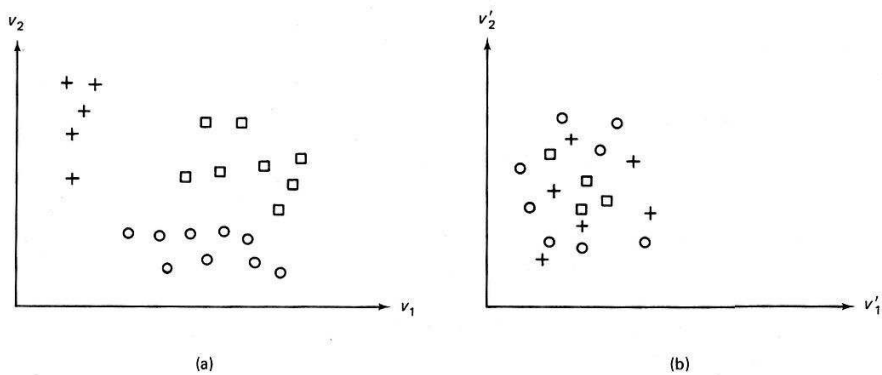


Fig. 6.16 Feature space for texture discrimination. (a) effective features (b) ineffective features.

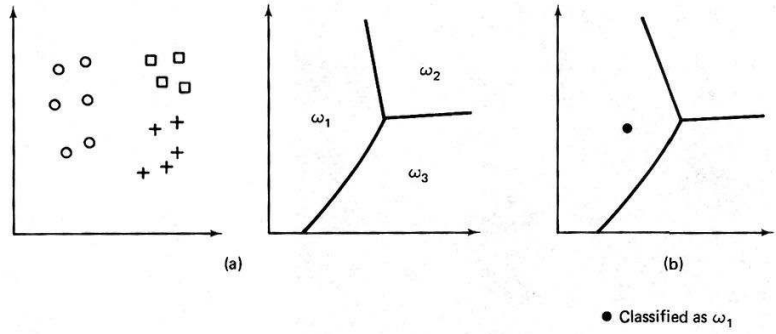


Fig. 6.17 Pattern recognition paradigm.

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

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

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

6.4.1 Texture Energy

Fourier Domain Basis

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

Radial features are given by

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

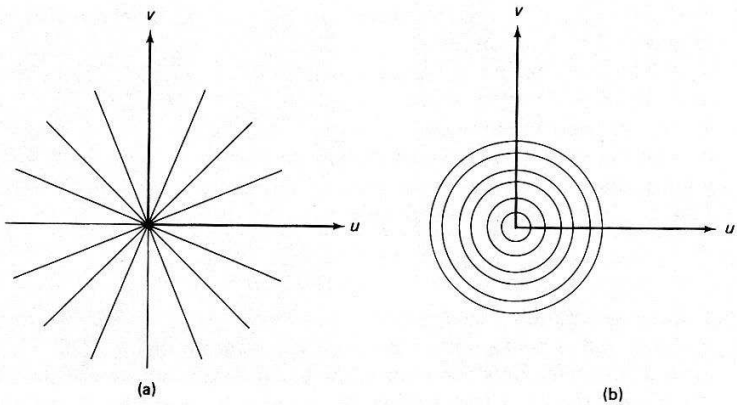


Fig. 6.18 Partitioning the Fourier domain into bins.

where the limits of integration are defined by

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

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

Features that measure angular orientation are given by

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

where the limits of integration are defined by

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

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

Texture Energy in the Spatial Domain

From Section 2.2.4 we know that the Fourier approach could also be carried out in the image domain. This is the approach taken in [Laws 1980]. The advantage of this approach is that the basis is not the Fourier basis but a variant that is more

matched to intuition about texture features. Figure 6.19 shows the most important of Laws' 12 basis functions.

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

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

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

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

6.4.2 Spatial Gray-Level Dependence

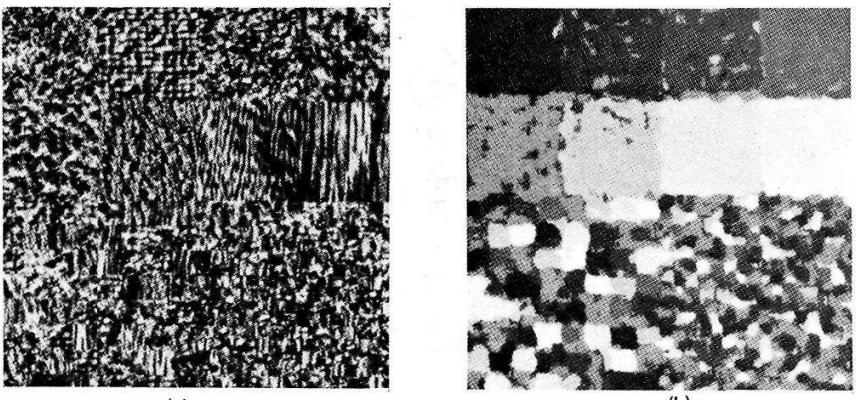
Spatial gray-level dependence (SGLD) matrices are one of the most popular sources of features [Kruger et al. 1974; Hall et al. 1971; Haralick et al. 1973]. The SGLD approach computes an intermediate matrix of measures from the digitized image data, and then defines features as functions on this intermediate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

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

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

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

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



(a) (b)

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

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

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

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

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

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

1. *Energy*

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

2. *Entropy*

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

3. *Correlation*

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

4. *Inertia*

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

5. *Local Homogeneity*

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

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

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

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

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

and

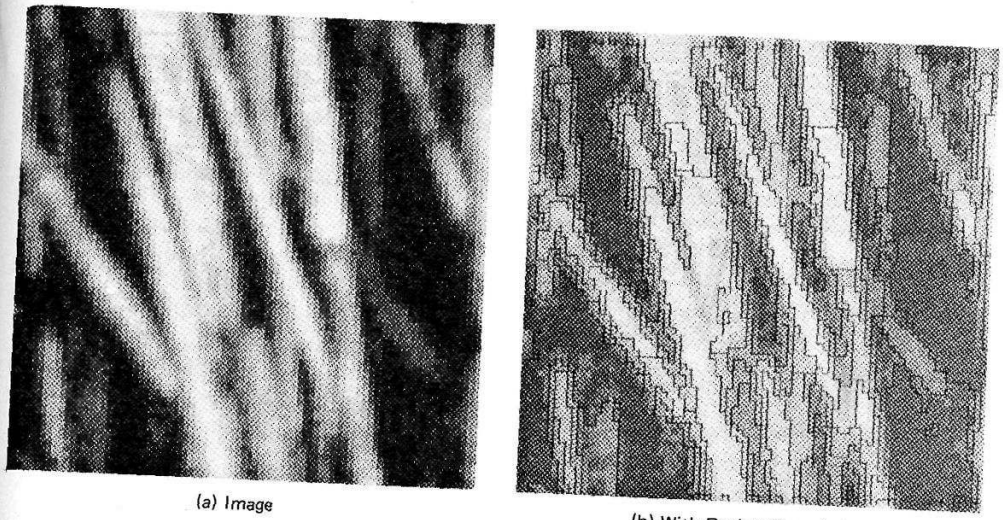
$$\sigma_y^2 = \sum_{i=0}^{K} (j-\mu_y)^2 \sum_{i=0}^{K} f(i, j|d, \theta) \tag{6.13d}$$

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

6.4.3 Region Texels

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

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



(a) Image (b) With Region Boundaries

Fig. 6.21 Region segmentation for straw texture.

6.5 THE TEXTURE GRADIENT

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

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

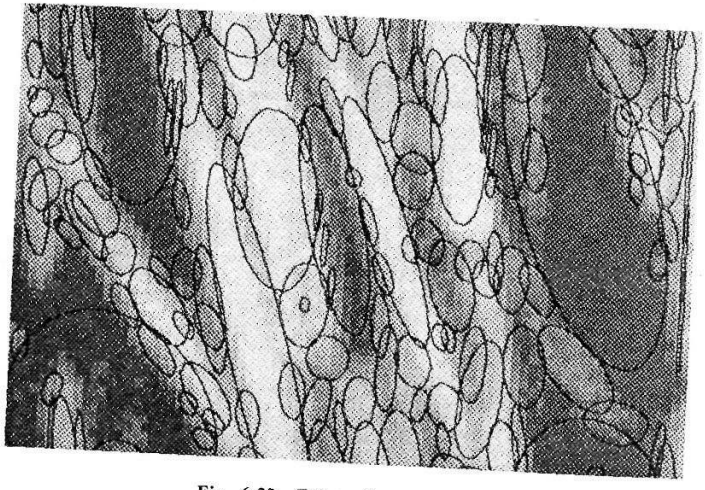
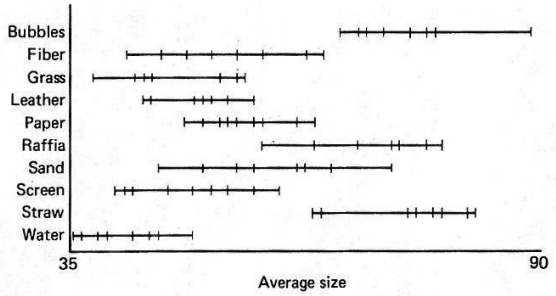


Fig. 6.22 Ellipses for straw texture.



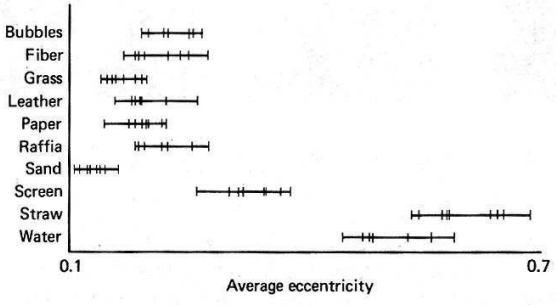


Fig. 6.23 Features defined on ellipses.

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

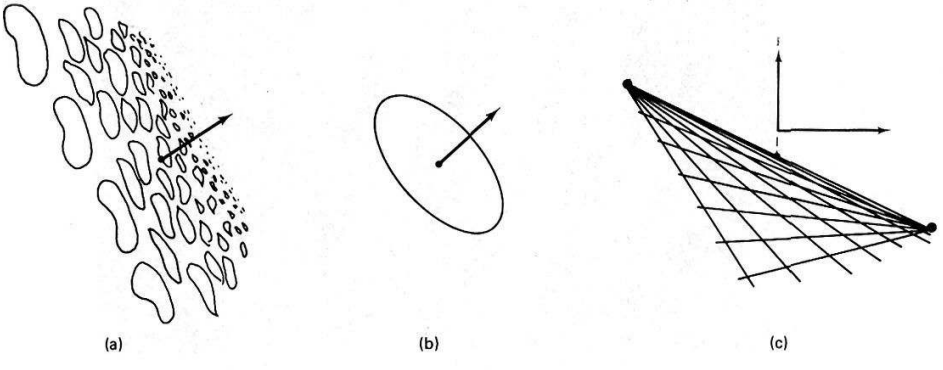


Fig. 6.24 Methods for calculating surface orientation from texture.

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

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

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

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

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

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

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

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

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

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

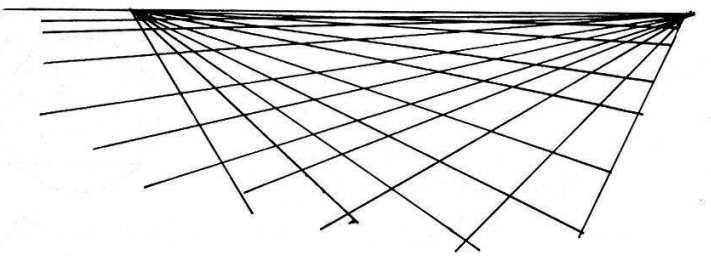


Fig. 6.25 Orthogonal line segments comprising a texture.

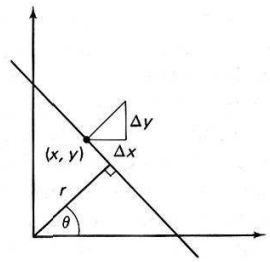


Fig. 6.26 r-θ transform.

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

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

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

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

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

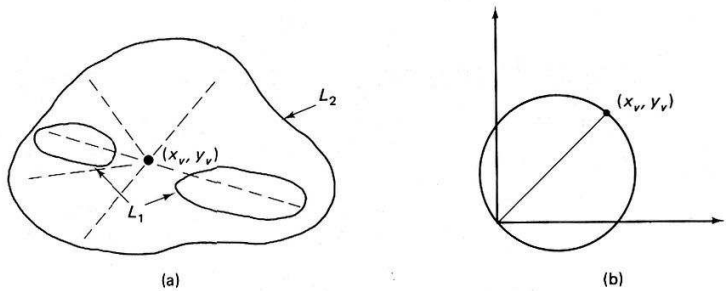


Fig. 6.27 Detecting the vanishing point with the Hough transform.

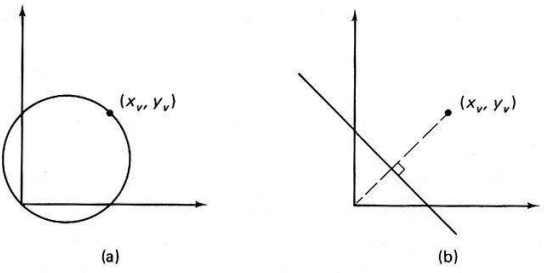


Fig. 6.28 Vanishing point loci.

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

EXERCISES

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

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

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

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

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

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

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

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

REFERENCES

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

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

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

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

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

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

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

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

HALL, E. L, R. P. KRUGER, S. J. DWYER III, D. L. HALL, R. W. MCLAREN, and G. S. LODWICK. "A survey of preprocessing and feature extraction techniques for radiographic images." *IEEE Trans. Computers 20*, September 1971.

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

HARALICK, R. M., R. SHANMUGAM, and I. DINSTEIN. "Textural features for image classification." *IEEE Trans. SMC 3*, November 1973, 610–621.

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

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

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

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

KRUGER, R. P., W. B. THOMPSON, and A. F. TWINER. "Computer diagnosis of pneumoconiosis." *IEEE Trans. SMC 45*, 1974, 40–49.

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

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

MALESON, J. T., C. M. BROWN, and J. A. FELDMAN. "Understanding natural texture." *Proc.*, DARPA IU Workshop, October 1977, 19–27.

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

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

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

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

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

TAMURA, H., S. MORI, and T. YAMAWAKI. "Textural features corresponding to visual perception." *IEEE Trans. SMC 8*, 1978, 460–473.

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

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

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

Motion 7

7.1 MOTION UNDERSTANDING

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

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

A further departure from the "static is basic" view is that motion understanding is qualitatively different from static vision. A logical extreme of this view is that there are many visual processing operations whose primitives are points in motion, and that in fact static vision is the puzzle, being ill-suited to the needs and mechanisms of biological systems. Serious work in computer motion understand-

ing has begun even more recently than computer vision as a whole, and it is too early to dismiss any approach out of hand. There are domains and applications in which the "static is basic" paradigm seems natural, but it also seems very reasonable that animals have perceptual systems or subsystems for which "motion is basic."

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

7.1.1 Domain Independent Understanding

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

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

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

7.1.2 Domain Dependent Understanding

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

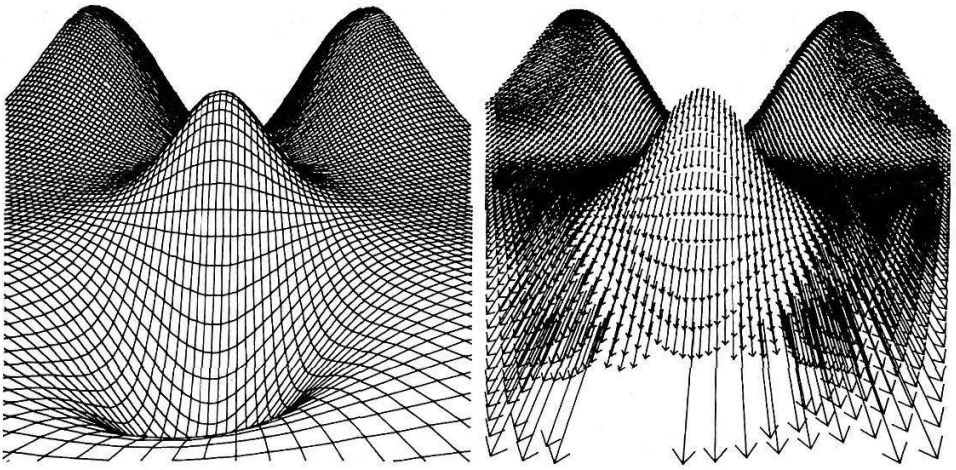


Fig. 7.1 An example of an optical flow field for an approaching "hill." (a) The hill. (b) Flow field.

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

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

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

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

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

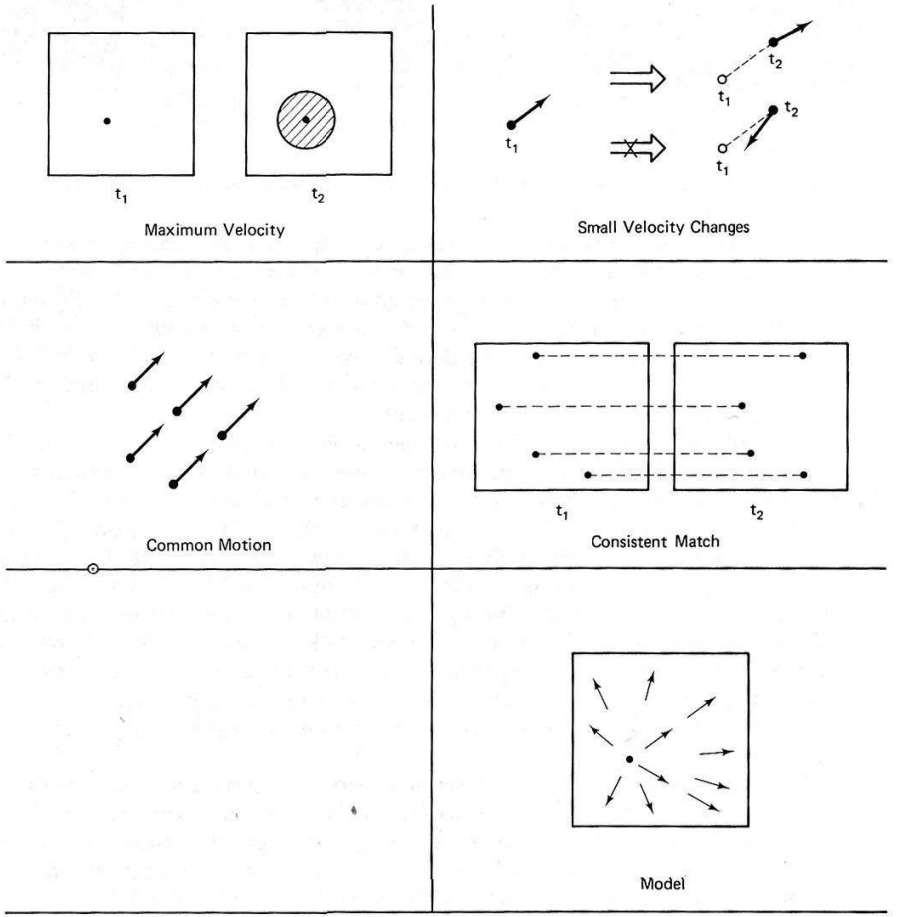


Fig. 7.2 Five heuristics.

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

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

7.2 UNDERSTANDING OPTICAL FLOW

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

7.2.1 Focus of Expansion

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

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

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

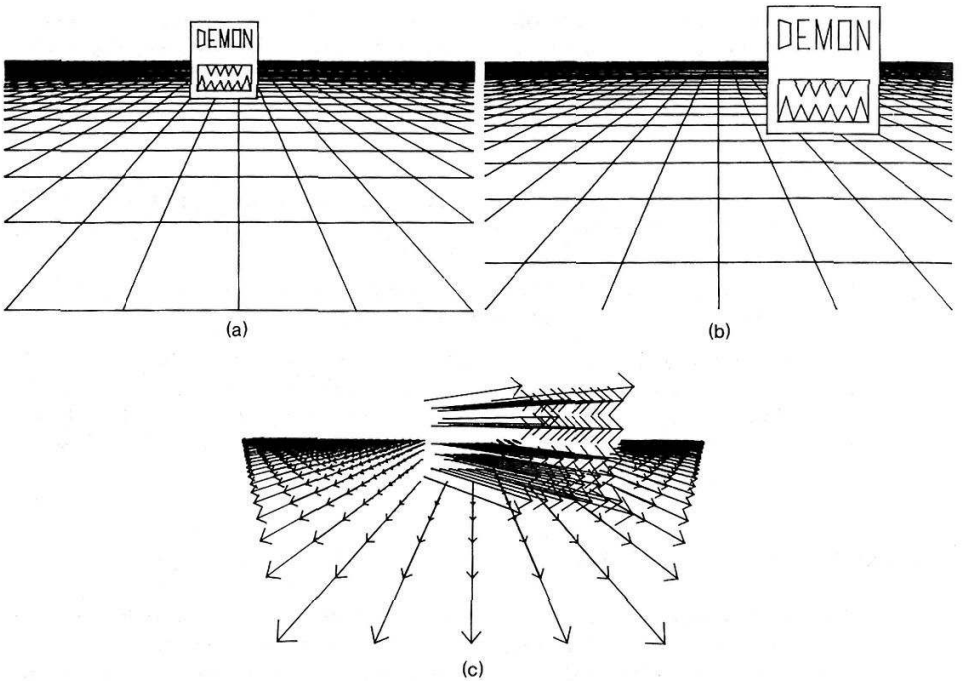


Fig. 7.3 FOE for rectilinear observer motion. (a) An image. (b) Later image. (c) Flow shows different FOEs for static floor and moving object.

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

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

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

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

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

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

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

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

7.2.2 Adjacency, Depth, and Collision

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For example, the position **P** along the observer's path at which a world point approaches closest is given by

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

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

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

7.2.3 Surface Orientation and Edge Detection

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

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

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

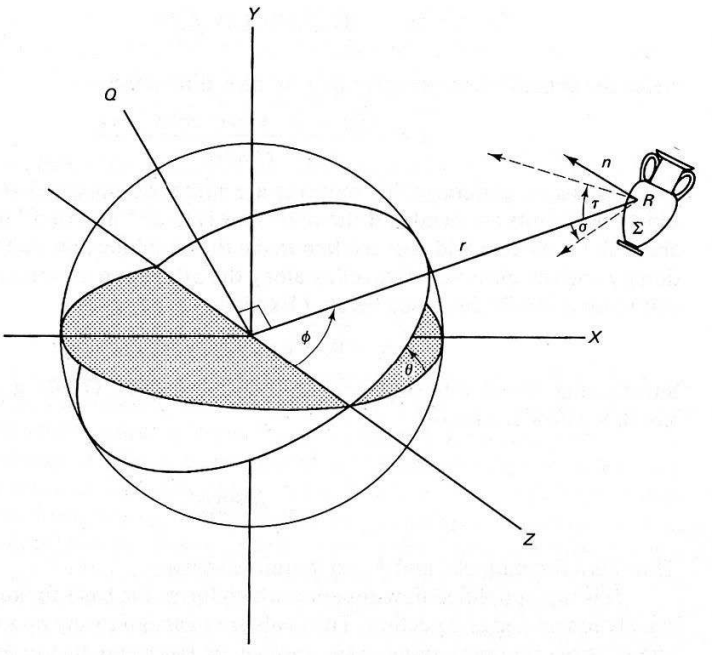


Fig. 7.4 Spherical coordinate system, and the definition of σ and τ.

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

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

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

Then from the coordinate transformation equations of Appendix 1,

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

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

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

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

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

The derivation of ϵ proceeds from the coordinate transformation equation

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

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

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

yields the general expression for flow in the ϕ direction:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

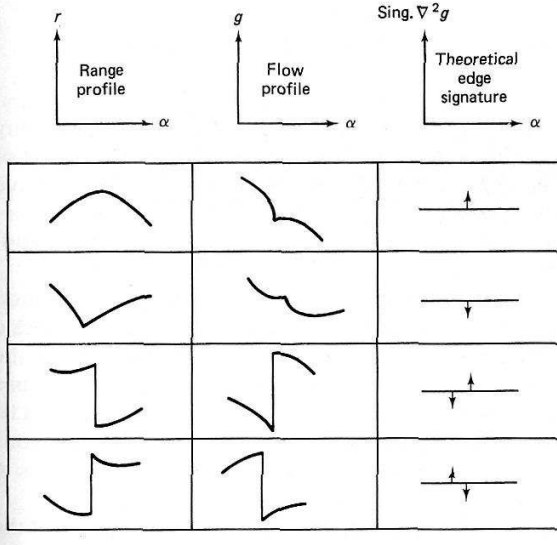


Fig. 7.5 The singularities of the second derivative of the flow profile inform about the type of edge.

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

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

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

7.2.4 Egomotion

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

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

The calculation yields a method for deriving relative depth, or the ratio of the

distances of points from the observer. An approximation to surface orientation may be obtained using several relative depth measurements in a small area and assuming that the surface normal varies slowly in tne area.

7.3 UNDERSTANDING IMAGE SEQUENCES

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

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

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

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

7.3.1 Calculating Flow from Discrete Images

This method is a form of disparity calculation that is not only used for flow calculations, but may also be used for stereo matching or tracking applications. The com-

putations are implemented with "relaxation" techniques.

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

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

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

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

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

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

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

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

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

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

where

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

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

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

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

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

and the probabilities are updated by

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

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

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

The following simplified example makes these ideas more concrete.

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

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

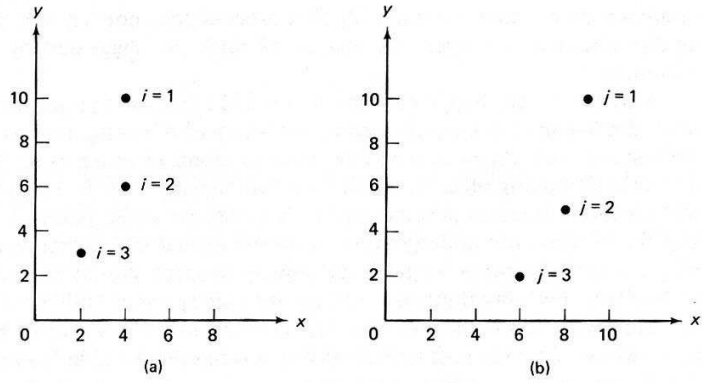


Fig. 7.6 Discrete matching: a concrete example.

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

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

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

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

which are normalized to be

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

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

7.3.2 Rigid Bodies from Motion

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

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

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

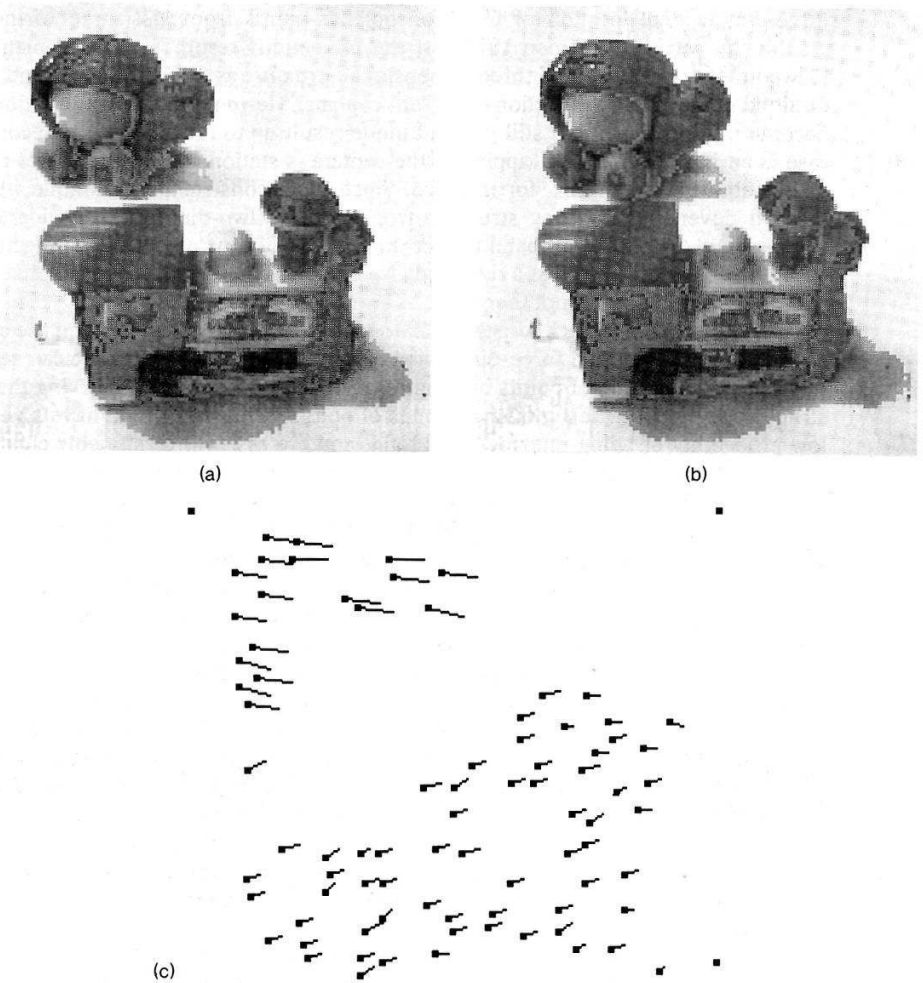


Fig. 7.7 Optical flow from feature point analyses. (a) An image. (b) Later image. (c) Optical flow found by relaxation.

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

If point correspondences are known, it is possible to compute a unique

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

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

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

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

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

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

The image coordinates are in fact

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

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

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

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

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

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

Equations (7.39) and (7.40) yield

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

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

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

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

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

To see how the equations might be set up, consider the perspective geometry of Section 7.2.1. In this simplified Cartesian system, Eqs. (7.1) and (7.2) are used as before. Since $z(x', y', 1) = (x, y, z)$, the location of any point is determined (up

to a scale factor, since the focal length is not explicit) from its image coordinates and its depth coordinate, z. For $F \geqslant 1$ images and $N \geqslant 3$ points there are $FN - 1$ unknowns (the ability to scale distance allows one point to be placed arbitrarily).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

One of the domains that provides the purest aspects of motion vision is moving light displays (MLDs). These are sequences of images which track only a few discrete points per frame. A typical way to produce an MLD is to attach small glass bead reflectors to a person's major joints (shoulders, elbows, wrists, hips, knees,

ankles), focus a strong light on him or her, and manipulate the contrast of a videotape recorder so as to produce on videotape a record of the movement of the reflective points on the joints. A single frame from such a record is unrecognizable by an inexperienced subject (Fig. 7.8).

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

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

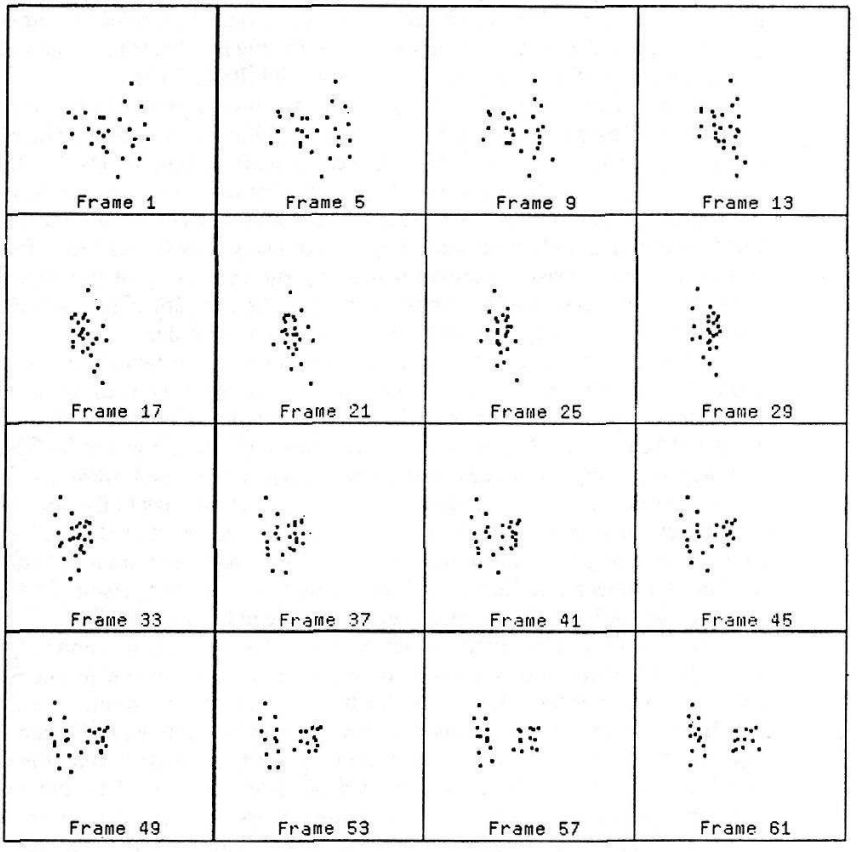


Fig. 7.8 An MLD for a man walking his dog.

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

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

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

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

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

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

A program can be given similar resilience by having it suspend judgment on contradictory clues and use succeeding frames to resolve the problem [Rashid 1980; O'Rourke 1980]. Having established local point correspondences, the next problem is to group the points into coherent three-dimensional structures and separate individual bodies moving in the scene. When constraints on the scene are available that make analytic techniques applicable (Section 7.3.1), explicit grouping of points prior to analysis may be unnecessary. In fact, with complex MLDs such as Ullman studied (e.g. two transparent but spotty coaxial cylinders rotating in opposite directions about an axis in the viewing plane), most naive grouping

strategies based on two-dimensional motion in the image will fail. Ullman's method chooses four-tuples of points from such a scene; on the average seven-eighths of such groups involve points from both cylinders, but with accurate data the algorithm can identify such nonrigid four-tuples. The remaining one-eighth of the groups have consistent interpretations as rigid rotating groups, and the groups fall into two classes, one for each cylinder.

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

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

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

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

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

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

7.3.4 Human Motion Understanding—A Model-Directed Approach

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

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

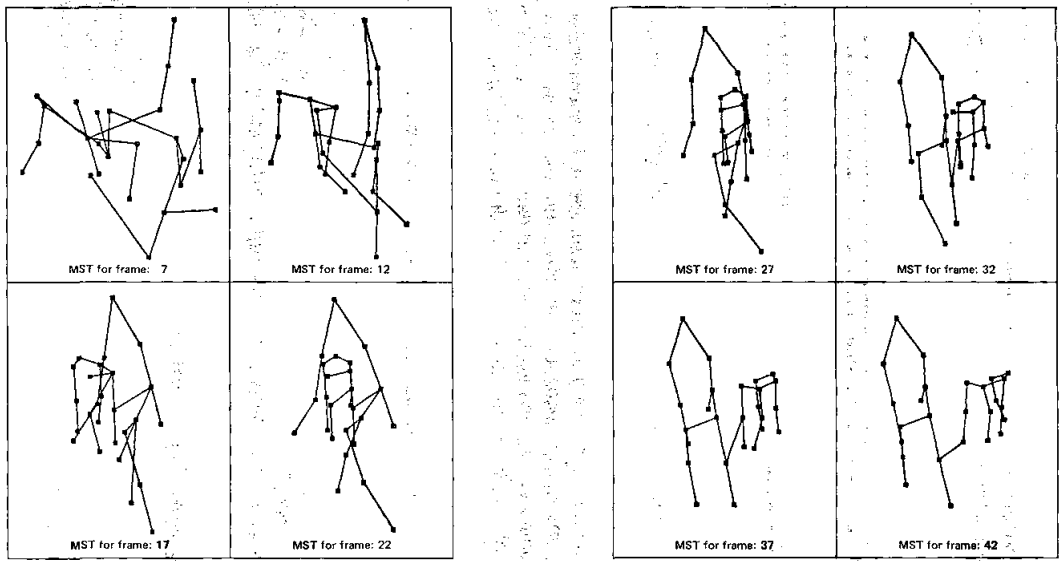


Fig. 7.9 The minimal spanning tree for the man and dog.

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

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

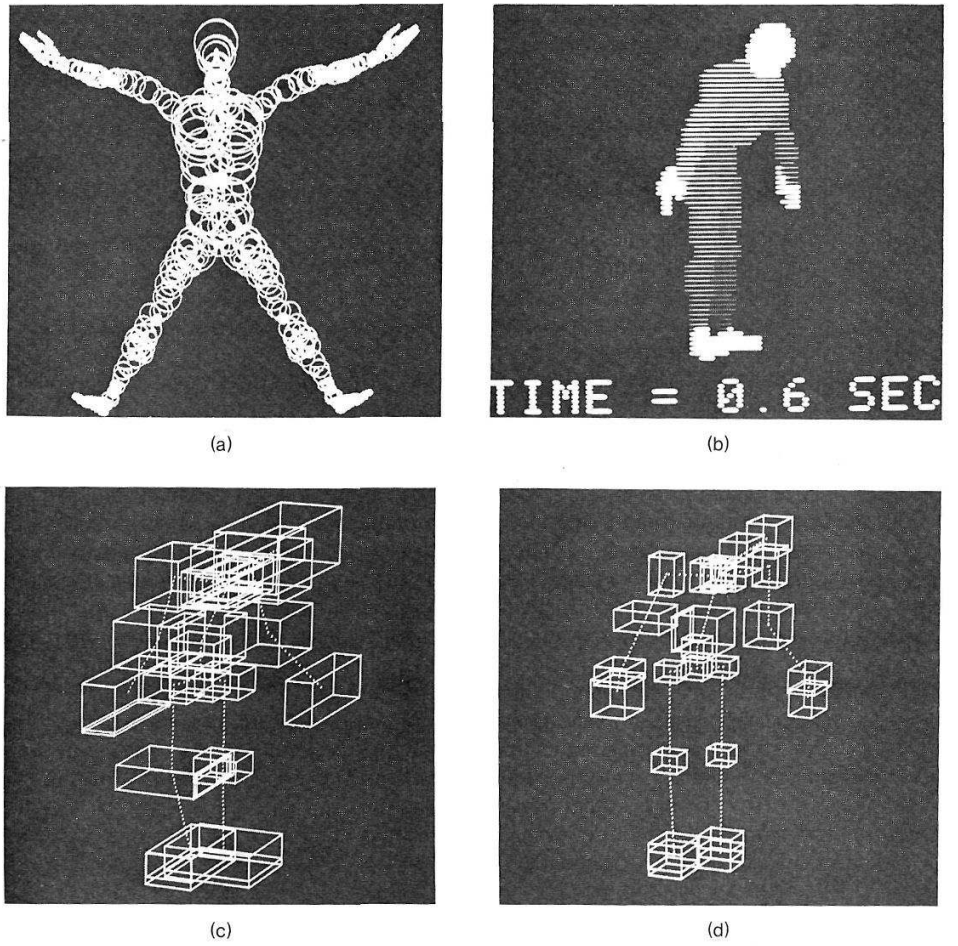


Fig. 7.10 Understanding human motion through the incorporation of many constraints. (a) Bubble Man from simulation program. (b) Input to motion understander; a bowing man. (c, d) Initial and final stages in understanding the motion of the bowing man.

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

7.3.5 Segmented Images

Moving Polygons and Line Drawings

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

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

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

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

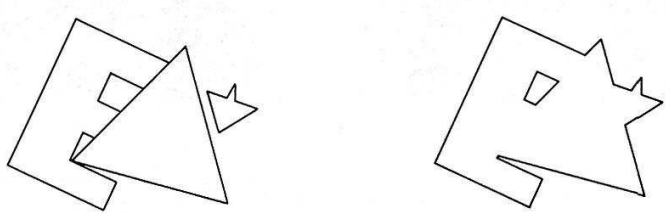


Fig. 7.11 Two frames from a motion image of three moving polygons.

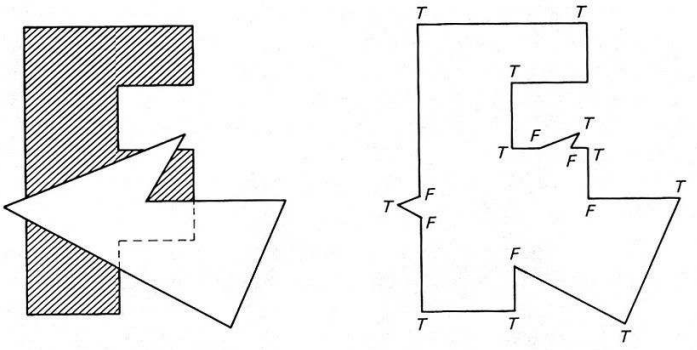


Fig. 7.12 True (T) and False (F) vertices in a scene of two overlapping polygons.

some three-dimensional aspects of the input, and to transform the pictorial input into natural language descriptions of motion.

Similarity Analysis, Then Difference Measurement

This approach is probably the most intuitive if motion perception is thought to be built up from perception of successive frames. The idea is simply to extract an object in one frame, and to search for it in the next frame. Obviously, the basic techniques here are the description-extraction process (i.e., static computer vision, the topic of most of this book) and matching (Chapter 11).

The entire range of matching techniques, from image matching to description matching, has been applied to image sequences. One characteristic of this approach in its pure form is that motion is merely a nuisance—segmentation is performed without using motion information. Usually the approach is pursued in a more pragmatic and domain-dependent fashion: for instance, the matching may be guided by knowledge about the motions.

One advanced system that uses this basic paradigm is described in [Price 1976; Price 1978; Price and Reddy 1977]. It segments and describes both images first. Using the symbolic descriptions, it matches complex scenes (such as houses or aerial images) that have been relatively rotated by large amounts (45 to 180°) and have size differences as well. It also derives the geometric transformation that produced the second image from the first.

Clearly, the major problems in systems of this sort come from generating and matching descriptions. The matching must be sophisticated, and to be successful in general it must combine symbolic and geometric components. The constraint that successive frames do not reflect violent motions eases the matching problem considerably, and iconic correlation techniques may sometimes apply.

Difference Measurement, Then Similarity Analysis

The idea behind this approach is to guide the similarity analysis with information about image differences. This seems a promising idea, because differences are easy to compute, whereas the very definition of similarity is open to question, and computing it may be arbitrarily complex.

In particular, in locating moving objects in an image sequence, one is invited to ignore the stationary background. The area of changing image can be tracked easily from image to image, and subjected to further analysis. Rather than trying to track an object from image to image, it is attractive to consider letting the object move far enough that it does not overlap between two images. Then the difference between the images will actually reflect the structure of the object.

One possible method [Nagel 1978a, 1978b; Jain and Nagel 1978] proceeds as follows:

1. Obtain two images from the motion sequence such that the object of interest will have moved far enough not to overlap in position in the two images. (One clearly needs information about the objects and the imaging parameters to assure no overlap.)
2. Segment the two images into regions.
3. Compute a dissimilarity measure between the overlapping areas of regions in the two images. One reasonable measure is the likelihood ratio for the two hypotheses that the intensities in the overlaps come from the same distribution of intensities or from different distributions.
4. In one of the images, take all regions that are most consistent with the hypothesis of different distributions and assume that they arise from the moving object (or its old vacated position). Merge these regions by a reasonable technique into one which is taken to include the moving object.
5. Take the boundary of the candidate region and use it as a template for correlation detection tracking between adjacent frames.
6. The offsets revealed by the correlation process give the velocity, and can be used to "subtract out" the motion, register the views of the object in several images, and thus obtain a more accurate characterization of the object.

This approach leads to results such as those shown in Fig. 7.13.

EXERCISES

7.1 Write a geometric explanation of the FOE phenomenon.

7.2 Devise a motion segmentation scheme for rigid bodies in translational three-dimensional motion that uses the FOE calculation.

7.3 Prove that the parametric flow path equation (7.3) indeed does produce a straight line in image coordinates.

7.4 Prove the time-to-adjacency relation (7.5). A geometrical demonstration may be made with similar triangles; an algebraic one is not very hard.

7.5 Express Eq. (7.12) as much as possible in terms of observables in the optical flow "image." What is left unspecified?

7.6 Perform Exercise 7.5 with equation (7.13).

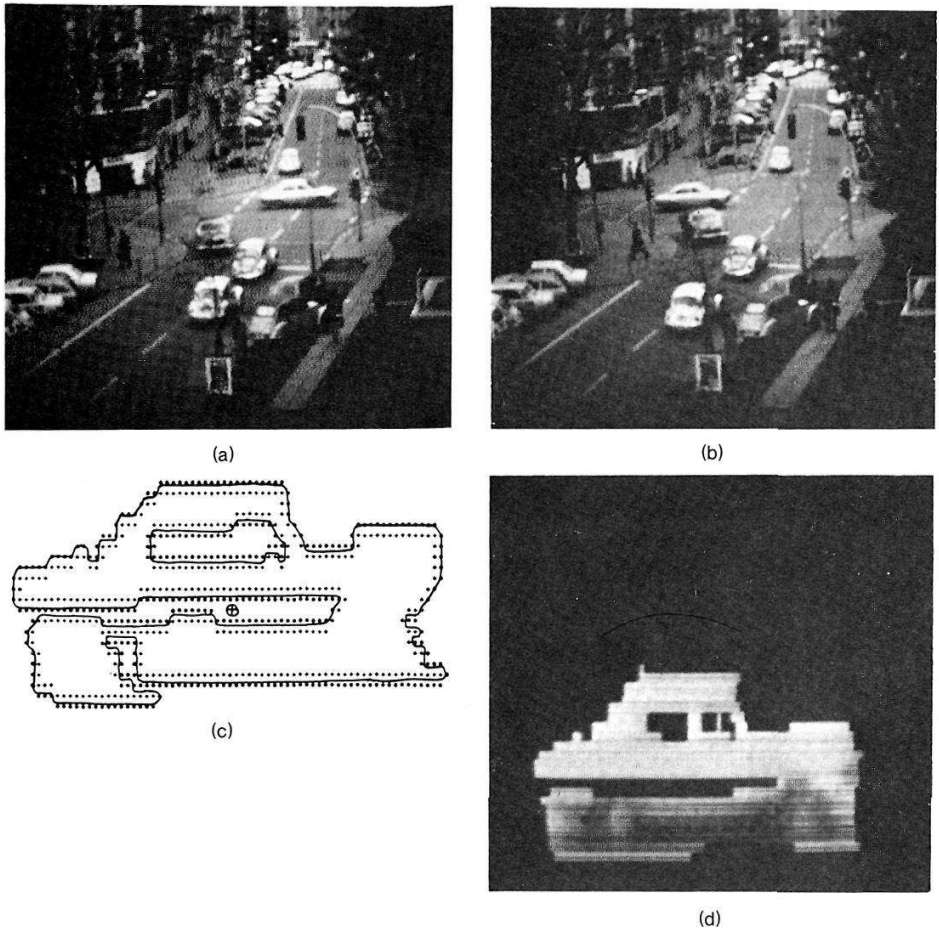


Fig. 7.13 Motion from segmented images. Initial (a) and final (b) frames from 16-frame sequence. The object of interest is the car moving left to right in the intersection. (c) Car segmentation from an intermediate frame. (d) Car reconstructed from several frames; the gray values result from aligning the values extracted from individual frames by segmentation.

7.7 Specialize the result of Exercise 7.6 to the case that the observer is moving in the direction of his direction of view [the FOE is at $(0, 0)$].

7.8 Fill in the steps in the derivations of the general and special cases of δ and ϵ (Eqs. (7.18) and (7.21) through (7.23)).

7.9 Fill in the steps in the derivations of $\tan\sigma$ and $\tan\tau$ (Eqs. (7.28) and (7.29)).

7.10 Show how to compute absolute depth from flow (Section 7.2.2) if the observer speed is known.

7.11 The Laplacian of ϵ in Section 7.2.3 is the sum of the second partial derivatives of ϵ

with respect to θ and ϕ. Write it out and show that it has singularities only when the Laplacian of depth (r) does except at $\phi = 0$ or π or $r = 0$.

7.12 In Section 7.2.2, the θ, ϕ system is divorced from the retinal position. How might this coordinate system be deduced from optical flow, or how might this deduction be unnecessary?

7.13 Work out the details of the vector equation referred to in the last paragraph of Section 7.3.2.

7.14 What do flow paths look like if the observer (or the environment) only executes rotational motion? Pick a congenial coordinate system and prove your supposition.

7.15 Tighten up the "common motion" heuristic in Section 7.1.2. What domains under what sorts of world motion yield what sorts of "common" image motions for objects?

REFERENCES

AGGARWAL, J. K. and R. O. DUDA. "Computer analysis of moving polygonal images." *IEEE Trans. Computers 24*, 1975, 966–976.

AGGARWAL, J. K., L. S. DAVIS, and W. N. MARTIN. "Correspondence processes in dynamic scene analysis." *Proc. IEEE 69*, 5, May 1981, 562–571.

BADLER, N. "Temporal scene analysis: conceptual descriptions of object movements." Technical Report 80, Dept. of Computer Science, Univ. Toronto, February 1975.

BADLER, N. I. and S. W. SMOLIAR. "Digital representations of human movement." *Computing Surveys 11*, 1, 19–38, March 1979.

BARNARD, S. T. and W. B. THOMPSON. "Disparity analysis of images." Technical Report 79-1, Computer Science Dept., Univ. Minnesota, January 1979.

BRAUNSTEIN, M. L. *Depth Perception through Motion.* New York: Academic Press, 1976.

CLOCKSIN, W. F. "Computer prediction of visual thresholds for surface slant and edge detection from optical flow fields." Ph.D. dissertation, Univ. Edinburgh, 1980.

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

GIBSON, J. J. *The Perception of the Visual World.* Boston: Houghton Mifflin, 1950.

GIBSON, J. J. "Continuous perspective transformations and the perception of rigid motion." *J. Experimental Psychology 54*, 1957, 129–138.

GIBSON, J. J. "Research on the visual perception of motion and change." In *Readings in the Study of Visually Perceived Movement*, Irwin M. Spigel (Ed.). New York: Harper & Row, 1965.

GIBSON, J. J. *The Ecological Approach to Visual Perception.* Ithaca, NY: Cornell University Press, 1966.

HELMHOLTZ, H. VON. *Treatise on Physiological Optics* (translated by J. P. C. Southall). New York: Dover Publications, 1925.

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

JAIN, R. and H.-H. NAGEL. "On a motion analysis process for image sequences from real world scenes." *Proc.*, IEEE Workshop on Pattern Recognition and Artificial Intelligence, Princeton, NJ, 1978.

JOHANSSON, G. "Perception of motion and changing form." *Scandinavian J. Psychology 5*, 1964, 181–208.

LAWTON, D. T. "The processing of dynamic images and the control of robot behavior." Ph.D. dissertation, Univ. Massachusetts, 1981.

LEE, D. N. and J. R. LISHMAN. "Visual proprioceptive control of stance." *J. Human Movement Studies 1*, 1975, 87–95.

MARTIN, W. N., and J. K. AGGARWAL. "Dynamic scene analysis." *CGIP 7*, 1978, 356–374.

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

NAGEL, H.-H. "Formation of an object concept by analysis of systematic time variations in the optically perceptible environment." *CGIP 7*, 2, June 1978a, 149–194.

NAGEL, H.-H. "Analysis techniques for image sequences." *Proc.*, 4th IJCPR, November 1978b, 186–211.

NAKAYAMA, K. and J. M. LOOMIS. "Optical velocity patterns, velocity sensitive neurons, and space perception." *Perception 3*, 1974, 63–80.

O'ROURKE, J. "Image analysis of human motion." Ph.D. dissertation, The Moore School of Electrical Engineering, Univ. Pennsylvania, 1980.

O'ROURKE, J. and N. I. BADLER. "Model-based image analysis of human motion using constraint propagation." *IEEE Trans. PAMI 2*, 4, November 1980.

POTTER, J. L. "Velocity as a cue to segmentation." *IEEE Trans. SMC 5*, 1975, 390–394.

PRAGER, J. M. "Segmentation of static and dynamic scenes." COINS Technical Report 79-7, Computer and Information Science, Univ. Massachusetts, May 1979.

PRAZDNY, K. "Egomotion and relative depth map from optical flow." Computer Science Dept., Univ. Essex, March 1979.

PRICE, K. E. "Change detection and analysis in multi-spectral images." Ph.D. dissertation, Dept. of Computer Science, Carnegie-Mellon Univ., 1976.

PRICE, K. E. "Symbolic matching and analysis with substantial changes in orientation." *Proc.*, IEEE Workshop on Pattern Recognition and Artificial Intelligence, Princeton, NJ, 1978.

PRICE, K. E., and R. REDDY. "Change detection and analysis in multi-spectral images." *Proc.*, 5th IJCAI, August 1977, 619–625.

RASHID, R. F. "LIGHTS: a system for interpretation of moving light displays." Ph.D. dissertation, Computer Science Dept., Univ. Rochester, April 1980.

ROACHE, J. W. and J. K. AGGARWAL. "On the ambiguity of three-dimensional analysis of a moving object from its images." IEEE Workshop on Computer Analysis of Time-Varying Imagery, April 1979.

ROGERS, B. and M. GRAHAM. "Motion parallax as an independent cue for depth." *Perception 8*, 1979, 125–134.

SCHIFF, W. "The perception of impending collision: A study of visually directed avoidant behavior." *Psychological Monographs 79*, 1965.

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.

SHAPIRA, R. and H. FREEMAN. "Computer description of bodies bounded by quadratic surfaces from a set of important projections." *IEEE Trans. Computers 27*, 9, September 1978, 841–854.

SNYDER, W. E. (ed.). "Computer analysis of time varying images," *IEEE Computer 14*, 8, August 1981.

THOMPSON, W. B. "Combining motion and contrast for segmentation." Technical Report 79-7, Computer Science Dept., Univ. Minnesota, March 1979.

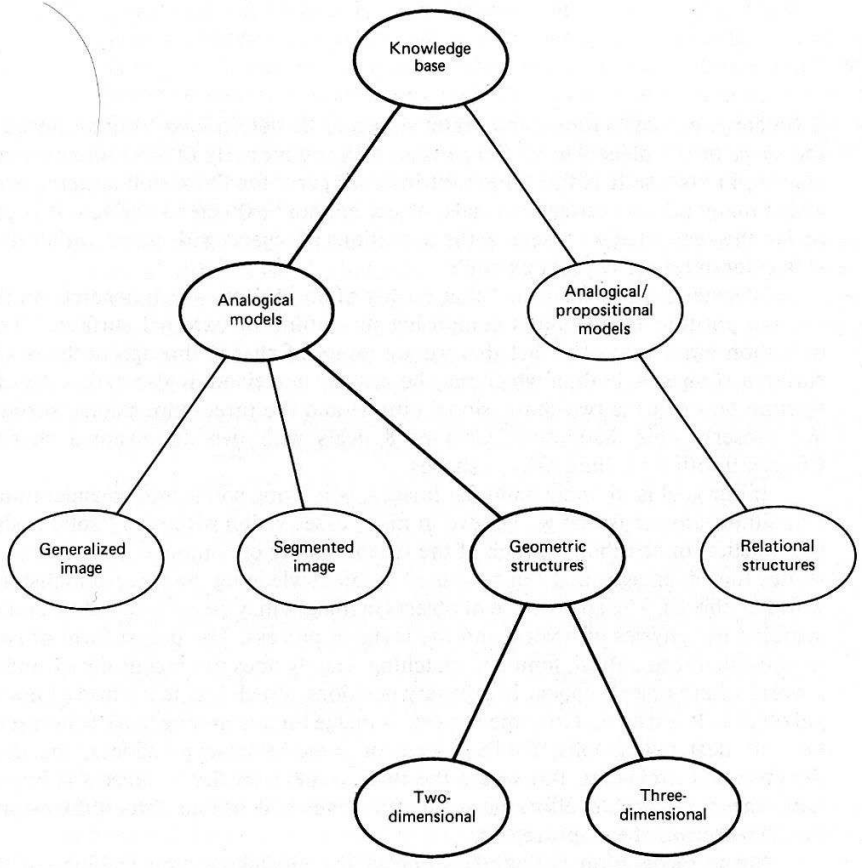
TSOTSOS, J. K., J. MYLOPOULOS, H. D. COVVEY and S. W. ZUCKER. "A framework for visual motion understanding." *IEEE Trans. PAMI 2*, 6, November 1980, 563–573.

ULLMAN, S. *The Interpretation of Visual Motion* (Ph.D. dissertation). Cambridge, MA: MIT Press, 1979.

WALLACH, H. and D. N. O'CONNELL. "The kinetic depth effect." *J. Experimental Psychology 45*, 4, 1953, 205–217.

WESLEY, M. A. and G. MARKOVSKY. "Fleshing out projections," Research Rpt. RC8884, Computer Sciences Dept., IBM, T. J. Watson Research Center, April 1981.

GEOMETRICAL STRUCTURES III



Ultimately, one of the most important things to be determined from an image is the *shape* of the objects in it. Shape is an intrinsic property of three-dimensional objects; in a sense it is the primal intrinsic property for the vision system, from which many others (surface normals, object boundaries) can be derived. It is primal in the sense that we associate the definitions of objects with shape, rather than with color or reflectivity, for example.

Webster defines shape as "that quality of an [object] which depends on the relative position of all points composing its outline or external surface." This definition emphasizes the fact that we are aware of shapes through outlines and surfaces of objects, both of which may be visually perceived. It also makes the distinction between the two-dimensional outline and the three-dimensional surface. We preserve this distinction: Chapter 8 deals with two dimensional shapes, Chapter 9 with three dimensional shapes.

If our goal is to understand flat images, why bring solids into consideration? Our simple answer is that we believe in many cases vision without a "solid basis" is a practical impossibility. Much of the recent history of computer vision demonstrates the advantages that can be gained by acknowledging the three-dimensional world of objects. The appearance of objects in images may be understood by understanding the physics of objects and the imaging process. The purest form of two-dimensional recognition, template matching, clearly does not practically extend to a world where objects appear in arbitrary positions, much less to a world of nonrigid objects. It is true that in some important image understanding tasks (interpretation of chest radiographs, ERTS images or some microscope slides), the third dimension is irrelevant. But where the three-dimensionality of objects is important, the considerable effort necessary to develop a usable three-dimensional model will always be amply repaid.

Shape recognition is doubtless one of the most important facilities of the mammalian visual system. We have seen how important shape information can be

extracted from images in early processing and segmentation. One of the major challenges to computer vision is to represent shapes, or the important aspects of shapes, so that they may be learned, matched against, recollected, and used. This effort is hampered by several factors.

1. *Shapes are often complex.* Whereas color, motion, and intensity are relatively simply quantified by a few well-understood parameters, shape is much more subtle. Common manufactured or natural shapes are incredibly complex; they may be represented "explicitly" (say by representing their surface) only with hundreds of parameters. Worse, it is not clear what aspects of shapes are important for applications such as recognition. An explicit and complete representation may be computationally intractable for such basic uses as matching. What "shape features" can be used to ease the burden of computation with complex shapes?
2. *Introspection is no help.* Human beings seem to have a large fraction of their brains devoted to the single task of shape recognition. This important activity is largely "wired in" at a level below our conscious introspection. Why is shape recognition so easy for human beings and shape description so hard? The fact that we have no precise language for shape may argue for the inaccessibility of our shape-processing algorithms or data structures. This lack of cognitive leverage is a trifle daunting, especially when taken with the complexity of everyday shapes.
3. *There is little classical guidance.* Mathematics traditionally has not concerned itself with shape. For instance, only recently has there been a mathematical definition of "rigid solid" that accords with our intuition and of set operations on solids that preserve their solidity. The fact that such basic questions are only now being addressed indicates that computer science must do more than encode some already existing proven ideas. Thus we have the next point.
4. *The discipline is young.* Until very recently, human beings communicated about complex shapes mainly through words, gestures, and two-dimensional drawings. It was not until the advent of the digital computer that it became of interest to represent complex shapes so that they could be specified to the machine, manipulated, computed with, and represented as output graphics. No generally accepted single representation scheme is available for all shapes; several exist, each with its advantages and disadvantages. Algorithms for manipulating shapes (for example, for computing how to move a sofa up a flight of stairs, or computing the volume of a specified shape) are surprisingly complex, and are research topics. Often the representations good for one application, such as recognition, are not good for other computations.

It is the intention of this part of the book to indicate some of what is known about the representation of shape. Although the details of geometric representations may be still under development, they are an essential part of our layered computer vision organization. They are more abstract than segmented structures and are distinguished from relational structures by their preponderance of metric information.

Representation of Two-Dimensional Geometric Structures 8

8.1 TWO-DIMENSIONAL GEOMETRIC STRUCTURES

The structures of this chapter are the intuitive ones of well-behaved planar regions and curves. A mathematical characterization of these structures that bars "pathological" cases (such as regions of a single point and space-filling curves) is possible [Requicha 1977]. Basically the requirement is that regions be "homogeneously two-dimensional" (contain no hanging or isolated structures of different dimension—solids, lines or points). Similarly, curves should be homogeneously one-dimensional. The property of regularity is sometimes important; a regular set is one that is the closure of its interior (in the relevant one- or two-dimensional topology). Intuitively, regularizing a two-dimensional set (taking the closure of its interior) first removes any hanging one- and zero-dimensional parts, then covers the remainder with a tight skin (Fig. 8.1). In computer vision, often regions and curves are discrete, being defined on a raster of pixels or on an orthogonal grid of possible primitive edge segments. It is frequently convenient to associate a direction with a curve, hence ordering the points along it and defining portions of the plane to its left and right.

The one-dimensional closed curve that bounds a well-behaved region is an unambiguous representation of it; Section 8.2 deals with representations of curves and hence indirectly of regions. Section 8.3 deals with other unambiguous representations of regions that are not based on the boundary. Sometimes unambiguous representation is not the issue; it may be important to have qualitative description of a region (its size or shape, say). Section 8.4 presents several terse descriptive properties for regions.

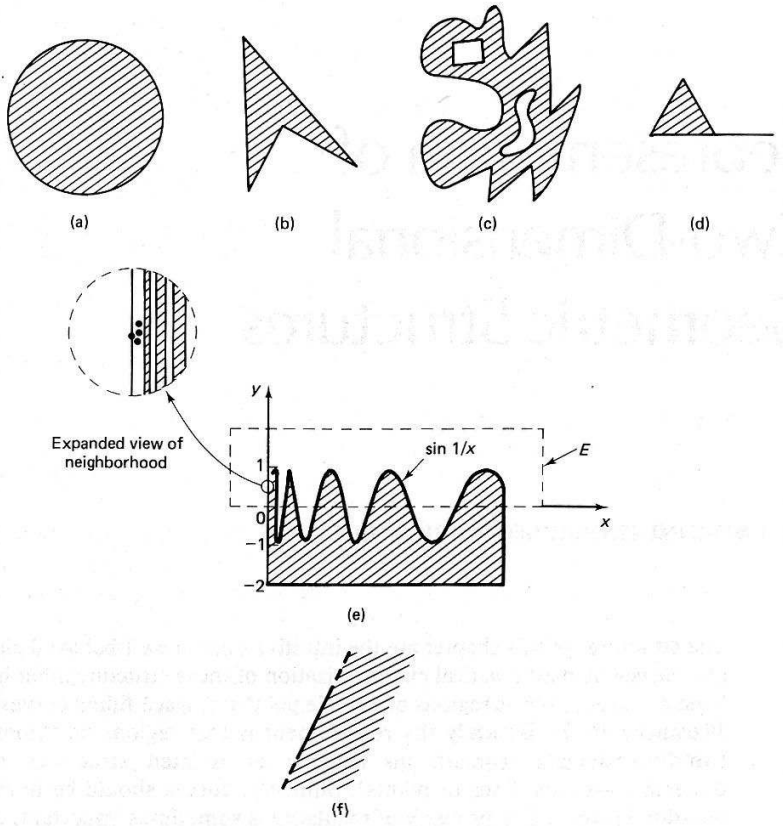


Fig. 8.1 (a, b, c) are Regions; (d) (e) and (f) are not.

8.2 BOUNDARY REPRESENTATIONS

8.2.1 Polylines

The "two-point" form of a line segment (see Appendix 1) extends easily to the *polyline*, which represents a concatenation of line segments as a list of points. Thus the point list $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ represents the concatenation of the line segments from $\mathbf{x}_1$ to $\mathbf{x}_2$ and from $\mathbf{x}_2$ to $\mathbf{x}_3$. If the first point is the same as the last, a closed boundary is represented.

Polylines can approximate most useful curves to any desired degree of accuracy. One might think there is one obvious way to approximate a boundary curve (or raw data) with a polygonal line. This is not so: many different approaches are possible. Finding a satisfying polygonal approximation to a given curve basically involves segmentation issues. The problem is to find corners or *breakpoints* that

yield the "best" polyline. As with region-based segmentation schemes, the ideas here can be characterized by the concepts of *merging* and *splitting*. Splitting and merging schemes may be combined, especially if the appropriate number of linear segments is known beforehand. For details, see [Horowitz and Pavlidis 1976].

In a merging algorithm, points along a curve (possibly in image data) are considered in order and accepted into a linear segment as long as they fit sufficiently well. When they do not, a new segment is begun. The efficiency and characteristics of these schemes are quite variable, and endless variations on the general idea are possible. A few examples of "one pass" merging schemes are given here: explicit algorithms are available in [Pavlidis 1977].

If the boundary (represented on a discrete grid) is known to be piecewise linear, it is specified by its breakpoints. To find them, one can look along the boundary, monitoring the angle between two line segments. One segment is between the current point and a point several points back along the boundary; the other is between the current point and one several points forward. When the angle between these segments reaches a maximum over some threshold, a breakpoint is declared at the current point. This scheme does not adjust breakpoint positions, and so is fast [Shirai 1975] but works best for piecewise linear input curves.

Tolerance-band solutions place a point on either side of the curve at the maximum allowable error distance, and then find the longest piece of the curve that lies entirely between parallel lines through the two points [Tomek 1974]. This method proceeds without breakpoint adjustment, and may not find the most economical set of segments (Fig. 8.2).

An approximation of a curve with a polyline of minimum length in error by at most a pixel is given in [Sklansky and Kibler 1976]. Each curve pixel is considered a square and the resulting pixel structure is four-connected. The approximation describes the shape of an elastic thread placed in the pixel structure (Fig. 8.3). The

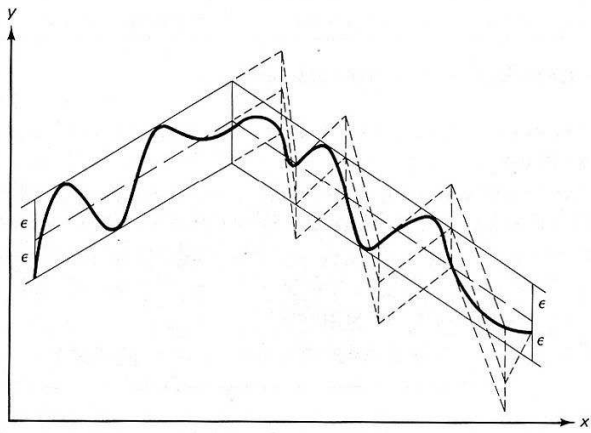


Fig. 8.2 Simple tolerance-band solution (dotted lines). Better solution (solid lines).

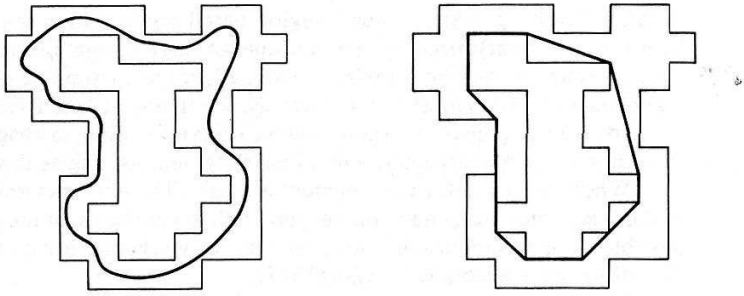


Fig. 8.3 Minimum length polyline.

method tends to have difficulties with curves that are sharp relative to the grid size.

Another scheme, [Roberts 1965] is to keep a running least-squared-error best-fit line calculation for points as they are merged into segments [Appendix 1]. When the residual (error) of a point goes over some threshold or the accumulated error for a segment exceeds a threshold, a new segment is started. Difficulties arise here because the concept of a breakpoint is nonexistent; they just occur at the intersections of the best-fit lines, and without a phase of adjusting the set of points to be fit by each line (analogous to breakpoint adjustment), they may not be intuitively appealing.

Generally, one-pass merging schemes do not produce the most satisfying polylines possible under all conditions. Part of the problem is that breakpoints are only introduced after the fit has deteriorated, usually indicating that an earlier breakpoint would have been desirable.

In a *splitting* scheme, segments are divided (usually into two parts) as long as they fail some fitting condition [Duda and Hart 1973; Turner 1974]. Algorithm 8.1 provides an example.

Algorithm 8.1: Curve Approximation

1. Given a curve as in Fig. 8.4a, draw a straight line between its end points (Fig. 8.4b).
2. For every point on the curve, compute its perpendicular distance to the approximating (poly)line. If it is everywhere within some tolerance, exit.
3. Otherwise, pick the curve point farthest from the approximating (poly)line, make it a new breakpoint (Fig. 8.4c) and replace the relevant segment of polyline with two new line segments.
4. Recursively apply the algorithm to the two new segments (Fig. 8.4d).

A straightforward extension is needed to deal with the case of curve segments parallel to the approximating one at maximum distance (Fig. 8.4e).

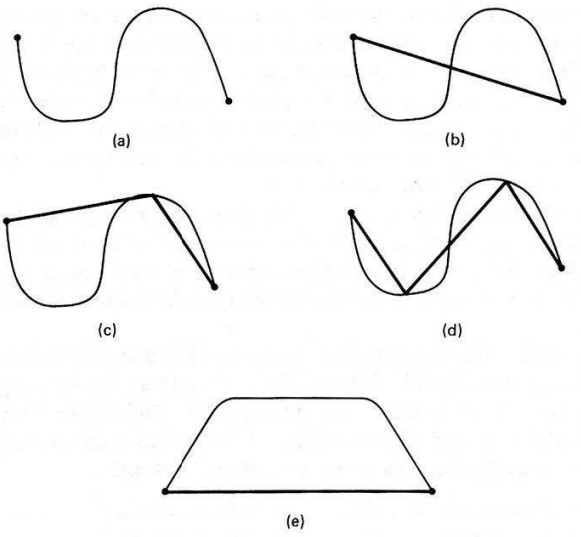


Fig. 8.4 Stages in the recursive linear segmenter (see text).

The area of a polygon may easily be computed from its polyline representation [Roberts 1965]. For a closed polyline of n points $(x(i), y(i))$, $i=0, \ldots, n-1$, labeled clockwise around a polygonal boundary, the area of the polygon is

$$\frac{1}{2} \sum_{i=0}^{n-1} (x_{i+1}y_i - x_i y_{i+1}) \tag{8.1}$$

where subscript calculations are modulo n. This formula can be proved by considering it as the sum of (signed) areas of triangles, each with a vertex at the origin, or of parallelograms constructed by dropping perpendiculars from the polyline points to an axis. This method specializes to chain codes, which are a limiting case of polylines.

8.2.2 Chain Codes

Chain codes [Freeman 1974] consist of line segments that must lie on a fixed grid with a fixed set of possible orientations. This structure may be efficiently represented because of the constraints on its construction. Only a starting point is represented by its location; the other points on a curve are represented by successive displacements from grid point to grid point along the curve. Since the grid is uniform, direction is sufficient to characterize displacement. The grid is usually considered to be four- or eight- connected; directions are assigned as in Fig. 8.5, and each direction can be represented in 2 or 3 bits (it takes 18 bits to represent the starting point in a 512×512 image).

Chain codes may be made position-independent by ignoring the "start point." If they represent closed boundaries they may be "start point normalized" by choosing the start point so that the resulting sequence of direction codes forms

an integer of minimum magnitude. These normalizations may help in matching. Periodic correlation (Section 3.2.1) can provide a measure of chain code similarity. The chain codes without their start point information are considered to be periodic functions of "arc length." (Here the arc length is just the number of steps in the chain code.) The correlation operation finds the (arc length) displacement of the functions at which they match up best as well as quantifying the goodness of the match. It can be sensitive to slight differences in the code.

The "derivative" of the chain code is useful because it is invariant under boundary rotation. The derivative (really a first difference mod 4 or 8) is simply another sequence of numbers indicating the relative direction of chain code segments; the number of left hand turns of $\pi/2$ or $\pi/4$ needed to achieve the direction of the next chain segment.

Chain codes are also well-suited for merging of regions [Brice and Fennema 1970] using the data structure described in Section 5.4.1. However, the pleasant properties for merging do not extend to union and intersection. Chain codes lend themselves to efficient calculation of certain parameters of the curves, such as area. Algorithm 8.2 computes the area enclosed by a four-neighbor chain code.

Algorithm 8.2: Chain Code Area

Comment: For a four-neighbor code (0: $+x$, 1: $+y$, 2: $-x$, 3: $-y$) surrounding a region in a counterclockwise sense, with starting point (x, y):

```
begin Chain Area;
1. area := 0;
2. yposition := y;
3. For each element of chain code
   case element-direction of
   begin case
     [0] area := area-yposition;
     [1] yposition := yposition + 1;
     [2] area := area + yposition;
     [3] yposition := yposition - 1;
   end case;
end Chain Area;
```

To merge two region boundaries is to remove any boundary they share, obtaining a boundary for the region resulting from gluing the two abutting regions together. As we saw in Chapter 5, the chain codes for neighboring regions are closely related at their common boundary, being equal and opposite in a clearly defined sense (for N-neighbor chain codes, one number is equal to the other plus $N/2$ modulo N (see Chapter 5). This property allows such sections to be identified readily, and easily scissored out to give a new merged boundary. As with polylines, it is not immediately obvious from a chain-coded boundary and a point whether the point is within the boundary or outside. Many algorithms for use with chain code representations may be found in [Freeman 1974; Gallus and Neurath 1970].

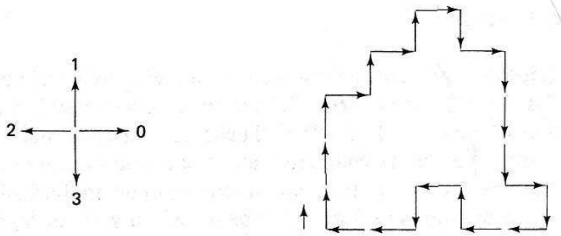


Fig. 8.5 (a) Direction numbers for chain code elements. (b) Chain code for the boundary shown. (c) Derivative of (b).

8.2.3 The ψ -s Curve

The $\psi-s$ curve is like a continuous version of the chain code representation; it is the basis for several measures of shape. ψ is the angle made between a fixed line and a tangent to the boundary of a shape. It is plotted against s, the arc length of the boundary traversed. For a closed boundary, the function is periodic, with a discontinuous jump from 2π back to 0 as the tangent reattains the angle of the fixed line after traversing the boundary.

Horizontal straight lines in the $\psi-s$ curve correspond to straight lines on the boundary (ψ is not changing). Nonhorizontal straight lines correspond to segments of circles, since ψ is changing at a constant rate. Thus the $\psi-s$ curve itself may be segmented into straight lines [Ambler et al. 1975], yielding a segmentation of the boundary of the shape in terms of straight lines and circular arcs (Fig. 8.6).

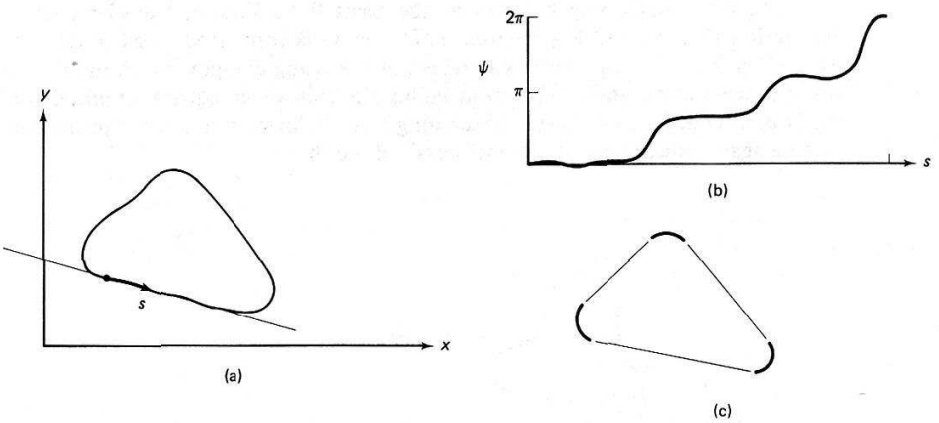


Fig. 8.6 ψ-s segmentation. (a) Triangular curve and a tangent. (b) ψ-s curve showing regions of high curvature. (c) Resultant segmentation.

8.2.4 Fourier Descriptors

Fourier descriptors represent the boundary of a region as a periodic function which can be expanded in a Fourier series. There are several possible parameterizations, summarized in [Persoon and Fu 1974]. These frequency-domain descriptions provide an increasingly accurate characterization of shape as more coefficients are included. In the infinite limit, they are unambiguous; individual coefficients are descriptive representations indicating "lobedness" of various degrees.

The boundary itself may provide the parameters for the Fourier transform as shown in Fig. 8.7. The parameterization of Fig. 8.7 gives the following series expansions:

$$\mathbf{x}(\overset{S}{p}) = \Sigma \mathbf{X}_k e^{jkw_0 \overset{S}{s}} \qquad w_o = 2\pi/P, \quad P = \text{perimeter} \tag{8.2}$$

where the discrete Fourier coefficients $\mathbf{X}_k$ are given by

$$\mathbf{X}_k = \frac{1}{P} \int_0^P \mathbf{x}(s) e^{-jkw_o s} ds \tag{8.3}$$

A common feature for the Fourier descriptors is that typically the general shape is given rather well by a few of the low-order terms in the expansion of the boundary curve. Properly parameterized, the coefficients are independent of size, translation, and rotation of the shape to be described. The descriptors do not lend themselves well to reconstruction of the boundary; for one thing, the resulting curve may not be closed if only a finite number of coefficients is used for the reconstruction.

The $\psi - s$ curve may be used as the basis for a Fourier transform shape description [Barrow and Popplestone 1971]. $\psi(s)$ is converted to $\phi(s)$: $\phi(s) = \psi(s) - 2\pi\ s/P$. This operation subtracts out the rising component. A number of shape-indicating numbers arise from taking the root-mean-square amplitudes of the Fourier components of $\phi(s)$, discarding phase information. The shape descriptors are again indicative of the "lobedness" of the shape.

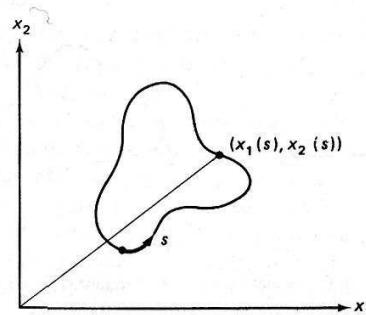


Fig. 8.7 Parameterization for Fourier Series Expansion.

8.2.5 Conic Sections

Polynomials are a natural choice for curve representation, and certain polynomials of degree 2 (namely, circles and ellipses) are closed curves and hence define regions. Circles may be represented with three parameters, ellipses by five, and general conics by six. Thus the coefficients or parameters of conic sections are terse representations. Conics are often good models for physical curves such as the edges of manufactured objects.

Conics are commonly used to represent general curves approximately [Paton 1970]. Conics have some annoying properties, however; an important one is the difficulty of producing a well-behaved conic from noisy data to be fitted. Unless one is careful in defining the error measure [Turner 1974], a "least-squared error" fit of a conic to data points yields a conic which is a nonintuitive shape or even of a surprising type (such as a hyperbola when an ellipse was expected). Conic representations and algorithms are explored in Appendix 1.

8.2.6 B-Splines

Interpolative techniques may be used to yield approximate representations. B-splines are a popular choice of piecewise polynomial interpolant. Introduced in computer aided design and computer graphics, these classes of curves provide adequate aesthetic content for much design and also have many useful analytic properties. Usually, the fact that the curves are "interpolating" is not very relevant. What is relevant is that they have predictable properties which make them easy to manipulate in image processing, that they "look good" to human beings, that they closely approximate curves of interest in nature, and so forth. Several schemes exist for constructing complex curves that are useful in geometric modeling, and detailed expositions are to be found in [deBoor 1978; Barnhill and Riesenfeld 1974]. The B-spline formulation is one of the simplest that still has properties useful for interactive modeling and the extraction from raw data.

B-splines are piecewise polynomial curves which are related to a *guiding polygon.* Cubic polynomials are the most frequently used for splines since they are the lowest order in which the curvature can change sign. An example of the relationship between the guiding polygon and its spline curve is shown in Fig. 8.8. Splines are useful in computer vision because they allow accurate, manipulable internal models of complex shapes. The models may be used to guide and monitor segmentation and recognition tasks. Interactive generation of complex shape models is possible with B-splines, and the fact that the complex spline curves have terse representations (as their guiding polygons) allows programs to manipulate them easily.

Spline approximations have good computational properties as well as good representational ones. First, they are *variation diminishing.* This means that the curve is guaranteed to "vary less" than its guiding polygon (many interpolation schemes have a tendency to oscillate between sample points). In fact, the curve is guaranteed to lie between the convex hull of groups of $n + 1$ consecutive points where n is the degree of the interpolating polynomial (Fig. 8.9.) The second advan-

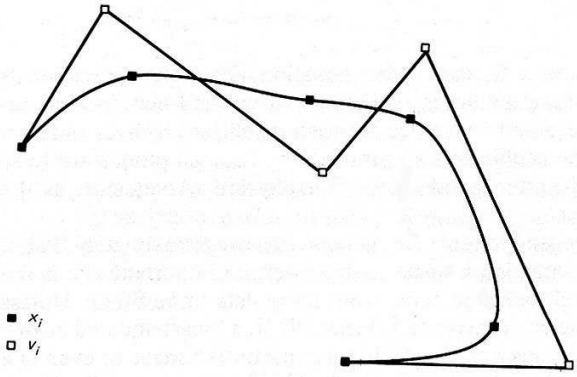


Fig. 8.8 A spline curve and its guiding polygon.

tage is that the interpolation is local; if a point on the guiding polygon is moved, the effects are intuitive and limited to nearby points on the spline. A third advantage is directly related to its use in vision; a technique for matching a spline-represented boundary curve against raw data is to search perpendicular to the spline for edges whose direction is parallel to the spline curve and location perpendicular to the spline curve. Perpendicular and parallel directions are computable directly from the parameters representing the spline.

B-Spline Mathematics

The interpolant through a given set of points $\mathbf{x}_i$, $i = 1, ..., n$ is $\mathbf{x}(s)$, a vector valued piecewise polynomial function of the parameter s; s changes uniformly between data points. For convenience, assume that $\mathbf{x}(i) = \mathbf{x}_i$, that is: s assumes integer values at data points, and $s = 1, ..., n$. Each *piece* of $\mathbf{x}(s)$ is a cubic polyno-

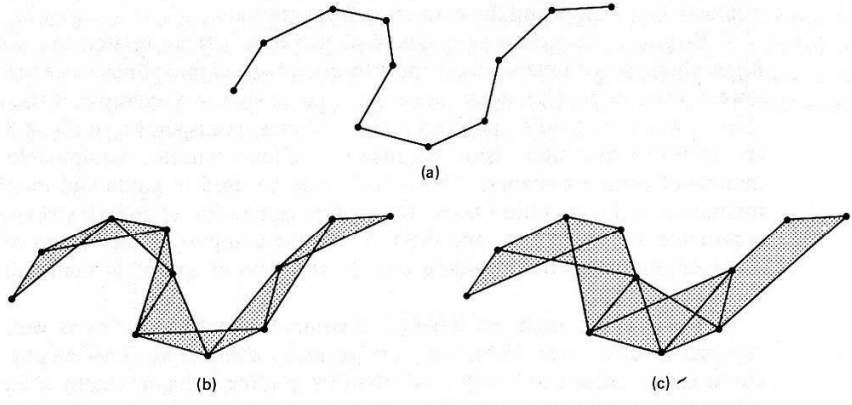


Fig. 8.9 The spline of degree n must lie in the convex hull formed by consecutive groups of $n + 1$ points. (a) $n = 1$ (linear). (b) $n = 2$ (quadratic). (c) $n = 3$ (cubic).

mial. Globally, $\mathbf{x}(s)$ has three orders of continuity across data points (i.e., up to continuity of second derivative: curvature). Formally, $\mathbf{x}(s)$ is defined as

$$\mathbf{x}(s) = \sum_{i=0}^{n+1} \mathbf{v}_i B_i(s) \tag{8.4}$$

The $\mathbf{v}_i$ are *coefficients* representing the curve $\mathbf{x}(s)$. They also turn out to be the vertices of the guiding polygon. They are a dual to the set of points $\mathbf{x}_i$; each can be derived from the other. The n data points $\mathbf{x}$ determine n $\mathbf{v}$'s. There are actually $n+2$ $\mathbf{v}$'s; the additional two coefficients are determined from *boundary conditions.* For example, if the curvature at the end points is to be 0,

$$\begin{aligned} \mathbf{v}_1 &= \frac{(\mathbf{v}_0 + \mathbf{v}_2)}{2} \\ \mathbf{v}_n &= \frac{(\mathbf{v}_{n-1} + \mathbf{v}_{n+1})}{2} \end{aligned} \tag{8.5}$$

Thus only n of the $n + 2$ coefficients are selectable.

The basis functions $B_i(s)$ are nonnegative and have a *limited support*, that is, each B_i is non-zero only for s between $i-2$ and $i+2$, as shown in Fig. 8.10. The limited support means that on a given span $(i, i+1)$ there are only four basis functions that are nonzero, namely: $B_{i-1}(s)$, $B_i(s)$, $B_{i+1}(s)$, and $B_{i+2}(s)$. Figure 8.11 shows this configuration. Thus, to calculate $\mathbf{x}(s_0)$ for some s_0, simply find in which span it resides, and then use only four terms in the summation (8.4), since there are only four basis functions which are non-zero there.

The basis functions $B_i(s)$ are, themselves, piecewise cubic polynomials and their definition depends on the relative size (in parameter space) of the spans under their support. If the spans are of uniform size (e.g., unity), then all the basis functions have the same *form* and are merely translates of each other. Moreover, each of the basis functions, on its nonzero support, is made of four pieces. So, in Fig. 8.11 in the span $(i, i+1)$ appear: the fourth piece of $B_{i-1}(s)$, the third piece of $B_i(s)$, the second piece of $B_{i+1}(s)$, and the first piece of $B_{i+2}(s)$. Call these pieces $C_{i,0}(s)$, ..., $C_{i,3}(s)$ respectively; then $x(s)$ on the interval $(i, i+1)$ is given by:

$$\begin{aligned} \mathbf{x}(s) = {} & C_{i-1,3}(s)\mathbf{v}_{i-1} + C_{i,2}(s)\mathbf{v}_i \\ & + C_{i+1,1}(s)\mathbf{v}_{i+1} + C_{i+2,0}(s)\mathbf{v}_{i+2} \end{aligned}$$

No matter what i is, $C_{i,j}$ will have the same shape; this property allows a simplification in calculations. Define four *primitive basis functions*, and interpolate along the curve by parameter shifting:

$$C_{i,j}(s) = C_j(s-i) \quad i=0, \ldots, n+1; \qquad j=0, 1, 2, 3 \tag{8.6}$$

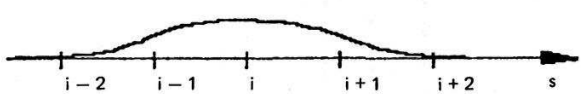


Fig. 8.10 Uniform B-spline: $B_i(s)$. Its support is non-zero only for s between $i-2$ and $i+2$.

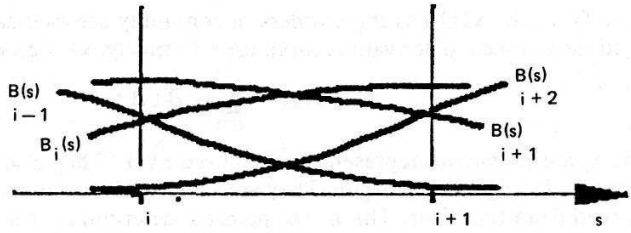


Fig. 8.11 The only four basis functions that are non-zero over the span $(i, i+1)$. Only the overlapping parts on this span are shown.

To find $\mathbf{x}(s_0)$, if s_0 is in the span $(i,\ i+1)$, use the formula:

$$\mathbf{x}(s) = \mathbf{v}_{i-1}C_3(s-i) + \mathbf{v}_i C_2(s-i) + \mathbf{v}_{i+1}C_1(s-i) + \mathbf{v}_{i+2}C_0(s-i) \tag{8.7}$$

where the $C_i(t)$ are given by:

$$C_0(t) = \frac{t^3}{6}$$

$$C_1(t) = \frac{-3t^3 + 3t^2 + 3t + 1}{6}$$

$$C_2(t) = \frac{3t^3 - 6t^2 + 4}{6}$$

$$C_3(t) = \frac{-t^3 + 3t^2 - 3t + 1}{6}$$

Formal derivations may be found in [Barnhill and Riesenfeld 1974; deBoor 1978].

Useful Formulae

The formulae may be simplified still further. $\mathbf{x}(s)$ is calculated in pieces (segments); define the segments $\mathbf{x}_i(t)$ where t ranges from 0 to 1. Then

$$\mathbf{x}_i(0) = \mathbf{x}_i \qquad \text{for } i = 1, \ldots, n-1$$

and

$$\mathbf{x}_{n-1}(1) = \mathbf{x}_n \tag{8.8}$$

In matrix notation, and explicitly calculating the definition of the cubic polynomials $C_i(t)$,

$$\mathbf{x}_i(t) = [t^3,\ t^2,\ t,\ 1][C][\mathbf{v}_{i-1},\ \mathbf{v}_i,\ \mathbf{v}_{i+1},\ \mathbf{v}_{i+2}]^T \tag{8.9}$$

where $[C]$ is the matrix:

$$\frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

The ith column in the matrix $[C]$ in Eq. (8.9) above is the coefficients of the cubic polynomial $C_i(t)$ $(i = 0, 1, 2, 3)$.

There is a distinction between *open* and *closed* curves. For open curves the boundary conditions must be used to solve for the two additional coefficients, as above. For closed curves, simply

$$\mathbf{v}_0 = \mathbf{v}_n \qquad \text{and} \qquad \mathbf{v}_{n+1} = \mathbf{v}_1 \tag{8.10}$$

The relation between the different $\mathbf{v}_i$ and $\mathbf{x}_i$ is summarized as follows. For open curves with zero curvature at the endpoints:

$$\begin{bmatrix} 6 & 0 & & & & & & 1 \\ 1 & 4 & 1 & & & & & \\ & \cdot & & \cdot & & & & \\ & & \cdot & & \cdot & & & \\ & & & & 1 & 4 & 1 & \\ & & & & & 0 & 6 & \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{v}_{n-1} \\ \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{x}_{n-1} \\ \mathbf{x}_n \end{bmatrix}$$

and for closed curves:

$$\begin{bmatrix} 4 & 1 & & & & & & 1 \\ 1 & 4 & 1 & & & & & \\ & \cdot & & \cdot & & & & \\ & & \cdot & & \cdot & & & \\ & & & & 1 & 4 & 1 & \\ 1 & & & & & 4 & 4 & \end{bmatrix} \begin{bmatrix} \mathbf{v}_0 \\ \mathbf{v}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{v}_{n-1} \\ \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{x}_{n-1} \\ \mathbf{x}_n \end{bmatrix} \tag{8.11}$$

Equation (8.10) gives the relationship between the points on the guiding polygon and the points on the spline. It may be derived from Eq. (8.9) with $t = 0$ (see exercises). To interpolate between these points, use a value of t between the extremes of 0 and 1. Choosing $t = k\,dt$ for $k = 0, \ldots, n$ where $n\,dt = 1$ and substituting into Eq. (8.9) yields

$$\mathbf{x}_i(k\,dt) = [(k\,dt)^3 (k\,dt)^2 (k\,dt) 1]\,[C]\,[\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1}, \mathbf{v}_{i+2}]^I \tag{8.12}$$

This can be decomposed [Wu et al. 1977; Gordon 1969] into the following equation.

$$\mathbf{x}_i(k\,dt) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & 1 \end{bmatrix}^k \begin{bmatrix} 6 & & & \\ -6 & 2 & & \\ 1 & -1 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} dt^3 \\ dt^2 \\ dt \\ 1 \end{bmatrix} [C] \begin{bmatrix} \mathbf{v}_{i-1} \\ \mathbf{v}_i \\ \mathbf{v}_{i+1} \\ \mathbf{v}_{i+2} \end{bmatrix} \tag{8.13}$$

The tangent at a curve is obtained by differentiation:

$$\mathbf{x}'_i(k\,dt) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} 1 & & \\ 1 & 1 & \\ 1 & 1 & 1 \end{bmatrix}^k \begin{bmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3dt^2 \\ 2dt \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_{i-1} \\ \mathbf{v}_i \\ \mathbf{v}_{i+1} \\ \mathbf{v}_{i+2} \end{bmatrix} \tag{8.14}$$

8.2.7 Strip Trees

In many computational problems there are space-time trade-offs. A nonredundant explicit representation for a general discrete curve, such as a chain code, is terse but may be difficult to use for certain computations. On the other hand, a representation for curves may take up much space but allow operations on those curves be very efficient. A representation with the latter property is *strip trees* [Ballard 1981]. Strip trees are closed under intersection and union operations, and these operations may be efficiently implemented.

A strip tree is a binary tree. The datum at each node is a eight-tuple, of which six entries define a strip (rectangle) and two denote addresses of the sons (if any). Thus each strip is defined by a six-tuple $S(\mathbf{x}_b, \mathbf{x}_e, \mathbf{w})$ as shown in Fig. 8.12. (Only five parameters are necessary to define an arbitrary rectangle, but the redundant representation proves useful in union and intersection algorithms to follow.)

The tree can be created from any curve by the following recursive procedure, which is very similar to Algorithm 8.1.

Algorithm 8.3: Making a Strip Tree

Find the smallest rectangle with a side parallel to the line segment $[\mathbf{x}_0, \mathbf{x}_n)$ that just covers all the points. This rectangle is the datum for the root node of a tree. Pick a point $\mathbf{x}_k$ that touches one of the sides of the rectangle. Repeat the above process for the two sublists $[\mathbf{x}_0, \ldots, \mathbf{x}_k)$ and $[\mathbf{x}_k, \ldots, \mathbf{x}_n)$. These become sons of the root node. Repeat the process until the approximation is accurate enough.

The half-open interval facilitates the computations to follow. In the example above the point $\mathbf{x}_k$ explicitly appears in both subtrees but implementationally need not be part of the left one. Figure 8.13 shows the strip tree construction process.

Intersecting Two Curves via Strip Trees

Consider what happens when a strip from one tree intersects a strip from another, as shown in Fig. 8.14. If the strips do not intersect, the underlying curves

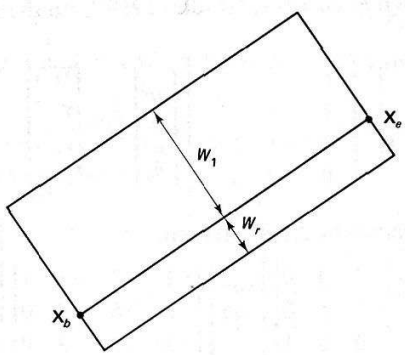


Fig. 8.12 Strip definition.

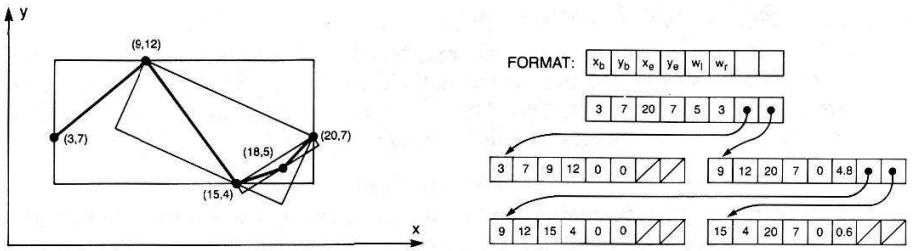


Fig. 8.13 Strip tree construction process.

do not intersect. If the strips do intersect, the underlying curves may or may not. To determine which, the computation may be applied recursively. At the leaf level of the tree defined as the *primitive* level, the problem can always be resolved.

Algorithm 8.4: Intersecting Two Strip Trees Representing Curves

Boolean Procedure TreeInt ($T1$, $T2$, L)
Begin
case intersection type of two strips $T1$ and $T2$ *of*
begin case
[primitive] *return* (true)
[null] *return* (false)
[possible] *If* $T2$ is the "fatter" strip
return (TreeInt($T1$,LSon($T2$)) or TreeInt($T1$,RSon($T2$))
Else return (TreeInt(LSon($T1$),$T2$) or TreeInt(RSon($T1$),$T2$));
end case;
end;

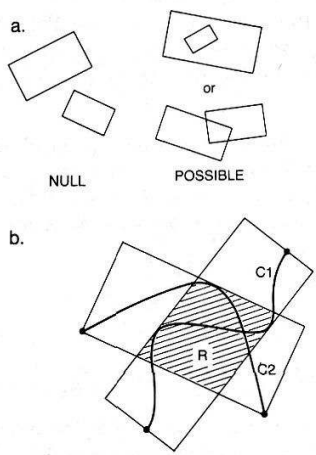


Fig. 8.14 Types of strip intersections. (a) Two kinds of intersections: NULL on the left; various POSSIBLE intersections on the right. (b) Under certain conditions the underlying curves must intersect.

The "Union" of Two Strip Trees

The "union" of two strip trees may be defined as a strip that covers both of the two root strips. The two curves defined by $[\mathbf{x}'_0, \ldots, \mathbf{x}'_n)$, $[\mathbf{x}''_0, \ldots, \mathbf{x}''_m)$ are treated as two concatenated lists. That is, the resultant ordering is such that $\mathbf{x}_0 = \mathbf{x}'_0$, $\mathbf{x}_{m+n+1} = \mathbf{x}''_m$. This construction is shown in Fig. 8.15.

Closed Curves Represented by Strip Trees

A region may be represented by its (closed) boundary. The strip-tree construction method described in Algorithm 8.3 works for closed curves and, incidentally, also for self-intersecting curves. Furthermore, if a region is not simply connected (has "holes") it can still be represented as a strip tree which at some level has connected primitives.

Many useful operations on regions can be carried out with strip trees. Examples are intersection between a curve and a region and intersecting two regions. Another example is the determination of whether a point is inside a region. Roughly, if any semi-infinite line terminating at the point intersects the boundary of the region an odd number of times, the point is inside. The implied algorithm is computationally simplified for strip trees in the following manner:

> *Point Membership Property.* To decide whether a point z is a member of a region represented by a strip tree, compute the number of nondegenerate intersections of the strip tree with any semi-infinite strip L which has $\|w\| = 0$ and emanates from z. If this number is odd, the point is inside the region.

This is because for clear intersections the underlying curves may intersect more than once but must intersect an odd number of times. A potential difficulty exists when the strip L is tangent to the curve. To overcome this difficulty in practice, a different L may be used.

Intersecting a Curve with a Region

The strategy behind intersecting a strip tree representing a curve with a strip tree representing a region is to create a new tree for the portion of the curve that overlaps the region. This can be done by trimming the original curve strip tree. Trimming is done efficiently by taking advantage of an obvious property of the intersection process:

> *Pruning Property*: Consider two strips S_C from T_C and S_a from T_a. If the intersection of S_C with T_a is null, then (a) if any point on S_C is inside T_a, the entire tree whose root strip is S_C is inside or on T_a, and (b) if any point on S_C is outside of T_a then the entire tree whose root strip is S_C is outside T_a.

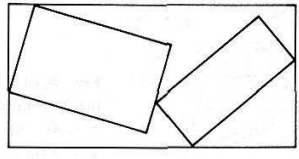


Fig. 8.15 Construction for "union" of strip trees representing two curves.

This leads to the Algorithm 8.5 for curve-region intersection using trees. If the curve strip is "fatter" (i.e., has more area), copy the node and resolve the in-

tersection at lower levels. In the converse case prune the tree sequentially by first intersecting the resultant pruned tree with the right region strip.

Algorithm 8.5: Curve–Region Intersection

```
comment A Reference Procedure returns a pointer;
reference procedure CurveRegionInt(T1,T2)
 begin
 A := T2;
 comment R is a global used by CRInt;
 return (CRInt(T1,T2));
 end;

reference procedure CRInt(T1,T2)
 begin
 begin Case StripInt(T1,T2) of
  [Null or Primitive]
   if intersection (T1,R,TRUE) = null then
    if Inside(T1,R) then return (T1)
    else return (null);
   else return (T1);
   [Possible] if T1 is "fatter" then
   begin
    NT := NewRecord;
    x_b(NT) := x_b (T);
    x_e(NT) := x_e (T);
    w_l(NT) := w_l(T);
    w_r(NT) := w_r(T);
    LSon(NT) := CRInt (LSon(T1),T2);
    RSon(NT) := CRInt (RSon(T1),T2);
    return(NT);
    end
   else comment T2 is "fatter"
    Return (CRInt(CRInt(T1,LSon(T2)),RSon(T2)));
   end;
end Case;
end;
```

The problem of intersecting two regions can be decomposed into two curve-region intersection problems (Fig. 8.16). Thus algorithm 8.5 can also be used to solve the region-region intersection problem.

8.3 REGION REPRESENTATIONS

8.3.1 Spatial Occupancy Array

The most obvious and quite a useful representation for a region on a raster is a membership predicate $p(x, y)$ which takes the value 1 when point (x, y) is in the

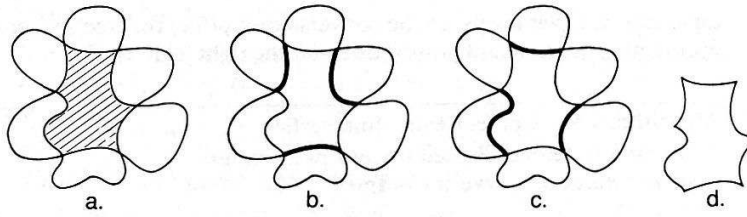


Fig. 8.16 Decomposition of Region-Region Intersection. (a) Desired result. (b) Portion of boundary generated by treating three-lobed region as a curve. (c) Portion of boundary generated by treating five-lobed region as a curve. (d) Result of union operation.

region and the value 0 otherwise. One easy way to implement such a function is with a *membership array*, an array of 1's and 0's with the obvious interpretation. Such arrays are quicky interrogated and also quite easily unioned, merged and intersected by AND and OR operations, applied elementwise on the operand arrays. The disadvantages of this representation are that it requires much space and does not represent the boundary in a useful way.

8.3.2 *y*-Axis

A representation that is more compact and which offers reasonable algorithms for intersection, merging, and union is the *y*-axis representation [Merrill 1973]. This is a run-length encoding of the membership array, and as such it provides no explicit boundary information. It is a list of lists. Each element on the main list corresponds to a row of constant y in the image raster. Each row of constant y is encoded as a list of x-coordinate points; the first x point at which the region is entered while moving along that y row, then the x point at which the region is exited, then the x point at which it next is entered, and so forth. The y-rows with no region points are omitted from the main list. Thus, in a notation where successive levels of sublist are surrounded by successive levels of parentheses, the y-axis encoding of a region is shown in Fig. 8.17; here the first element of each sublist is the y coordinate, followed by a list of "into" and "out of" x coordinates. Where a y coordinate con-

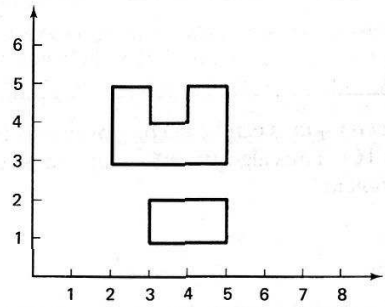


Fig. 8.17 *y*-axis region representation.

tains an isolated point in the region, this point is repeated in the x-axis representation, as shown by the example in Fig. 8.17. Thus "lines" (regions of unit width) can be easily (although not efficiently) represented in this system.

Union and intersection are implemented on y-axis representations as merge-like operations which take time linearly proportional to the number of y rows. Two instances of y-axis representations and the representation of their union are shown in Fig. 8.18. Note that the union amounts to a merge of x elements along rows organized within a merge of rows themselves.

The y-axis representation is wasteful of space if the region being represented is long, thin, and parallel to the y axis. In this case one is invited to encode it in x-axis format, in an obvious extension. Working with mixed x-axis and y-axis formats presents no conceptual difficulties, but considerable loss of convenience.

8.3.3 Quad Trees

Quad trees [Samet 1980] are a useful encoding of the spatial occupancy array. The easiest way to understand quad trees is to consider pyramids as an intermediate representation of the binary array. Figure 8.19 shows a pyramid (Section 3.7) made from the base image (on the left). Each pixel in images above the lowest level has one of three values, BLACK, WHITE, or GRAY. A pixel in a level above the base is BLACK or WHITE if all its corresponding pixels in the next lower level are BLACK or WHITE respectively. If some of the lower level pixels are BLACK and others are WHITE, the corresponding pixel in the higher level is GRAY.

Such a pyramid is easy to construct. To convert the pyramid to a quad tree, simply search the pyramid recursively from the top to the base. If an array element in the pyramid is either BLACK or WHITE, form a terminal node of the corresponding type. Otherwise, form a GRAY node with pointers to the results of

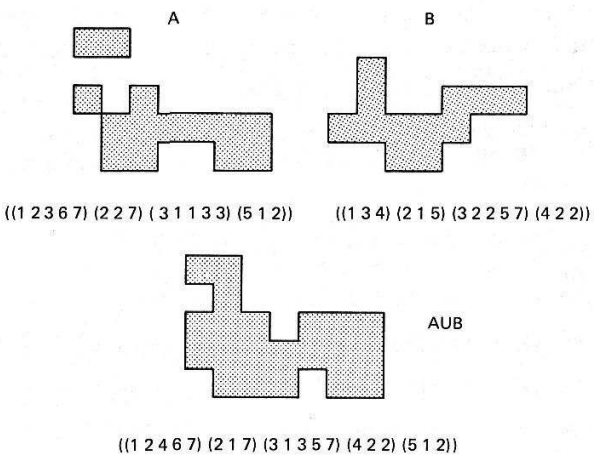


Fig. 8.18 Two point sets A, B, and $A \cup B$, with their y-axis representations.

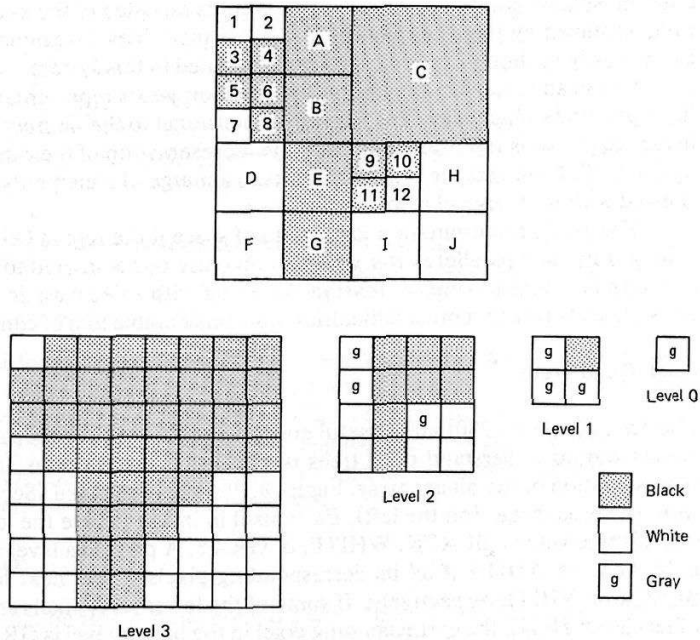


Fig. 8.19 Pyramid used in quad tree construction. Letters correspond to pixels in the pyramid that are either BLACK or WHITE.

the recursive examination of the four elements at the next level in the tree (Algorithm 8.6).

Algorithm 8.6: Quad Tree Generation

```
Reference Procedure QuadTree (integer array pyramid; integer x, y, level);
Comment NW, NE, SW, SE are fields denoting the sons of a quadtree node;
Newnode(P);
TYPE(P) := Pyramid(IND (x,y,Level));
if TYPE(P) = BLACK or WHITE then return (P)
else begin
  SW(P):=QuadTree(Pyramid, 2*x, 2*y, Level + 1);
  SE(P):=QuadTree(Pyramid, 2*x + 2*Level, 2*y, Level + 1);
  NW(P):=QuadTree(Pyramid, 2*x, 2*y + 2*Level, Level + 1);
  NE(P):=QuadTree(Pyramid, 2*(x + Level), 2*(y + Level), Level + 1);
  return (P)
  end;
```

Here an implementational point is that the entire pyramid fits into a linear array of size $2(2^{2\times level})$. IND is an indexing function which extracts the appropriate value given the x, y and level coordinates. The reader can apply this algorithm to the example in Fig. 8.19 to verify that it creates the tree in Fig. 8.20.

The quad tree can be created directly from the base of the pyramid, but the algorithm is more involved. This is because proceeding upward from the base, one must sometimes defer the creation of black and white nodes. This algorithm is left for the exercises [Samet 1980].

Many operations on quad trees are simple and elegant. For example, consider the calculation of area [Schneier 1979]:

Algorithm 8.7: Area of a Quad Tree

```
Integer Procedure Area (reference QuadTree; integer height)
  Begin
  Comment NW, NE, SW, SE are fields denoting the sons of
  a quadtree node;
  BlackArea := 0;
  if TYPE(QuadTree) = GRAY then
   for I in the set {NW, NE, SW, SE} do
    BlackArea = BlackArea + Area(I(QuadTree), height-1)
  else if TYPE(QuadTree) = BLACK then
   BlackArea = BlackArea + 2^2*height;
  return(BlackArea)
  end;
```

Other examples may be found in the References and are pursued in the Exercises.

The quad tree and the associated pyramid have two related disadvantages as a representation. The first is that the resolution cannot be extended to finer resolution after a grid size has been chosen. The second is that operations between quad

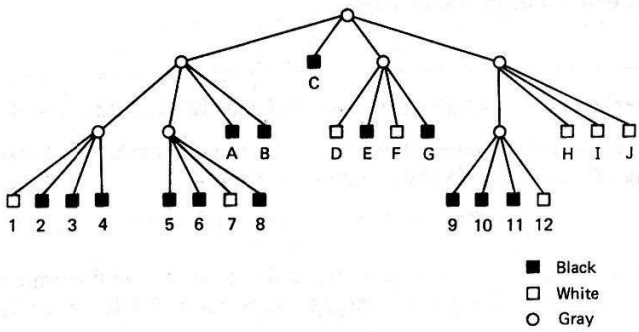


Fig. 8.20 Quad tree for the example in Fig. 8.19.

trees tacitly assume that their pyramids are defined on the same grids. The grids cannot be shifted or scaled without cumbersome conversion routines.

8.3.4 Medial Axis Transform

If the region is made of thin components, it can be well described for many purposes by a "stick-figure" *skeleton.* Skeletons may be derived by thinning algorithms that preserve connectivity of regions; the medial axis transform (MAT), of [Blum 1973; Marr 1977] is a well-known thinning algorithm.

The skeleton is defined in terms of the distance of a point $\mathbf{x}$ to a set A:

$$d_s(\mathbf{x}, A) = \inf\{d(\mathbf{x}, \mathbf{z}) | \mathbf{z} \text{ in } A\} \tag{8.15}$$

Popular metrics are the Euclidean, city block, and chessboard metrics described in Chapter 2.

Let B be the set of boundary points. For each point P in a region, find its closest neighbors (by some metric) on the region boundary. If *more than one* boundary point is the minimum distance from $\mathbf{x}$, then $\mathbf{x}$ is on the skeleton of the region. The skeleton is the set of pairs $\{\mathbf{x}, d_s(\mathbf{x}, B)\}$ where $d_s(\mathbf{x}, B)$ is the distance from $\mathbf{x}$ to the boundary, as defined above (this is a definition, not an efficient algorithm.) Since each $\mathbf{x}$ in the skeleton retains the information on the minimum distance to the boundary, the original region may be recovered (conceptually) as the union of "disks" (in the proper metric) centered on the skeleton points.

Some common shapes have simply structured medial axis transform skeletons. In the Euclidean metric, a circle has a skeleton consisting of its central point. A convex polygon has a skeleton consisting of linear segments; if the polygon is nonconvex, the segments may be parabolic or linear. A simply connected polygon has a skeleton that is a tree (a graph with no cycles). Some examples of medial axis transform skeletons appear in Fig. 8.21.

The figure shows that the skeleton is sensitive to noise in the boundary. Reducing this sensitivity may be accomplished by smoothing the boundary, using a polygonal boundary approximation, or including only those points in the skeleton that are greater than some distance from the boundary. The latter scheme can lead to disconnected skeletons.

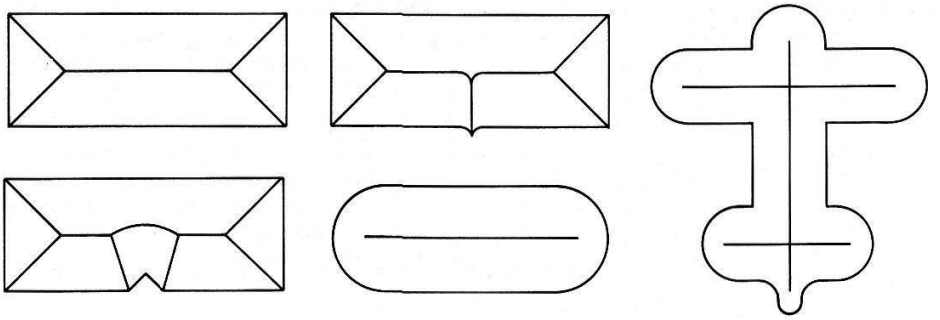
Algorithm 8.8: *Medial Axis Transformation* [Rosenfeld and Kak 1976]

Let region points have value 1 and exterior points value 0. These points define an image $f^0(\mathbf{x})$. Let $f^k(\mathbf{x})$ be given by

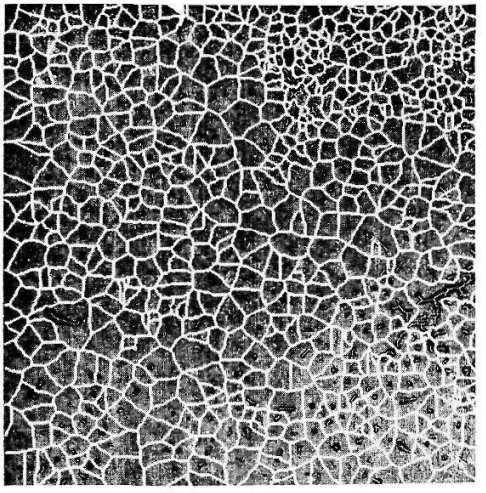
$$f^k(\mathbf{x}) = f^0(\mathbf{x}) + \min_{d(x,z)\leqslant 1} [f^{k-1}(\mathbf{z})], \qquad k > 0$$

The points $f^k(\mathbf{x})$ will converge when k is equal to the maximum thickness of the region. Where $f^k(\mathbf{x})$ has converged, the skeleton is defined as all points $\mathbf{x}$ such that

$$f^k(\mathbf{x}) \geqslant f^k(\mathbf{z}), \qquad d(\mathbf{x}, \mathbf{z}) \leqslant 1.$$



(a)



(b)

Fig. 8.21 Medial Axis Transform skeletons (a), and the technique applied to human cell nuclei (b). Shown in (b) are both the "normal" skeleton obtained by measuring distances interior to the boundaries, and the exo-skeleton, obtained by measuring distances exterior to the boundary.

This algorithm can produce disconnected skeletons for excursions or lobes off the main body of the region. Elegant thinning algorithms to compute skeletons are given in [Pavlidis 1977].

8.3.5 Decomposing Complex Regions

Much work has been done on the decomposition of point sets (usually polygons) into a union of convex polygons. Such convex decompositions provide structural analysis of a complex region that may be useful for matching different point sets.

An example of the desired result in two dimensions is presented here, and the interested reader may refer to [Pavlidis 1977] for the details. Such a decomposition is not unique in general and in three dimensions, such difficulties arise that the problem is often called ill-formed or intractable [Voelcker and Requicha 1977].

The shapes of Fig. 8.22 have three "primary convex subsets" labeled *X, Y,* and *Z*. They form different numbers of "nuclei" (roughly, intersection sets). The shape is described by a graph that has nodes for nuclei and primary convex subsets and an arc between intersecting sets (Fig. 8.22c). Without nodes for the nuclei (i.e., if only primary convex subsets and their intersections are represented), regions with different topological connectedness can produce identical graphs (Fig. 8.22b).

8.4 SIMPLE SHAPE PROPERTIES

8.4.1 Area

The *area* of a region is a basic descriptive property. It is easily computed from curve boundary representations (8.3.1) and thus also for chain codes (8.3.2); their con-

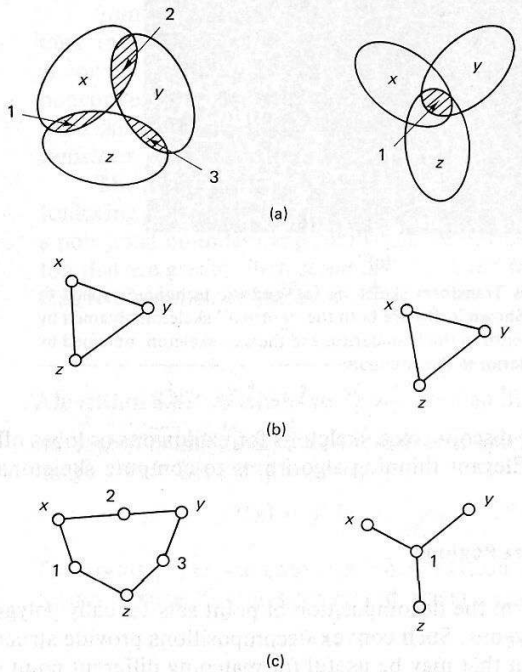


Fig. 8.22 Decomposition of polygon into primary convex subsets and nuclei (see text).

tinuous analog is also useful. Consider a curve parameterized on arc length s so that points (x, y) are given by functions $(x(s), y(s))$

$$\text{area} = \int_0^P \left(x \frac{dy}{ds} - y \frac{dx}{ds}\right) ds \tag{8.16}$$

where P is the perimeter.

8.4.2 Eccentricity

There are several measures of *eccentricity*, or "elongation". One of them is the ratio of the length of maximum chord A to maximum chord B perpendicular to A (Fig. 8.23).

Another reasonable measure is the ratio of the principal axes of inertia; this measure can be based on boundary points or the entire region [Brown 1979]. An (approximate) formula due to Tenenbaum for an arbitrary set of points starts with the mean vector

$$\mathbf{x}_0 = \frac{1}{n} \sum_{\mathbf{x} \text{ in } R} \mathbf{x} \tag{8.17}$$

To compute the remaining parameters, first compute the ijth moments M_{ij} defined by

$$M_{ij} = \sum_{\mathbf{x} \text{ in } R} (x_0 - x)^i (y_0 - y)^j \tag{8.18}$$

The orientation, θ, is given by

$$\theta = \frac{1}{2} \tan^{-1}\left(\frac{2M_{11}}{M_{20} - M_{02}}\right) + n\left(\frac{\pi}{2}\right) \tag{8.19}$$

and the approximate eccentricity e is

$$e = \frac{(M_{20} - M_{02})^2 + 4M_{11}}{\text{area}} \tag{8.20}$$

8.4.3 Euler Number

The Euler number is a topological property defining the set of objects that are equivalent under "rubber-sheet" deformations of the plane. It describes the connectedness of a region, not its shape. A *connected region* is one in which all pairs of

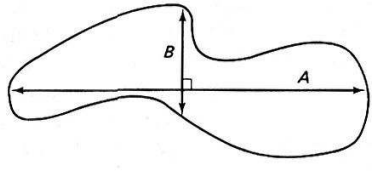


Fig. 8.23 An eccentricity measure: A / B.

points may be connected by a curve lying entirely in the region. If a complex two-dimensional object is considered to be a set of connected regions, where each one can have holes, the *Euler number* for such an object is defined as

$$\text{(number of connected regions)} - \text{(number of holes)}$$

The number of holes is one less than the connected regions in the set complement of the object.

8.4.4 Compactness

One measure of *compactness* (not compactness in the sense of point-set topology) is the ratio (perimeter^2)/area, which is dimensionless and minimized by a disk. This measure is computed easily from the chain-code representation of the boundary where the length of an individual segment of eight-neighbor chain code is given by $(\sqrt{2})$ if the (eight-neighbor) direction is odd and by 1 if the direction is even. The area is computed by a modification of Algorithm 8.2 and the perimeter may be accumulated at the same time.

For small discrete objects, this measure may not be satisfactory; another measure is based on a model of the boundary as a thin springy wire [Young et al. 1974]. The normalized "bending energy" of the wire is given by

$$E = \frac{1}{P}\int_0^P |\kappa(s)|^2 ds \tag{8.21}$$

where κ is *curvature*. This measure is minimized by a circle. E can be computed from the chain code representation by recognizing that $\kappa = d\theta/dS$, and also from the Fourier coefficients mentioned below since

$$|\kappa(s)|^2 = \left(\frac{d^2x}{ds^2}\right)^2 + \left(\frac{d^2y}{ds^2}\right)^2 \tag{8.22}$$

so that E, using Parseval's theorem, is

$$\sum_{k=-\infty}^{\infty} (kw_0)^4(|X_k|^2 + |Y_k|^2) \tag{8.23}$$

where $\mathbf{X}_k = (X_k, Y_k)$ are the Fourier descriptor coefficients in (8.2).

8.4.5 Slope Density Function

The $\psi - s$ curve can be the basis for the *slope density function* (SDF) [Nahin 1974]. The SDF is the histogram or frequency distribution of ψ collected over the boundary. An example is shown in Fig. 8.24. The SDF is flat for a circle (or in a continuous universe, any shape with a monotonically varying ψ); straight sides stand out sharply, as do sharp corners, which in a continuous universe leave gaps in the histogram. The SDF is the signature of the $\psi - s$ curve along the ψ axis.

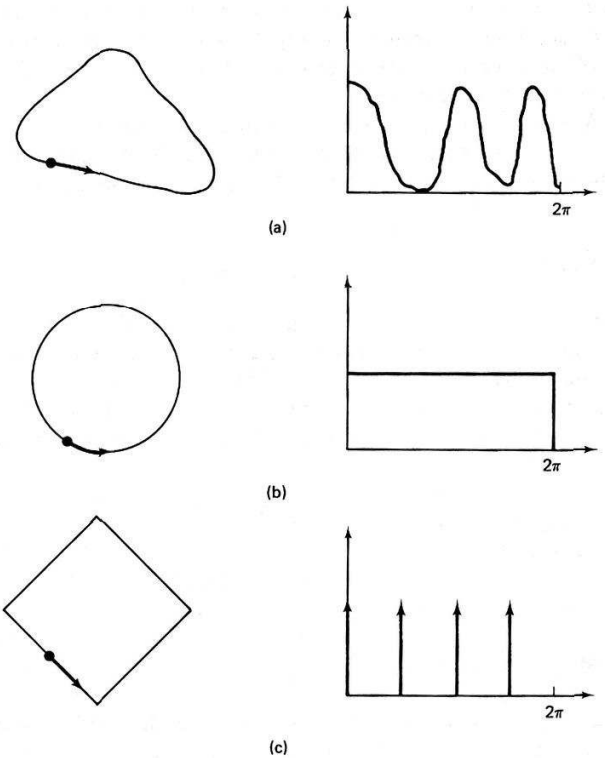


Fig. 8.24 The Slope Density Function for three curves: a triangular blob, a circle, and a square.

8.4.6 Signatures

By definition, a projection is not an information-preserving transformation. But Section 2.3.4 showed that (as with Fourier descriptors,) enough projections allow reconstruction of the region to any desired degree of accuracy. (This observation forms the basis for computer assisted tomography.)

Given a binary image $f(\mathbf{x}) = 0$ or 1, define the horizontal *signature* $p(x)$ as

$$p(x) = \int_y f(x, y) \tag{8.24}$$

$p(x)$ is simply the projection of p onto the x axis. Similarly, define $p(y)$, the vertical signature, as

$$p(y) = \int_x f(x, y) \tag{8.25}$$

Maxima and minima of signatures are often useful for establishing preliminary

landmarks in an image to reduce subsequent search effort [Kruger et al. 1972] (Fig. 8.25). If the region is not binary, but consists of a density function, Eq. (8.24) may still be used. Polar projections may be useful characterizations if the point of projection is chosen carefully.

Another idea is to provide a number of projections, $q_1, \ldots, q_n$, the ith one based on the ith sublist in each row in a y-axis-like region representation. This technique is more sensitive to non-convexities and holes than is a regular projection (Fig. 8.26).

8.4.7 Concavity Tree

Concavity trees [Sklansky 1972] represent information necessary to fill in local indentations of the boundary as far as the convex hull and to study the shape of the resultant concavities.

A region S is *convex* iff for any $\mathbf{x}_1$ and $\mathbf{x}_2$ in S, the straight line segment connecting $\mathbf{x}_1$ and $\mathbf{x}_2$ is also contained in S. The *convex hull* of an object S is the smallest H such that

$$S \subset H$$

and H is convex.

Figure 8.27 shows a region, the steps in the derivation of the concavity tree, and the concavity tree itself.

8.4.8 Shape Numbers

For closed curves and a 3-bit chain code (together with a controlled digitization scheme), many chain-coded boundaries can be given a unique *shape number* [Bri-

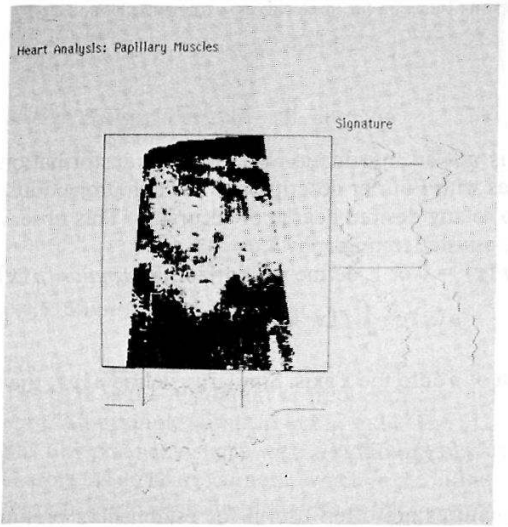


Fig. 8.25 The use of signatures to locate a left ventricle cross section in ultrasound data. (Outer curves are smoothed versions of inner signatures.)

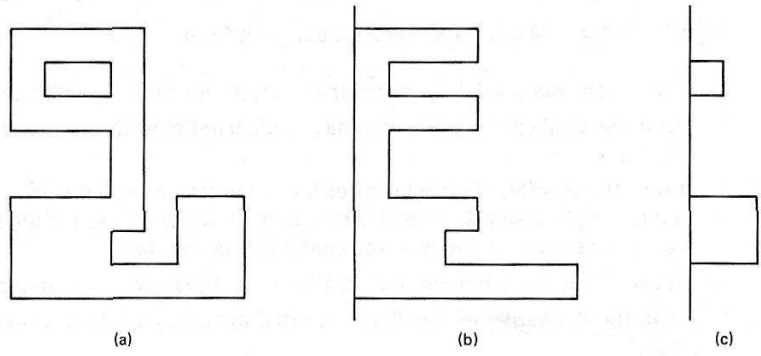


Fig. 8.26 A shape (a) and projections; from the first (b) and second (c) sublists of the y-axis representation.

biesca and Guzman 1979]. The shape number is related to the resolution of the digitization scheme. In a multiple resolution pyramid of digitization grids, every possible shape can be represented as a path through a tree. At each grid resolution corresponding to a level in the tree, there are a finite number of possible shapes. Moving up the tree, the coarser grids tend to blur distinctions between different shapes until at some resolution they are identical. This level can be used as a similarity measure between shapes. The basic idea behind shape numbers is the following. Consider all the possible closed boundaries with n chain segments. These form the possible shapes of "order n." The chain encoding for a particular boundary can be made unique by interpreting the chain-code direction sequence as a number and picking the start point that minimizes this number. Notice that the orders of shape numbers must be even on rectangular grids since a curve of odd order cannot close.

Algorithm 8.9 generates a shape number of order n.

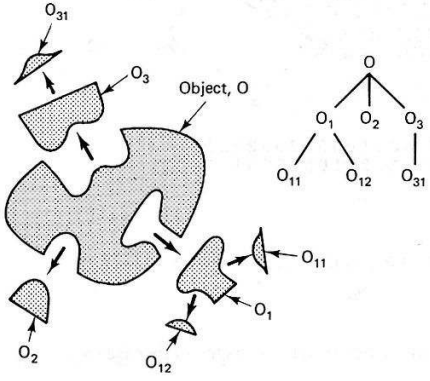


Fig. 8.27 Concavities of an object and the concavity tree.

Algorithm 8.9: Making a Shape Number of Order n

1. Choose the maximal diameter of the shape as one of the coordinate axes.
2. Find the smallest rectangle that has a side parallel to this axis and just covers the shape.
3. From the possible rectangles of order n, find the one that best approximates the rectangle in step 2. Scale this rectangle so that the length of the longest side equals that of the major axis, and center it over the shape.
4. Set all the pixels falling more than 50% inside the region to 1, and the rest to 0.
5. Find the derivative of the chain encoded boundary of the region of 1's from step 4.
6. Normalize this number by rotating the digits until the number is minimum. The normalized number is the shape number.

Figure 8.28 shows these steps.

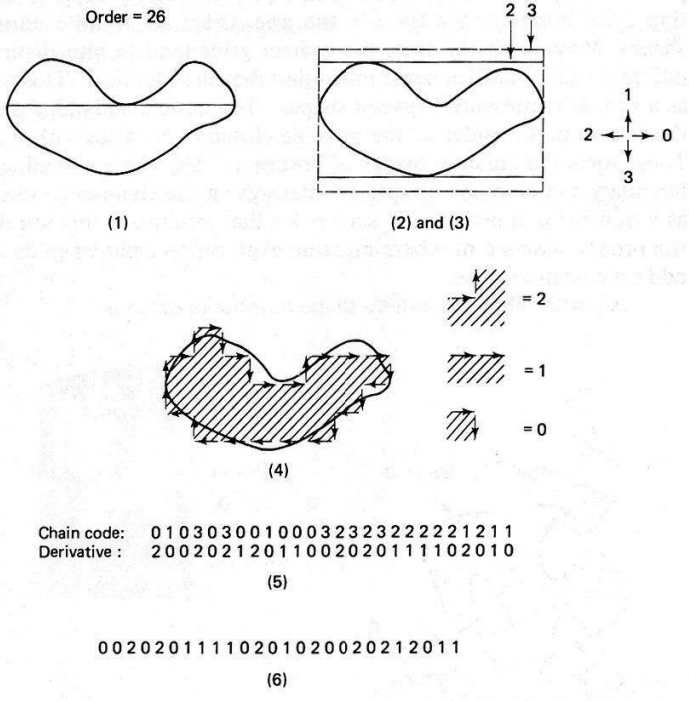


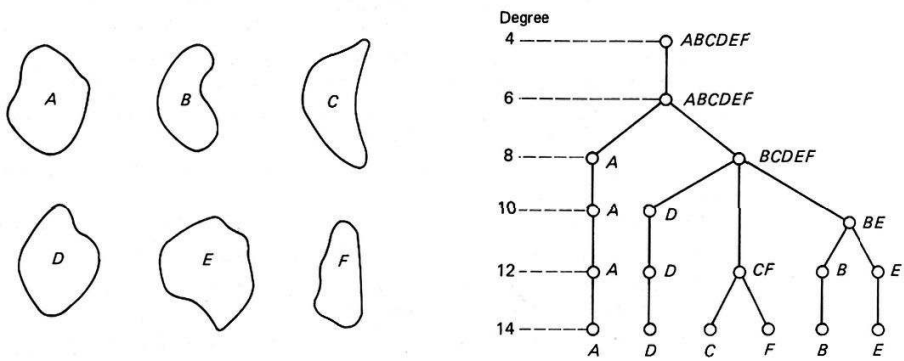
Fig. 8.28 Steps in determining a shape number (see text).

Generating a shape number of a specific order may be tricky, as there is a chance that the resulting shape number may be greater than order n due to deep concavities in the boundary. In this case, the generation procedure can be repeated for smaller values of n until a shape number of n digits is found. Even this strategy may sometimes fail. The shape number may not exist in special cases such as boundaries with narrow indentations. These features may cause step 4 in Algorithm 8.11 to fail in the following way. Even though the rectangle of step 3 was of order n, the resultant boundary may have a different order. Nevertheless, for the vast majority of cases, a shape number can be computed.

The degree of similarity for two shapes is the largest order for which their shape numbers are the same. The "distance" between two shapes is the inverse of their degree of similarity. This distance is an *ultradistance* rather than a norm:

$$\begin{aligned} d(S, S) &= 0 \\ d(S_1, S_2) &\geqslant 0 \qquad \text{for } S_1 \neq S_2 \\ d(S_1, S_3) &\leqslant \max(d(S_1, S_2), d(S_2, S_3)) \end{aligned} \tag{8.26}$$

Figure 8.29 shows the similarity tree for six shapes as computed from their shape numbers. When the shape number is well defined, it is a useful measure since it is unique (for each order), it is invariant under rotation and scale changes of an object, and it provides a metric by which shapes can be compared.



	A	B	C	D	E	F
A	∞	6	6	6	6	6
B		∞	8	8	10	8
C			∞	8	8	12
D				∞	8	8
E					∞	8
F						∞

Fig. 8.29 Six shapes, their similarity trees, and the ultradistances between the shapes.

EXERCISES

8.1 Consider a region segmentation where regions are of two types: (1) filled in and (2) with holes. Relate the number of junctions, boundaries, and filled-in regions to the Euler number.

8.2 Write a procedure for finding where two chain codes intersect.

8.3 Devise algorithms to intersect and union two regions in the y-axis representation.

8.4 Show that the number of intersections of the curves under a clear strip intersection is odd.

8.5 Modify Algorithm 8.4 to work with strip trees with varying numbers of sons.

8.6 Derive Eq. (8.9) from Eq. (8.7).

8.7 Show that Eqs. (8.12) and (8.13) are equivalent.

8.8 Given two points $\mathbf{x}_1$ and $\mathbf{x}_2$ and slopes $\phi(\mathbf{x}_1)$ and $\phi(\mathbf{x}_2)$, find the ellipse with major axis a that fits the points.

8.9 Write a procedure to intersect two regions represented by quad trees, producing the quad tree of the intersection.

8.10 Determine the shape numbers for (a) a circle and (b) an octagon. What is the distance between them?

REFERENCES

AMBLER, A. P., H. G. BARROW, C. M. BROWN, R. M. BURSTALL, and R. J. POPPLESTONE. "A versatile system for computer controlled assembly." *Artificial Intelligence 6*, 2, 1975, 129–156.

BALLARD, D. H. "Strip trees: A hierarchical representation for curves." *Comm. ACM 24*, 5, May 1981, 310–321.

BARNHILL, R. E. and R. F. RIESENFELD. *Computer Aided Geometric Design.* New York: Academic Press, 1974, 160.

BARROW, H. G. and R. J. POPPLESTONE. "Relational descriptions in picture processing." In *MI6*, 1971.

BLUM, H. "Biological shape and visual science (Part I)." *J. Theoretical Biology 38*, 1973, 205–287.

BRIBIESCA, E. and A. GUZMAN. "How to describe pure form and how to measure differences in shapes using shape numbers." *Proc.*, PRIP, August 1979, 427–436.

BRICE, C. R. and C. L. FENNEMA. "Scene analysis using regions." *Artificial Intelligence 1*, 3, Fall 1970, 205–226.

BROWN, C. M. "Two descriptions and a two-sample test for 3-d vector data." TR49, Computer Science Dept., Univ. Rochester, February 1979.

DEBOOR, C. *A Practical Guide to Splines.* New York: Springer-Verlag, 1978.

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

FREEMAN, H. "Computer processing of line drawing images." *Computer Surveys 6*, 1, March 1974, 57–98.

GALLUS, G. and P. W. NEURATH. "Improved computer chromosome analysis incorporating preprocessing and boundary analysis." *Physics in Medicine and Biology 15*, 1970, 435.

GORDON, W. J. "Spline-blended surface interpolation through curve networks." *J. Mathematics and Mechanics 18*, 10, 1969, 931–952.

HOROWITZ, S. L. and T. P. PAVLIDIS. "Picture segmentation by a tree traversal algorithm." *J. ACM 23*, 2, April 1976, 368–388.

KRUGER, R. P., J. R. TOWNE, D. L. HALL, S. J. DWYER, and G. S. LUDWICK, "Automatic radiographic diagnosis via feature extraction and classification of cardiac size and shape descriptors." *IEEE Trans. Biomedical Engineering 19*, 3, May 1972.

MARR, D. "Representing visual information." AI Memo 415, AI Lab, MIT, May 1977.

MERRILL, R. D. "Representations of contours and regions for efficient computer search." *Comm. ACM 16*, 2, February 1973, 69–82.

NAHIN, P. J. "The theory and measurement of a silhouette descriptor for image preprocessing and recognition." *Pattern Recognition 6*, 2, October 1974.

PATON, K. A. "Conic sections in automatic chromosome analysis." In *MI5*, 1970.

PAVLIDIS, T. *Structural Pattern Recognition.* New York: Springer-Verlag, 1977.

PERSOON, E. and K. S. FU. "Shape discrimination using Fourier descriptors." *Proc.*, 2nd IJCPR, August 1974, 126–130.

REQUICHA, A. A. G. "Mathematical models of rigid solid objects." TM-28, Production Automation Project, Univ. Rochester, November 1977.

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.

SAMET, H. "Region representation: quadtrees from boundary codes." *Comm. ACM 23*, 3, March 1980, 163–170.

SCHNEIER, M. "Linear time calculations of geometric properties using quadtrees." TR-770, Computer Science Center, Univ. Maryland, May 1979.

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

SKLANSKY, J. "Measuring concavity on a rectangular mosaic." *IEEE Trans. Computers 21*, 12, December 1972.

SKLANSKY, J. and D. P. KIBLER. "A theory of non-uniformly digitizing binary pictures." *IEEE Trans. SMC 6*, 9, September 1976, 637–647.

TOMEK, I. "Two algorithms for piecewise linear continuous approximation of functions of one variable." *IEEE Trans. Computers 23*, 4, April 1974, 445–448.

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

VOELCKER, H. B. and A. A. G. REQUICHA. "Geometric modelling of mechanical parts and processes." *Computer 10*, December 1977, 48–57.

WU, S., J. F. ABEL, and D. P. GREENBERG. "An interactive computer graphics approach to surface representations." *Comm. ACM 20*, 10, October 1977, 703–711.

YOUNG, I. T., J. E. WALKER, and J. E. BOWIE. "An analysis technique for biological shape I." *Information and Control 25*, 1974.

Representations of Three-Dimensional Structures 9

9.1 SOLIDS AND THEIR REPRESENTATION

We consider three general classes of representations for rigid solids:

1. Surface or boundary
2. Sweep (in general, generalized cylinders)
3. Volumetric (in general, constructive solid geometry)

The semantics of solid representations is intuitively clear but sometimes mathematically tricky. The representations have different computational properties, and readers should keep this in mind when assessing a representation for possible use. As a simple example, a surface representation can describe how an object looks; a volumetric version, which expresses the solid as a combination of subparts, may not explicitly contain information about the surface of the object. However, the solid representation may be better for matching, if it can be structured to reflect functional subparts.

Certainly we believe, as do others, that model-based vision will ultimately have to confront the issues of geometric modeling in three dimensions [Nishihara 1979]. Ultimately, nonrigid as well as rigid solids will have to be represented. The characterization of nonrigid solids presents very challenging problems.

Nonrigid solids are often a useful way to model time-varying aspects of objects. Here, again, the kind of model that is best depends heavily on the domain. For example, a useful mammal model may be one with a piecewise rigid linkage (for the skeleton) and some elastic covering (for the flesh). Computer vision in the domain of mammals, either static in various positions or actually moving, might be based on generalized cylinders (Section 9.3). However, another nonrigid domain is that of heart chambers, that change through time as the heart beats. Here the skeleton is a much less intuitive notion, so a different model of nonrigidity may apply. In most cases, nonrigid objects are modeled as parameterized rigid objects. In

the example of the human figure, the parameters may be joint angles for linkages representing the skeleton.

The last part of this chapter deals with understanding line drawings, an influential and well-publicized subfield of computer vision. This seemingly simple and accessible domain avoids many of the problems involving early processing and segmentation, yet it is important because it has furnished several important algorithmic and geometric insights. An important breakthrough in this domain was a move from "image understanding" in two dimensions to to an approach based on the three-dimensional world and laws governing three-dimensional solids.

9.2 SURFACE REPRESENTATIONS

The *enclosing surface*, or *boundary*, of a well-behaved three-dimensional object should unambiguously specify the object [Requicha 1980]. Since surfaces are what is seen, these representations are important for computer vision. Section 9.2.1 considers mainly planar polyhedral surface representations. More complex "sculptured surfaces" [Forrest 1972; Barnhill and Riesenfeld 1974; Barnhill 1977] are treated in Section 9.2.2. Some useful surfaces are defined as functions of three-dimensional directions from a central point of origin. Two of these are mentioned in Section 9.2.3.

9.2.1 Surfaces with Faces

Figure 9.1 shows the solid representation scheme most familiar to computer scientists. Solids are represented by their boundaries, or enclosing surfaces, which are represented in terms of such primitive entities as unbounded mathematical surfaces, curves, and points which together may be used to define "faces."

In general, a boundary is made up of a number of faces; faces are represented by mathematical surfaces and by information about their own boundaries (consisting of edges and possibly vertices). A closed surface such as the sphere or a spherical harmonic surface of Section 9.2.3 may be thought of as having only one face.

To specify a boundary representation, one must answer several important questions of representation design. What is a face, and how are faces represented? What is an edge, and how are edges represented? How much extra information (i.e., useful but redundant relationships and geometric data) should be kept?

What is a face? "Face" is an initially appealing but imprecise notion; it is at its clearest in the context of planar polyhedra. A face should probably always be a subset of the boundary of an object; presumably, it should have area but no dangling edges or isolated points, and the union of all the faces should make up the boundary or the object. Beyond this little can be said. For many purposes it makes sense to have faces overlap; it may be elegant to consider the letter on an alphabet block a special kind of face on the block that is a subset of the face making up the side of the block. On the other hand, it is easy to imagine applications in which faces should not overlap in area (then one easily can compute the surface area of a solid from its faces). In some objects, just what the faces are is purely a matter of