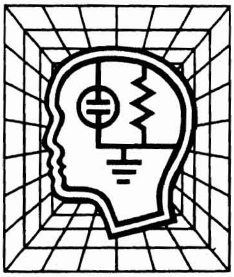
9

Computational Vision



Computational vision (CV), a subfield of artificial intelligence, is concerned with developing an understanding of the principles underlying visual competence in natural and artificial systems, and with providing a machine with some of the capabilities of the human visual system. Such capabilities include the ability to describe a scene based on data provided by imaging sensors, and to produce an understanding of the function, purpose, and intent of recognized objects. Table 9-1 summarizes the functional requirements of a general purpose vision system.

The challenge for computational vision is twofold: (1) The computing device should be capable of simulating physical experiments, such as *imagining* the movement or rearrangement and distortion of objects in the scene to solve a problem or compare the scene with reference scenes stored in memory, and (2) the computer should have some way of physically interacting with, and sensing, the outside world to build up a database of knowledge and experience. Without physical interaction, there is no reasonable way to capture and store in computer memory a suitably complete model that reflects all the complexity and detail of a real-world scene.

In other areas of AI, we have already observed the appropriateness of the saying, "If you are a hammer, everything

Table 9-1 ■ Functional Requirements for a General-Purpose Vision System

Geometric modeling. Determine the three-dimensional configuration of the surfaces and objects in a scene, including the location of the viewer (sensor) with respect to the scene being viewed.

Photometric modeling. Determine the location and nature of the illumination sources and the corresponding shadowing and reflectance effects induced in an image of the scene.

Scene segmentation. Partition the scene into meaningful or coherent subunits which can be independently analyzed and identified.

Naming and labeling. Identify the objects visible in a scene as either members of known object classes, or as known individuals. Determine the physical attributes (size, material composition, etc.) of recognized objects.

Relational description and reasoning. Determine the relationships among the objects in a scene, e.g., the appearance of the scene just prior to the time an image was acquired, and how the scene will appear immediately afterward. How can the objects in a scene be rearranged to achieve some given purpose?

Semantic interpretation. Determine the function, purpose, intent, etc., of objects in a scene.

looks like a nail." For computational vision, this can be paraphrased as, "If you are a digital computer, then everything looks like a number or a symbol." Thus, for a digital computer to deal with the visually perceived world, the signals acquired by the imaging sensors must first be converted into numbers and ultimately into symbols. We are therefore led to the *signals-to-symbols* paradigm described in the next section. The rest of this chapter discusses some of the techniques[7] involved in deriving symbolic descriptions from the sensed signals. The sections are sequenced to reflect the increasing complexity and abstraction of the corresponding techniques, beginning with the *low-level* representations and algorithms, and proceeding through the *intermediate* and *highest* levels.

A noteworthy difference between many computational vision representations and those of general AI is that in vision we often use arrays of picture elements (*pixels*) or other iconic (picturelike) representations that mirror the sensed image and thus retain a more direct correspondence to the real world. The accuracy and adequacy of any of the representations in the signals-to-symbols hierarchy is judged by how faithfully it portrays the real world scene that was originally sensed — i.e., the primary concern is with physical modeling of the world. This is in contrast to conventional AI systems which typically do not have a perceptual component, and thus work within a complete, consistent, and closed model of reality. The basic questions we address in this chapter are:

- What is the nature of the computer's symbolic description of the visual world, and how is it obtained?

[7] We attempted to select techniques that are both representative and can be understood without the need for an involved mathematical presentation.

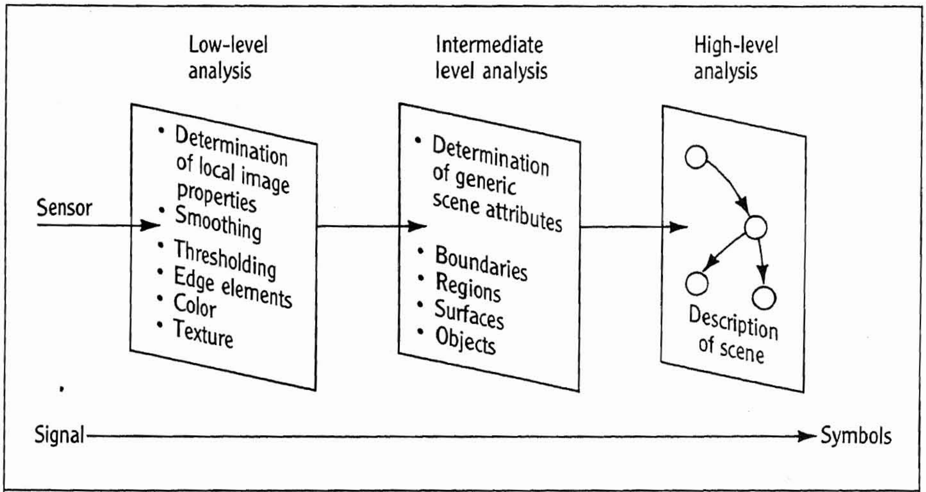


FIGURE 9-1 The Signals-to-Symbols Paradigm for Computational Vision.

Raw sensed data are transformed into a description of the scene by a series of inductive steps.

- What are the representations used in the signals-to-symbols paradigm?
- What algorithms exist for obtaining these representations and extracting information from them? How can we build a machine that can recognize objects and re-create scene geometry from the data provided by two-dimensional images?
- What are the few key ideas and major assumptions that underlie most of the current computational vision algorithms?

We conclude the chapter with a critical look at the signals-to-symbols paradigm, and indicate the requirements that must be satisfied by a computer if it is to achieve human-level competence in visual perception.

SIGNALS-TO-SYMBOLS PARADIGM

Computational vision (CV) is that body of theory and techniques that represents our present understanding of how competence for visual perception can be implemented in computing hardware. The dominant paradigm, signals-to-symbols, is one in which the raw sensed data is transformed into a meaningful and explicit description of the corresponding scene by a series of inductive steps employing progressively more abstract representations (Fig. 9-1). These steps can be partitioned into three categories, based on the nature of the modeling required to carry out the analysis: low-level scene analysis is based on local image properties, intermediate-level scene analysis uses generic geometric and photometric models, and high-level scene analysis is based on goal-oriented semantic models and relationships.

LOW-LEVEL SCENE ANALYSIS (LLSA)

At the lowest levels of the processing hierarchy, the representations and transformation techniques tend to be independent of any final purpose. Concern here is with physical and statistical modeling of the generic local properties of the visible surfaces in the scene and their appearance in the image. Low-level scene analysis (LLSA) is also concerned with the processes of transforming continuous sensor-derived signals into discrete digital representations, and the reduction of noise and distortion introduced by the sensing process. Inputs to this stage of processing are the raw signals from one or more sensors, and the output is typically a set of registered arrays, with each array corresponding to a particular scene attribute such as local surface orientation, surface reflectance, edge point location, etc. LLSA techniques have been developed to:

- Reduce imaging noise and unwanted scene detail without seriously degrading information needed for recovery of higher-level scene description.
- Disambiguate, i.e., separate the contributions of the illumination, surface reflectance, and surface orientation to the brightness of a point in the image (in a sense, to *invert* the imaging process).
- Detect local homogeneities and discontinuities that can be used to partition the image into regions corresponding to coherent objects in the scene.
- Detect distinguished local image features that are important markers and delimiters of scene features.
- Deduce local surface geometry (three-dimensional depth and orientation) from shading, texture, stereo analysis, and the analysis of a continuous sequence of images ("optic flow").

The analysis usually deals with local phenomena, using models based on such general concepts as continuity (or discontinuity) of intensity, texture, or color. While LLSA provides an interpretation of real-world physical phenomena for use by the integrative and reasoning mechanisms comprising intermediate- and high-level scene analysis, the LLSA techniques themselves have few (if any) of the attributes characteristic of reasoning. LLSA techniques are almost always based on the general position and the continuity assumptions: (1) that the image was taken from an essentially random location in space, and no deliberate attempts were made to make separated and discontinuous things look continuous in the image, or to align shadow and occlusion boundaries, or to make a curved line look straight, etc.; and (2) that because the scene is largely composed of continuous surfaces, the geometric, photometric, and physical properties measured at any point in the image are good predictors of the values appearing in the neighborhood of that point.

We first describe the image-acquisition process, and the preprocessing operations used to improve the quality and utility of the image. We then describe the various methods for detecting local discontinuity and homogeneity. Determination of scene geometry from single and from multiple images completes the discussion of low-level analysis.

Image Acquisition (Scanning and Quantizing)

The computational vision process begins either with an existing image (e.g., a picture taken at some previous time), or a scene currently being sensed by a television camera–type of sensor. To translate scene illuminance information into a form suitable for computer processing, it must first be converted into an array of numbers that represents the intensity of reflected light at each point in the scene.

If we start with an image, rather than the actual scene, we can obtain the intensity data by moving a small aperture or *window* over the image so that the average light intensity level of the image within the window is sensed by a photosensitive device. This scanning process can be carried out using a mechanical device that physically moves a sensor over the image in a regular and exhaustive manner, or by using a *flying spot scanner* that moves a beam of light sequentially over the image and senses the reflected (or transmitted) light from the image. (Such exhaustive scanning is in contrast to the selective scanning employed by the human visual system, as illustrated in Fig. 9-2.)

The continuous electrical signal that results from the mechanical scanning process must still be converted into an array of numbers by sampling the signal at regular time intervals (corresponding to regular spatial intervals over the image), and then approximating the measured voltage by the closest integer in some predefined range of numbers, as shown in Fig. 9-3. With current technology for storing, processing, and displaying pictorial data, a set of values for sampling and

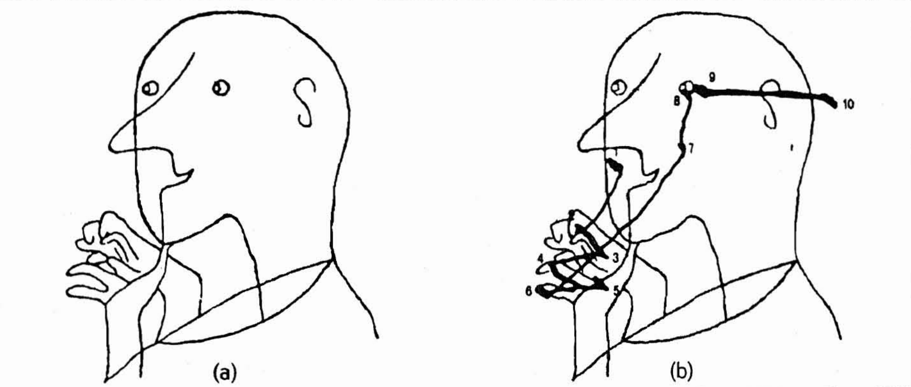
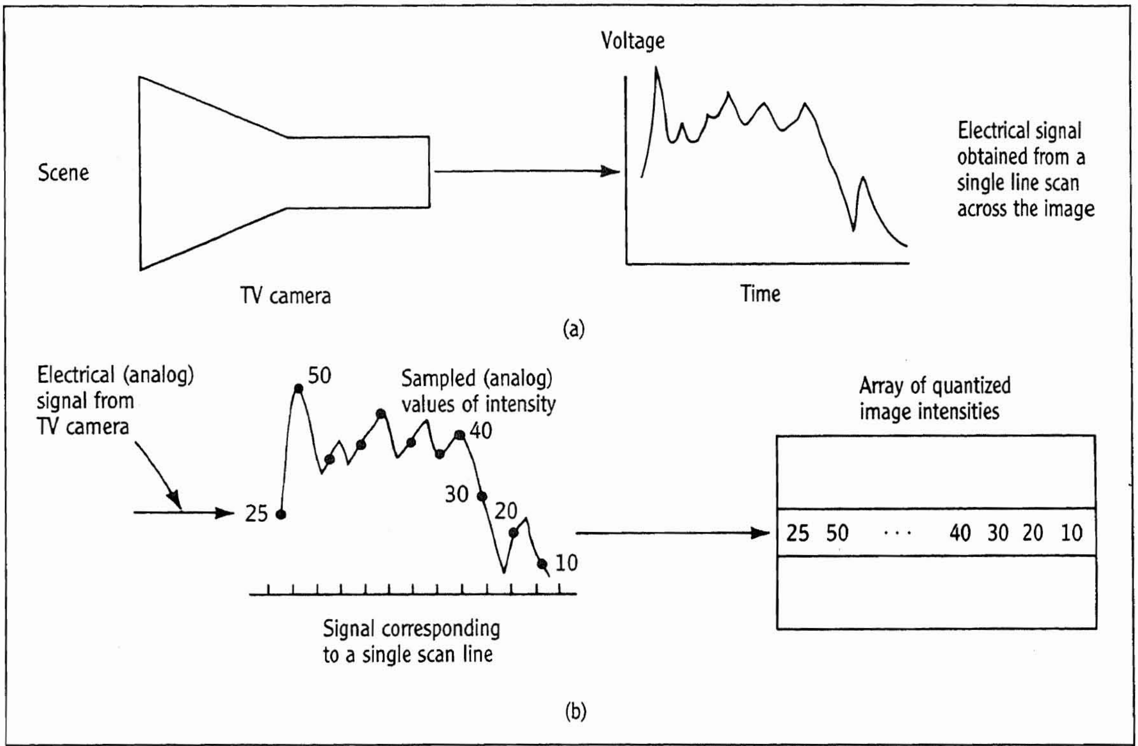


FIGURE 9-2 Adaptive Scanning of a Scene by the Human Eye.

(a) Original picture. (b) Picture with human eye scanning path shown. (From Norton and Stark, *Scientific American*, June 1971, with permission.)


Scene
TV camera
Voltage
Time
Electrical signal obtained from a single line scan across the image
(a)
Electrical (analog) signal from TV camera
25
50
Sampled (analog) values of intensity
40
30
20
10
Signal corresponding to a single scan line
Array of quantized image intensities
(b)

25 50 · · · 40 30 20 10

FIGURE 9-3
The Image Acquisition Process: Representing a Scene by an Electrical Signal and Then an Array of Numbers.

(a) Obtaining an electrical signal corresponding to intensities from a scene. (b) Converting an analog signal to sampled and quantized intensity values.

digitizing an image is typically a 256 × 256 grid with 256 possible intensity levels at each such grid point.

An important issue is the fidelity with which the actual scene appearance is captured by the array of integers that is extracted to describe it. In all of the digitizing approaches, the scene intensity is spatially sampled. The smallest image distance that can be tolerated between intensity samples depends on the characteristics of the lens system, on the sensitive surface of the sensor, and on the system used to convert the signals from spatially continuous to discrete quantities. The sampled intensities are further quantized into digital values to satisfy practical constraints on the memory or register word size of the computer. How much information is lost in this digitization process; i.e., how faithfully can we reproduce the original image from the derived finite array of numbers? Amazingly enough, the continuous image can be exactly re-created from its sampled (but unquantized) representation if the sample spacing is less than a value determined by the maximum spatial rate of change of intensities in the image. When the sampled analog signal values are converted to integer quantities, some amount of *quantization noise* is introduced in the analog-to-digital conversion which generally cannot be completely removed. However, a sufficient number of levels of quantization can be selected to insure that amplitude noise already introduced by the sensing process is not significantly increased. Thus, we conclude that the process of converting a continuous image into an array of integers does not cause any fundamental loss of information.

Image Preprocessing (Thresholding and Smoothing)

Image *preprocessing* uses operations that are relatively independent of scene content to alter the stored values in the digital array representing the scene. Preprocessing operations are generally intended to remove noise, enhance certain aspects of the image (e.g., edges), and induce other changes that will simplify the higher-level processing steps. It is assumed that image intensities are spatially continuous over most of the image, and that this continuity can be approximated, for example, by a low-order polynomial. An important goal in preprocessing is to avoid eliminating existing edges or introducing false edges.

Typical preprocessing operations are (1) *thresholding*, which reduces the digitally quantized image containing one of many possible integer intensity values at each image location to a binary picture containing one of only two possible values at each location, and (2) *smoothing*, the use of various filtering operations to enhance or suppress certain aspects of a scene.

1. *Thresholding.* The thresholding operation achieves image partitioning at an early stage in the analysis, reduces noise in the image, and simplifies later processing steps. The concept of thresholding is a simple one: we assume that pixels in a coherent region of the image all have an intensity greater than (or less than) a certain value. An intensity threshold is chosen, and all pixels whose intensity level is below this threshold are assigned one value ("black"), and all those above this threshold are assigned another value ("white"). Techniques for automatic threshold selection are discussed in Box 9-1.
2. *Smoothing (filtering).* Smoothing operations (1) remove noise and illumination artifacts that were introduced into the image during the sensing and image-acquisition process; (2) enhance edges and other selected image features; and (3) degrade unwanted detail below the level of resolution at which image interpretation is to be carried out (see Fig. 9-5). Smoothing is usually accomplished by replacing the intensity value of each pixel with a new value based on the intensity values of pixels in the immediate neighborhood of the given pixel. The problem that

BOX 9-1 Image Thresholding

Thresholding transforms a *gray-level* image, whose pixels can have any of a continuous range of intensity values, into a binary image in which each pixel is either black or white. Thresholding achieves a simple form of image partitioning, reduces noise in the image, enhances certain image features, and simplifies later processing steps. In some situations different thresholds may be used in different portions of the image to compensate for some known or deduced illumination variation or change in local scene contrast. A person interactively adjusting the threshold and viewing the effect of each such threshold setting can choose a threshold value that best achieves some desired effect. Automatic threshold selection is usually based on the following techniques:

- **Effect on image.** An iterative procedure for threshold selection can be based on the number, area, and stability of the regions generated by different thresholds—a good threshold setting should produce mostly large well-separated regions which retain their shape under small variations of the selected threshold value. An intensity contour map for an image can make apparent the effect of different thresholds on the final partitioning of the image (Fig. 9-4).

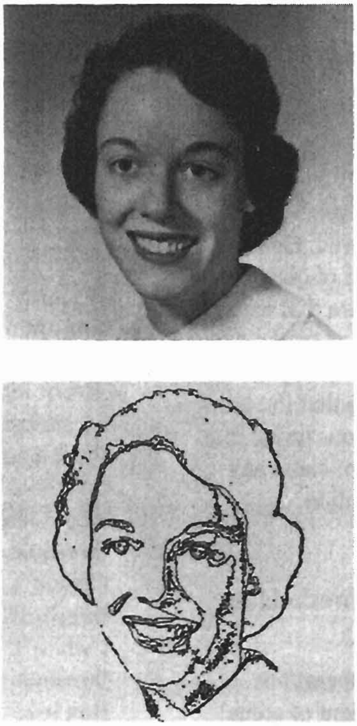


FIGURE 9-4
An Intensity Contour Map of an Image.

(Photos courtesy of SRI International, Menlo Park, Calif.)

- **Histogram analysis.** The intensity histogram is a graph whose x axis shows the range of possible intensity values, and whose y axis shows the number of pixels in the image that have each of these intensities. An image and its associated histogram are shown in Color Plate 3. Note that there are several peaks in the histogram, indicating the intensities that are most common in the image. An appropriate threshold setting for an image can automatically be determined by analyzing the histogram shape, often under the assumption that individual peaks of the histogram correspond to coherent (relatively constant intensity) regions of the image, and that the background is the lightest or darkest of these regions.

arises in the smoothing operation is how to avoid throwing the baby out with the bath water, i.e., how to avoid eliminating essential data in trying to accomplish the goals stated above. The basic issue is how to select the appropriate image subsets to process coherently without crossing boundaries separating different scene entities. For example, since the smoothing function usually consists of operations in a window centered around the pixel being modified, how can one keep from blurring edges when

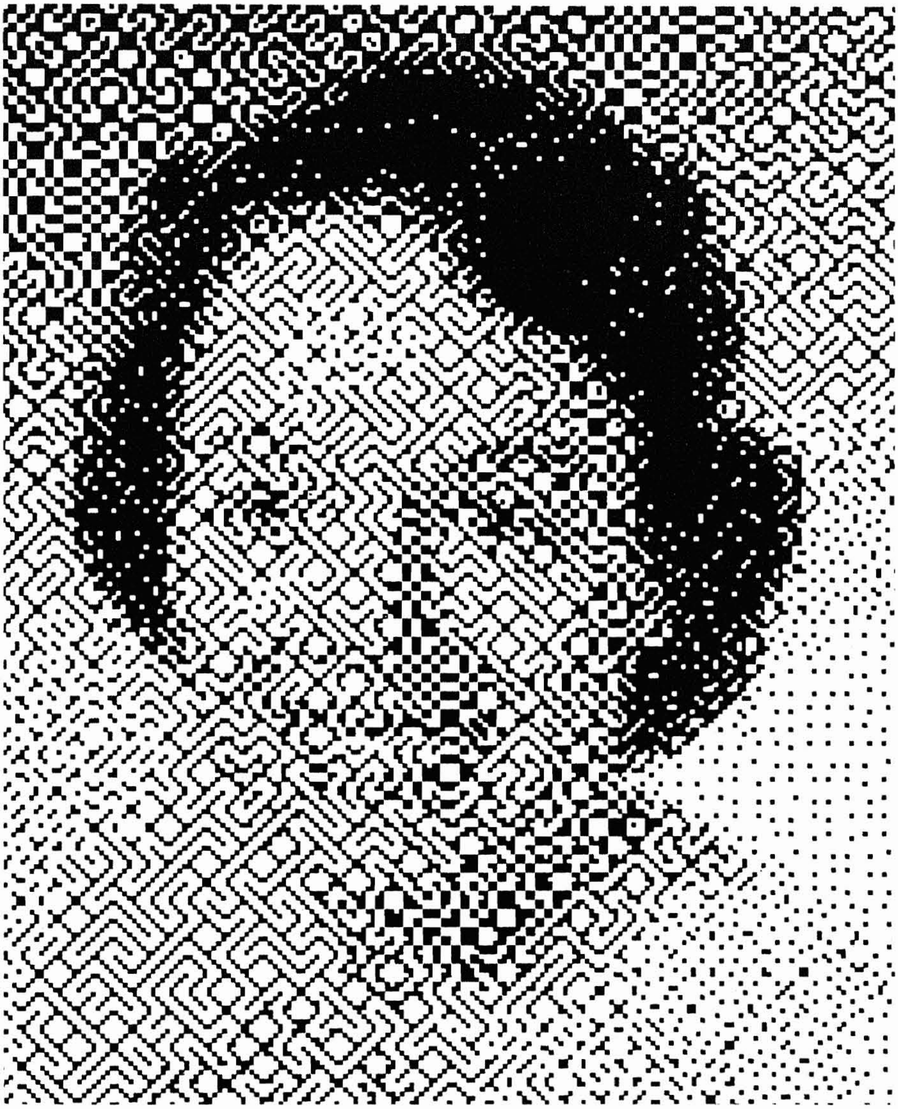


FIGURE 9-5
Need to Degrade Unwanted Detail.

Look at this picture from a distance. From far enough away, the texture elements disappear and this looks like a normal photograph. Now look at this picture through a narrow tube from the same distance at which the texture elements originally disappeared—the texture elements should become visible again, showing that under appropriate conditions, the human visual system deliberately degrades low-level detail. (Courtesy of SRI International, Menlo Park, Calif.)

pixels interior and exterior to the object fall in the same window? There is no single best smoothing algorithm. Some of the common approaches to smoothing and the implicit assumptions made for each are described in Box 9-2.

Detection of Local Discontinuities and Homogeneities (Edges, Texture, Color)

No reasonable semantic description or interpretation is possible if every point in a scene is unrelated to its neighbors. However, most of the scenes we encoun-

BOX 9-2 Image Smoothing

Image smoothing is employed to reduce noise, to enhance selected image features, and to degrade unwanted image detail. Most smoothing techniques fall into three broad categories: (1) local averaging, (2) model-based smoothing, and (3) geometric smoothing.

Smoothing via Local Averaging

Smoothing based on local averaging operations assumes that the *intensity surface* is continuous over most of the image. This assumption is a special case of the more general assumption that most of the image will depict continuous scene surfaces at an image resolution suitable for interpretation to be possible. In one smoothing approach based on this assumption, we replace the center of a small region around a pixel (typically a square *window*) by the weighted average of the values found within the window; this operation is identical for each pixel of the image. Another approach used to avoid the effect of deviant pixels and to retain edges is to use the median, rather than the average, of the intensity values in the window.

Smoothing Based On A Priori Models

The following technique is typical of a global approach to smoothing. Suppose we have a model of how the illumination varies in an image. For example, if we know (or assume) that the illumination can be modeled as a quadratic function, we can fit a quadratic surface to the intensity values of the pixels in an extended portion of the image. The intensity of each pixel is then subtracted from the corresponding value of the fitted surface, leaving only the higher-order variations of the underlying signal. We thus prevent a known artifact from interfering with our analysis of the intrinsic information residing in the image.

Geometric Smoothing

Geometric smoothing of an image can be carried out by assuming that very small isolated regions consist of noise and can be eliminated, and that small gaps between regions are imaging artifacts and can be filled in. Such smoothing can be readily accomplished in binary images using sequences of *shrink* and *grow* operations. In the shrink/grow approach to eliminating small noise regions, we first use a shrinking operation in which black pixels that are not completely surrounded by black are set to white. The shrinking operation can be iteratively applied several times. A growing operation can now be used in which all black pixels that are not completely surrounded by black pixels are provided with surrounding black pixels. Any small black noise regions will have been eliminated by the shrinking operations, and the larger regions will be left unaltered if the number of shrink and grow operations are equal. If the sequence of grow operations is applied first (followed by the shrink operations) small gaps in black objects or between adjacent objects will be filled, but the shapes of the objects will generally be unaltered.

ter can be decomposed into coherent objects or regions that are relatively homogeneous with respect to one or more of the following attributes: intensity, color, texture, distance, motion, material composition, physical cohesion, etc. Correspondingly, there are places in an image where there are sharp discontinuities in some of the above attributes; e.g., across an edge that occludes another part of the scene. Identifying homogeneous regions and discontinuities greatly simplifies the problem of analyzing the image in two important ways: (1) discontinuities are typically associated with the edges of objects, and locating the edges makes object shape explicit; and (2) identifying portions of an image corresponding to coherent objects allows us to analyze those portions in isolation if desired.

Some methods for finding such discontinuities and homogeneities are given below. Some of the techniques are independent of the objects in the scene, while others assume certain characteristics of specific real-world objects.

Local Edge Detection. It has long been recognized that the detection of the edges of the objects appearing in an image is an essential step in scene analysis, and for this reason there has been considerable effort devoted to developing effective edge detection algorithms. One class of such algorithms, local edge detectors (LEDs), assigns an edge value to individual pixels, but does not link pixels together to form an extended edge segment. Therefore, an additional association or linking step must still be carried out in the computer to obtain an internal representation of the connected edge segment. Edge linking is an intermediate level operation, and is discussed later. LED algorithms can be grouped into the following categories:

Local gradient operators: These algorithms are based on characterizing an edge as a local intensity discontinuity. The intensities in a local region of the image are examined and an edge value (and sometimes an edge orientation) is assigned to a picture element based on the change of intensity within that local region.

The simplest and most commonly employed gradient type LEDs have the following characteristics (e.g., the Sobel edge detector; see Fig. 9-6): They *convolve* a set of small digital *operator* arrays (e.g., a 3 × 3 pixel square) with the image array.[8] Each such operator array evaluates the intensity gradient in one particular direction at the image location corresponding to the center of the operator array (Fig. 9-6b). An approximate gradient can be computed by applying the operator in two orthogonal directions and employing vector addition (Fig. 9-6c). Better results are obtained by running the operator at a large number of angular orientations and selecting the maximum value obtained as the gradient magnitude at the given location, and the corresponding direction as the gradient direction. Edge pixels are determined by thresholding the gradient image.

The simple gradient-type LEDs, such as described above, ignore a number of considerations relevant to real imagery. First, most real images have extended

[8] In convolution the operator array is moved across the image, and at each placement the elements of the operator array are multiplied by the corresponding image elements. All the products are summed, and the result is assigned to the image location beneath the current center of the operator array.

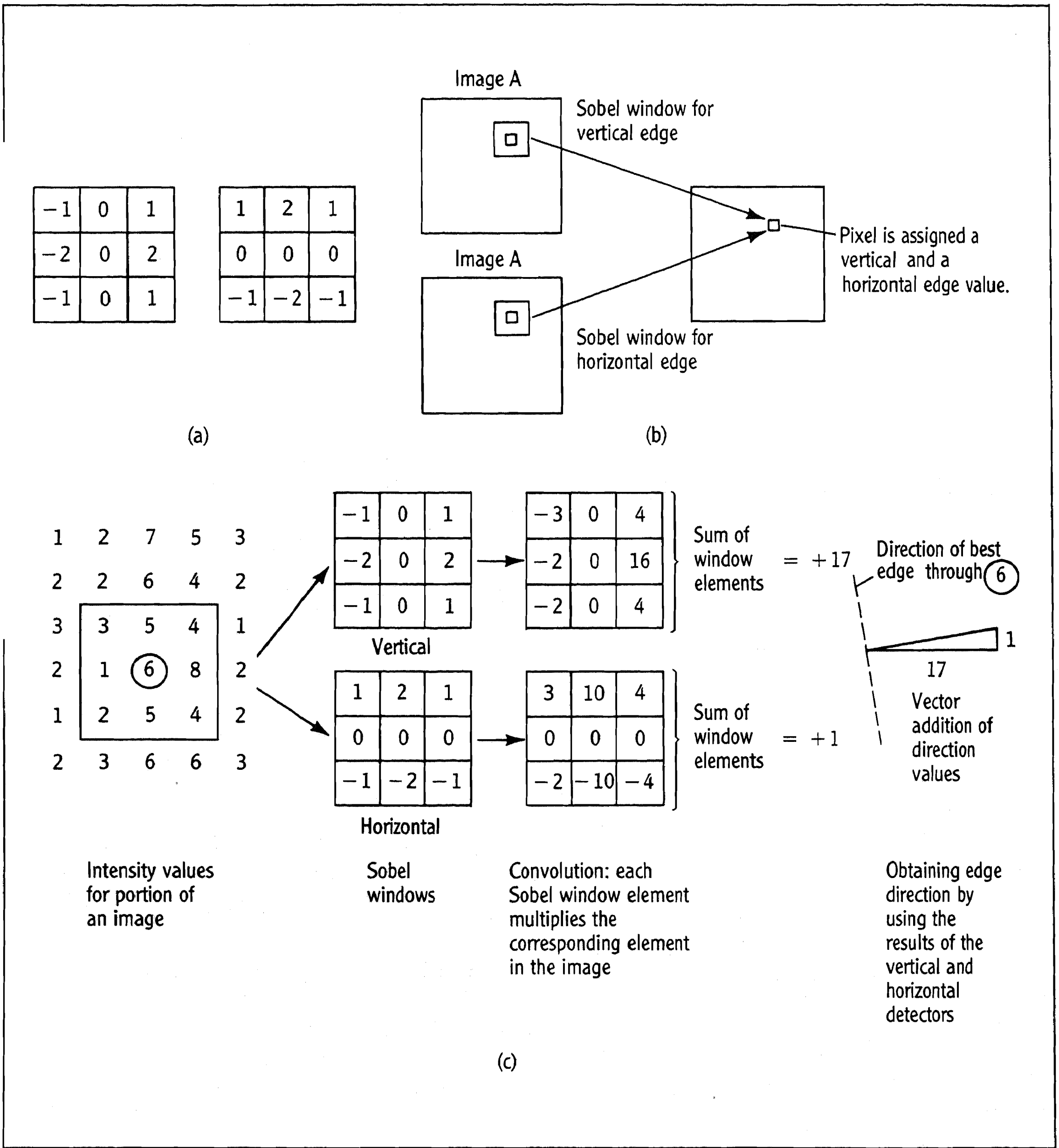


Image A
Sobel window for vertical edge
Image A
Sobel window for horizontal edge
Pixel is assigned a vertical and a horizontal edge value.
(a)
(b)
Vertical
Horizontal
Sum of window elements = +17
Sum of window elements = +1
Direction of best edge through 6
1
17
Vector addition of direction values
(c)

−1	0	1
−2	0	2
−1	0	1

1	2	1
0	0	0
−1	−2	−1

1	2	7	5	3
2	2	6	4	2
3	3	5	4	1
2	1	6	8	2
1	2	5	4	2
2	3	6	6	3

Intensity values for portion of an image

−1	0	1
−2	0	2
−1	0	1

1	2	1
0	0	0
−1	−2	−1

Sobel windows

−3	0	4
−2	0	16
−2	0	4

3	10	4
0	0	0
−2	−10	−4

Convolution: each Sobel window element multiplies the corresponding element in the image

Obtaining edge direction by using the results of the vertical and horizontal detectors

FIGURE 9-6 The Sobel Edge Detector.

(a) Sobel windows. (b) Applying Sobel windows to an image. (c) Example of Sobel edge detector used to find direction of edge through circled element in image array. (See Fig. 9-10 for example of application of Sobel detector to a real photograph.)

smooth gradients that are artifacts of the illumination and imaging processes. There is no reasonable way to set an absolute threshold on the local gradient (e.g., as required by the Sobel LED) to distinguish intensity discontinuities (edges) from these extended smooth gradient artifacts. We are thus faced with the problem of how to detect low contrast edges (using a fixed threshold gradient operator) without being deluged by false alarms arising from smooth gradients. There is also the problem that the intensity discontinuities corresponding to different edges in an image can vary over a range of widths (resolutions); a single size convolution mask (even at multiple orientations) is not adequate. Box 9-3 presents an approach to local edge detection which is better able to deal with these problems.

Generic model fitting: These algorithms are based on modeling an edge as a specific extended intensity profile. Within some local search area, a single best fit is selected to this specified intensity profile . The generic model-fitting approach is very specific about the type of discontinuity it is searching for, in contrast to the local gradient approach which is satisfied by a wide variety of intensity discontinuity types.

It is often acceptable to describe an edge as being a geometrically straight intensity step discontinuity over some local extent of the image. In such a case, we can move a small window over the image and find the best fit of the above model to the intensity pattern viewed through the window at each of its stopping locations. The *Hueckel edge detector* is the most commonly employed operator of this type. It accepts the digitized light intensities within a small disc-shaped subarea (containing at least 32 pixels) and yields a description of the most edgelike (brightness discontinuity) occurrence found within the disc.

Semantic model fitting: The algorithms based on "semantic edge models" use specific characteristics of the objects of interest to detect the edges. For example, various objects such as roads or rivers in an aerial photograph, or ribs in a medical x-ray film, have edge properties that are dependent on the nature of the objects themselves. The algorithms in this category are therefore tailored to search for the edges of a particular class of objects. For example, to detect roads in low resolution aerial images, the Duda road operator (Fig. 9-7) specifically requires

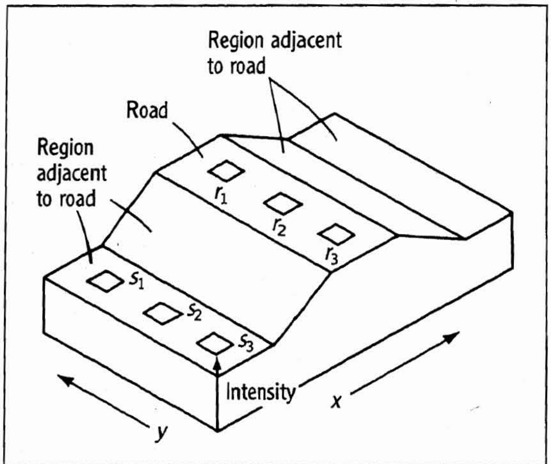


FIGURE 9-7 The Duda Road Operator.

The general idea is (1) adjacent pixels along a road should have similar intensities, and (2) adjacent on-road and off-road pixels should have different intensities. If these two conditions are satisfied, a high road score results.

Score = $f(r)/g(r,s)$. $f(r)$ is high if the differences between r_1,r_2,r_3 are small; $g(r,s)$ is low if (r_1-s_1), (r_2-s_2), (r_3-s_3) differences are large. For a road, $f(r)$ is high and $g(r,s)$ is low, resulting in a high score.

BOX 9-3 Local Edge Detection Based on Lateral Inhibition

Organic visual systems universally employ a mechanism, called *lateral inhibition,* that offers some relief from the edge-finding problems encountered by using the simple gradient-type LED. Computationally, lateral inhibition involves setting the edge "signature" of a picture element (pixel) to be the weighted difference of the average intensities of two differently sized masks centered on the pixel. In some simple implementations, each mask is a uniformly weighted rectangular box. In more sophisticated versions, and especially as found in biological systems, the masks have a gaussian, bell-shaped weighting, rather than a uniform weight distribution (see Fig. 9-8). Applying such an *operator* to a region of an image in which there is a uniform gradient, no matter how strong, will result in a zero response everywhere (assuming that the sum of the weights in the two masks are equal). If there is a sharp intensity discontinuity superimposed on the uniform gradient, then as the operator is moved along a path normal to the discontinuity, its value will be zero until the larger (outer) mask of the operator crosses the discontinuity. Then the value returned by the operator will increase until the smaller (center) mask intersects the discontinuity. At this point the value returned by the operator will rapidly decrease, becoming zero when the smaller mask is exactly centered on the discontinuity. Continuing, we now obtain a symmetrical response to what we had approaching the discontinuity, but with reversed sign. The edge is detected by locating the "zero-crossing" of the operator's response over some portion of the image (not just a zero value). The strength and direction of the edge must be determined by analyzing the response of the operator in the vicinity of the *zero-crossing.*

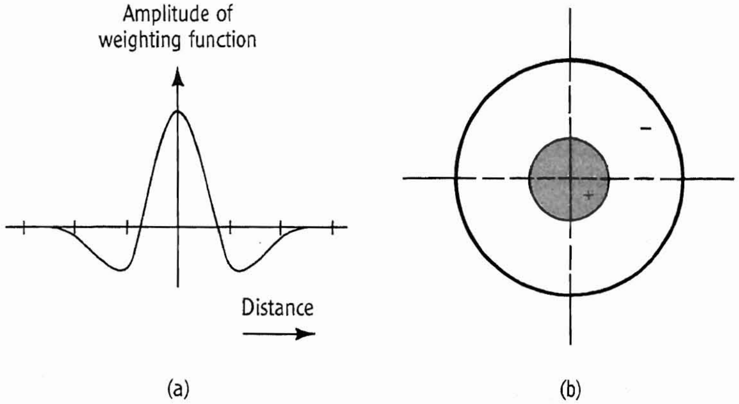


FIGURE 9-8
Lateral Inhibition using "Zero-Crossing" Detector based on Difference of Gaussians.

that a road fragment have relatively constant width and intensity values along a line segment in addition to the generic requirement of high intensity gradient normal to the direction of the segment.

The results obtained by applying several LED algorithms to the same scene are presented in Fig. 9-10. The Duda operator, specifically designed to detect roads, produces a more intuitively obvious result than the other more generic edge operators.

Analysis of Local Homogeneity. The identification of homogeneous regions is generally accomplished by first labeling

BOX 9-3 *(continued)*

To determine very sharp intensity discontinuities in an essentially noise-free image, the central mask can be one pixel in diameter, and the outer mask just slightly larger; however, if the intensity discontinuity is "blurred" over a number of pixels, and if the image is noisy, then the outer mask must have a diameter larger than the width of the edge transition region. In fact, the best response in terms of the magnitude and slope of the reversal on which the zero-crossing occurs will be obtained if the outer mask is made as big as possible without making it so large that it simultaneously covers more than one edge. Increasing the diameter of the central mask smoothes the response of the operator, but also decreases the amplitude and slope of the section on which we are looking for the zero-crossing. Choosing an optimal size for the central mask is a complex issue, but it probably should not be larger than the diameter of the edge transition. To deal with a complex scene, a set of zero-crossing operators with graded sizes of the central mask is required to detect edges of varying widths (Fig. 9-9).

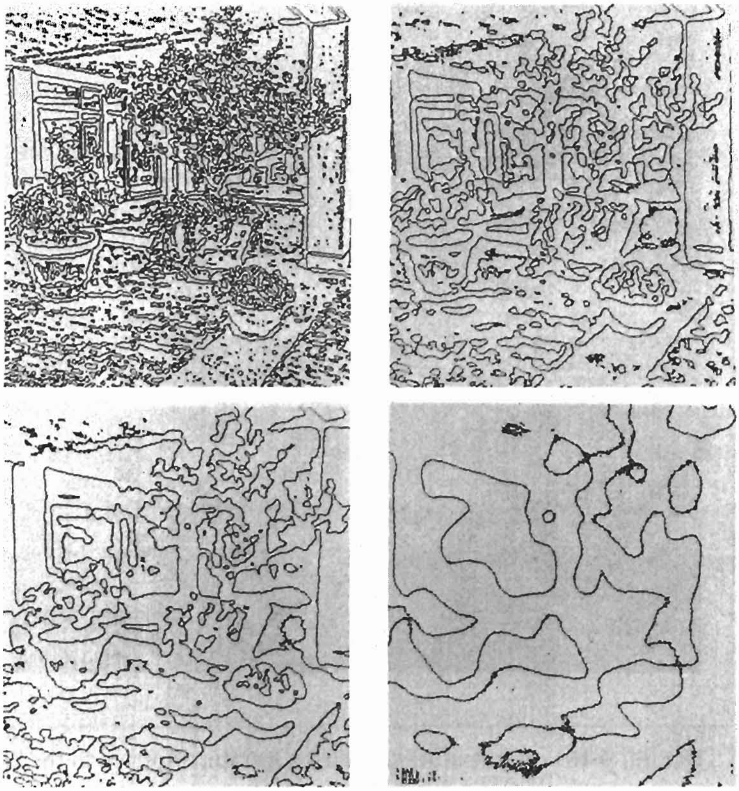


FIGURE 9-9 Use of Zero-Crossing Operators to Find Edges at Different Scales of Resolution.

The diameter of the zero-crossing operator was varied to obtain these results. (Courtesy of SRI International, Menlo Park, Calif.)

each pixel in the image with the values of attributes such as texture and color.

Texture analysis. Although there are many fascinating biological and computational aspects of texture perception/detection, we are concerned here only with the limited question of how analysis of texture can be used to partition an image into homogeneous regions. Since the human visual system can easily recognize different types of textures, it may come as a surprise to find that there is no generally accepted definition of texture and thus no agreement as to how it can be measured. It is not easy to formally characterize the basis of our perception

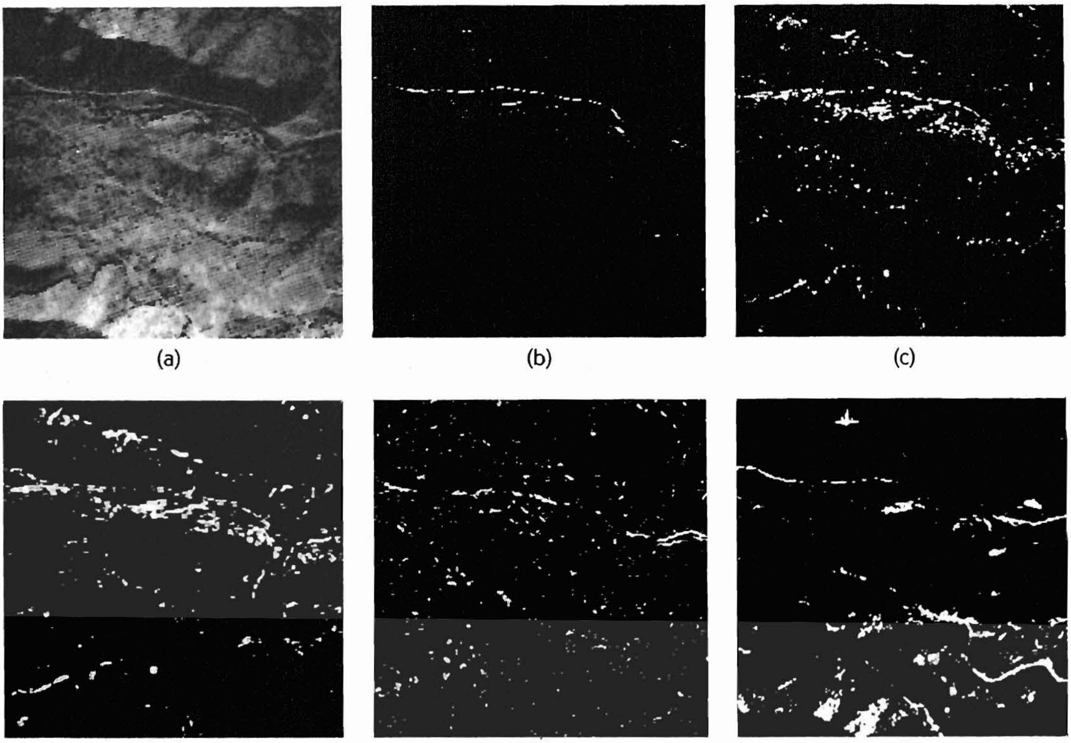


FIGURE 9-10 Result of Various Edge Operators Applied to the Same Scene.

(Operator scores are thresholded to highlight the locations assigned the best scores.) (a) Original image; (b) Duda road operator; (c) Roberts' cross gradient; (d) Sobel-type gradient; (e) Hueckel line operator; (f) Intensity. (Courtesy of SRI International, Menlo Park, Calif.)

of texture described by terms such as "fine," "coarse," "smooth," "granular," "random," "mottled," etc. We know intuitively that texture involves a statistical or structural relationship between the basic elements, and for figurative (cellular, macrostructure) textures such as a brick wall or a tiled floor, our visual system can detect the underlying patterns that make up the texture design, and we can describe the relationship of the elements (Fig. 9-11a). For microstructure texture, such as the fields seen in aerial photographs, or the texture of cloth, the underlying patterns are no longer obvious and it is difficult for the human to describe

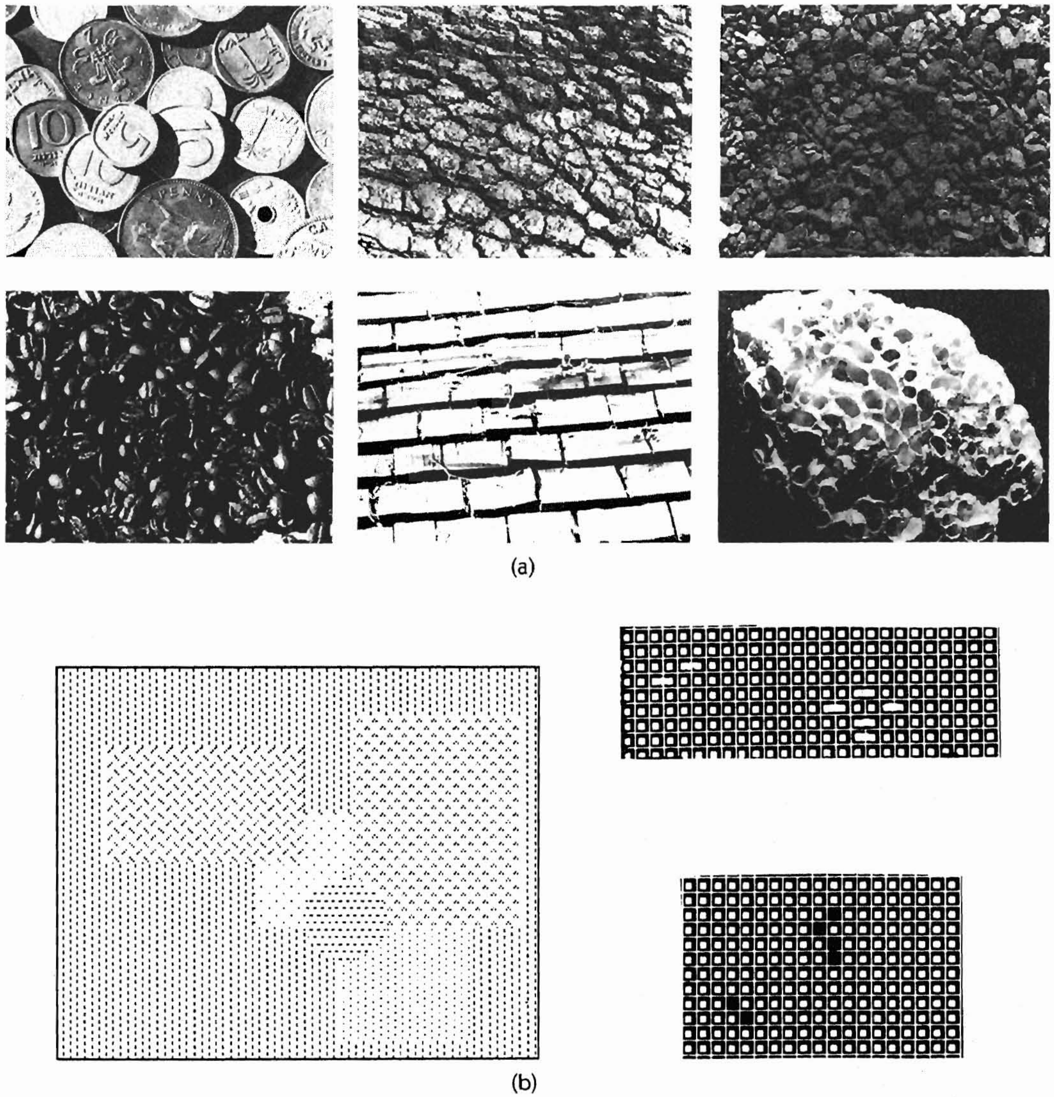


FIGURE 9-11 Texture Discrimination.

(a) Examples of macro- and microtextures. (Photos by O. Firschein.) (b) The human visual system has no trouble partitioning images containing various types of texture patterns.

(but not to recognize!) the texture (Fig. 9-11b). Analysis of micro- and macrostructured textures is discussed in Box 9-4.

Color. A colored image is typically represented in the computer as three separate arrays of numbers, with each array corresponding to a primary color, and the value of a pixel representing the intensity of the primary color component at the corresponding location (see Fig. 9-12). Each element of the image is therefore represented by a triple of values. Although all three images are usually obtained simultaneously using a *color vidicon,* the arrays can also be obtained by sensing the scene three times, each time through a different color filter. Sometimes the spectral (color) component of an image is relatively constant over large areas of the surface of a single object, while total intensity may change more erratically due to uneven direct illumination and local reflections. Thus color can be used to help find homogeneous regions in an image by grouping together neighboring pixels with similar color attributes. Errors may still occur due to the fact that objects will be colored by reflections from other colored objects, because of shadows, and because the basic assumption of homogeneity is not always valid.

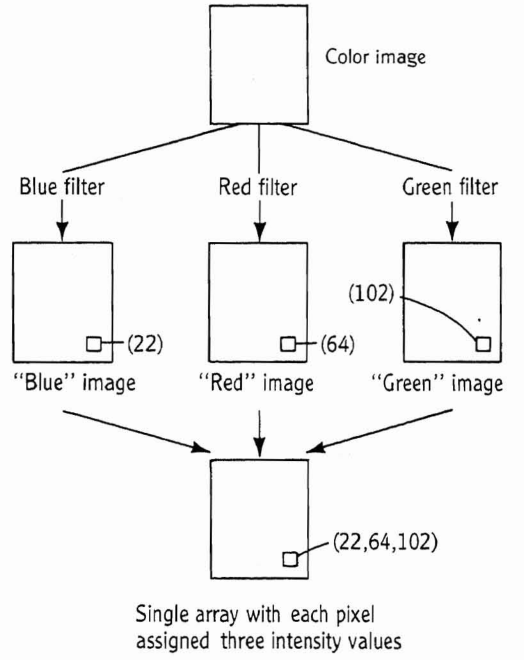


FIGURE 9-12
Color Represented as a Triple of Values.

(Color Plate 3 shows a color photograph and the "blue" image extracted from it.)

Local Scene Geometry from a Single Image (Shape from Shading and Texture)

We know that the shading and texture present in a single image can produce a vivid impression of three-dimensional structure (Fig. 9-13). What is the computational basis of this effect? A crucial source of information about three-dimensional structure is provided by the spatial distribution of surface markings in an image. Since projection distorts texture geometry in a manner that depends systematically on surface shape and orientation (see Fig. 9-14), isolating and measuring this projective distortion in the image allows recovery of the three-dimensional structure of the textured surface. This is not a straightforward task because the projective distortions encoding surface orientations are confounded in

BOX 9-4 Analysis of Micro- and Macrostructured Texture

Texture analysis is a basic operation in scene partitioning. It attempts to formalize our intuitive notion of surface appearance.

Microstructured Texture

Two general approaches can be taken to the analysis of microstructure textures, (1) analysis on the basis of microstructure regularity as detected by statistical or power spectrum techniques, and (2) the use of techniques that model the original surface that produced the texture patterns.

Statistical features approach. Because the basic units of microstructure are small, techniques that detect regularity in short sequences of pixels can be used to partition microtextured scenes. The basic strategy is to form a feature space based on measurements in a neighborhood about each pixel in the image. Segmentation of the image can be accomplished by assigning pixels to one or another region on the basis of the location of that pixel in the feature space.

- **Co-occurrence approach.** The spatial gray-level relationships can be expressed as S(i,j|d,A), the number of times a pixel of intensity i appears within d pixels and an angle A of a pixel of intensity j, in some neighborhood of the pixel to be classified. One can use functions of S as the components of a feature space. Typical of such functions is *energy*, formed by summing the square of S over all i and j, for given values of angle A and distance d.
- **Fourier analysis approach.** Regularity in gray level pattern shows up in the Fourier transform taken in various directions around a pixel. The set of Fourier measures, obtained by convolving a set of weighted windows over the image, then forms the components of a feature space. Each pixel in the image has an associated set of Fourier energy measurements, and can be represented in the feature space.

Modeling approach. In a *process-modeling* approach to texture analysis, one attempts to describe things in the world in terms of how they arose, e.g., man-made, growing, or *wearing-down* processes (as in a canyon). Using this point of view, it is possible to predict how natural surfaces will produce the texture patterns in an image. A technique based on *fractal functions* can model image textures arising from physical processes that alter the terrain via a sequence of small changes; the corresponding image turns out to have measurable statistical properties that are invariant over linear transformations of intensity and transformations of scale. The fractal dimension, D, of a surface corresponds roughly to our intuitive notion of jaggedness. Thus if we were to generate a series of scenes with increasing fractal dimension D, we would obtain what could be described as (1) a flat plane for D=2, rolling countryside for D=2.1, a worn, old mountain range for D=2.3, a young, rugged mountain for D=2.5, and finally, a stalagmite-covered plane at D=2.8 (see Color Plate 2, a synthetic scene generated using fractal textures.) It is possible to measure the fractal dimension of imaged data, and discover whether the corresponding three-dimensional surface is rough or smooth. This information can be used to partition the image into regions of homogeneous surface character, as follows. The fractal dimension is computed for each (say) 8×8 block of pixels in an image, and a histogram of the fractal dimensions is computed for the entire image. This histogram is then broken at the valleys between the modes of the histogram, and the image is segmented into regions belonging to one mode or another.

Macrostructured Textures

Techniques for dealing with macrostructure textures have met with only limited success. Macrostructure texture analysis is quite difficult because one must identify both the primitive(s) and the spatial relationship between them. Perspective effects add to the difficulty by changing the size and shape of the primitives depending on their position in the two-dimensional image.

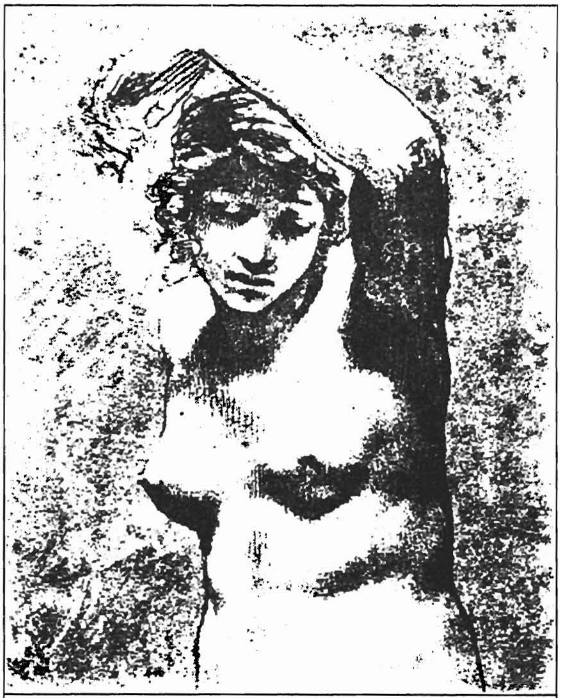


FIGURE 9-13
Three-Dimensional Structure from Shading and Texture.

(*Study of a Female Nude* by Pierre-Paul Prod'hon. Collection of Henry P. McIlhenny.)

the image with the properties of the original texture on which the distortion acted.

If the texture is simple and regular, such as a square tile pattern, the change in shape of the rectangles in the image can be measured to derive the surface shape. However, in most situations, there is not a simple, regular texture pattern. An effective technique for recovering surface orientations from general images must rest on texture descriptors that can actually be computed from such images. A natural assumption is that textures do not "conspire" to mimic projective effects or to cancel these effects. Thus it is reasonable to assume that what looks like projective distortion really is such distortion.

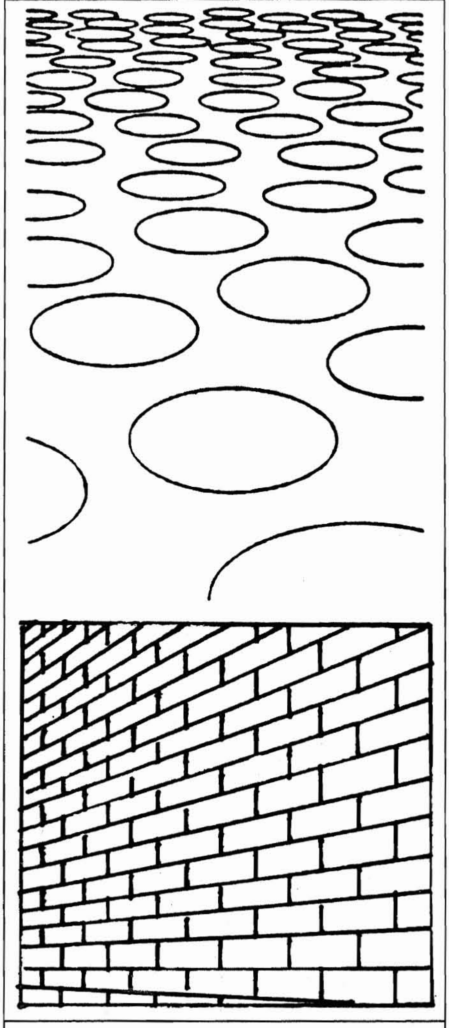


FIGURE 9-14
Effects of Projective Imaging on Regular Texture Patterns.

As much as possible of the observed variation is therefore attributed to projection—the surface orientation that best explains the data in an image is the best guess for the actual orientation of the surface in the scene.

The appearance of surface markings in the image is subject to two simple geometric distortions: (1) As a surface recedes from the viewer, its markings appear smaller (the *railroad track effect*); and (2) as a surface is inclined off the frontal plane, its markings appear foreshortened or compressed in the direction of inclination (a tilted circle projects as an ellipse). Thus, any method for recovering surface orientation from texture must be expressed in terms of some concrete description of the image texture that is sensitive to these two types of distortion.

When a plane texture is viewed at an unknown orientation, the original texture and the orientation of the plane with respect to the observer cannot be unambiguously recovered from the image. However, it is possible to produce a set of candidate reconstructions by applying an *inverse* projective transform at all values of tilt and slant angle, each associated with a particular orientation of the planar surface. The problem of recovering surface orientation can therefore be recast into that of choosing a "best" or most likely member from a set of possible reconstructions, by ordering the candidate reconstructions by some criteria of likelihood. For example, the ordering can be based on the assumption that all edge directions are equally likely in the scene. First, the edge pixels and their orientation in the image are found, and then one finds the best combination of tilt and slant angles of a plane on which these edges project so as to satisfy the "randomness of edge direction" assumption. The planar technique is extended to curved surfaces by finding a planar estimator to a circular region surrounding each image point. Repeated over the image, this method provides estimates of local surface orientation, but its validity depends on a randomness assumption which is frequently violated.

The human can perceive convoluted three-dimensional surfaces on the basis of the projective distortions imposed on complex and subtle textures; computational vision is a long way from duplicating this ability.

Local Scene Geometry from Multiple Images (Stereo and Optic Flow)

All of the previously described LLSA techniques operate on a single image. Below we discuss two techniques, stereo and optic flow, that recover scene information based on analyzing a sequence of images.

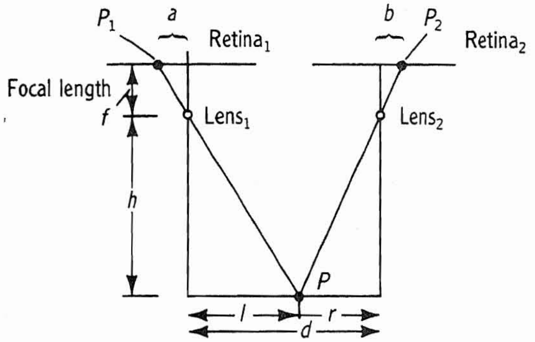
- *Stereo*. Stereoscopic vision allows a three-dimensional model of a scene to be derived from two sensors that observe the scene from slightly different viewpoints. The relative difference in the position of objects in the two images is called *disparity*, and is caused by the slight difference in angle from any given object to each sensor. In some conjectured, but still unknown manner, our brain measures this disparity and estimates the absolute distances between objects and the viewer (see Appendix 8-2). From experiments employing synthetic

BOX 9-5 Stereopsis

Our two eyes form slightly different images of the world because their spatial separation causes them to be at different spatial orientations with respect to objects in the scene. The relative difference in the position of an object on the two retinas is called "disparity," and can be used to estimate the distance of an object from the viewer. "Stereopsis," "binocular vision," and "stereo vision" are the terms used to describe the ability of a vision system to carry out this analysis.

Figure 9-15 shows a simplified two-dimensional example of a stereo system. Two lenses, separated by a distance, *d*, project a point, P, to the respective retinas at P_1 and P_2. P is distance *h* from the line of the lenses. The distance from the lens to the retina is *f*, the "focal length."

The disparity, the shift of the point's position in one image relative to the point's position in the other image, is $(a+b)$. The distance h is given by $h = (fd)/(a+b)$, where the focal length f and the distance between lenses d is a constant for a particular lens pair. If fd is unknown but a constant, then if we can find the disparity between points in the images, the relative distance of objects from the image plane can be determined. Measuring disparity requires that we first identify corresponding points in the two images; people are able to solve this correspondence problem even when the points to be matched are randomly scattered in the images and there is no edge or other obvious structure. The major problem in computer stereo analysis is the solution of the correspondence problem.



$a/l = f/h$
$l = ah/f$

$b/r = f/h$
$r = bh/f$

$$d = l + r = h(a + b)/f$$
$$h = fd/(a+b)$$

FIGURE 9-15 Stereo Geometry.

images composed of random dots, [Julesz 71], we know that the human does not depend on detection of recognizable features in each individual image to fuse a stereo pair of images, i.e., to recover the depth information. In computational vision, a stereo pair of images is obtained by using two separated cameras or by moving a single camera to two locations (*motion stereo*). The critical and difficult step is determining correspondences between points in the two images so that the disparity can be determined. Once this has been accomplished, straightforward geometric analysis can be used to compute the three-dimensional location of points in the scene (see Box 9-5). Figure 9-16 shows a stereo pair of images, and an artificially constructed view of the corresponding scene obtained by deducing the scene geometry using stereo analysis. The only scene information employed in this reconstruction was obtained from the stereo image pair shown in Fig. 9-16(a).

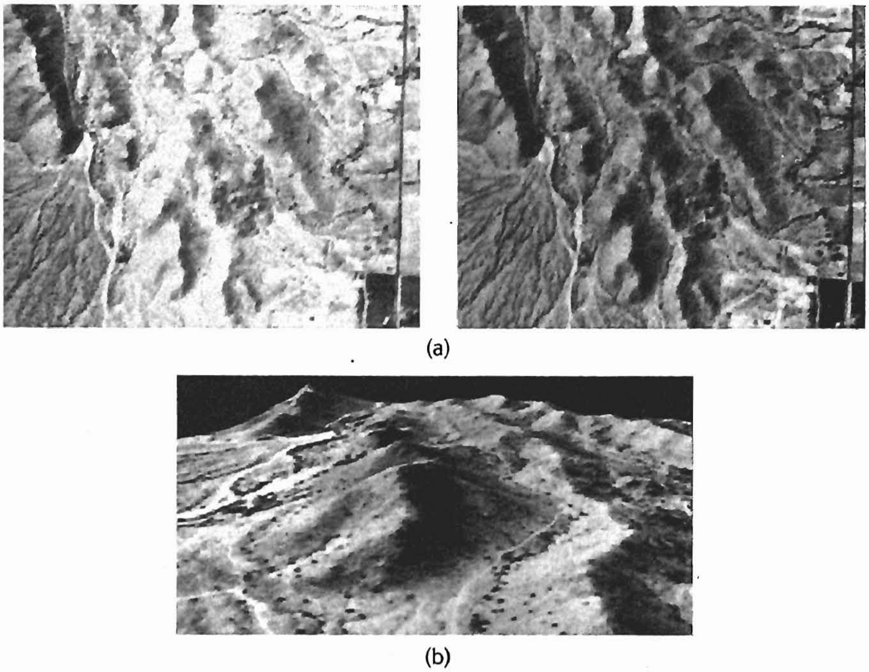


FIGURE 9-16 Three-Dimensional Scene Modeling using Stereo Analysis.

(a) Stereo pair of images. (b) An artificially constructed view of the scene depicted in (a), obtained by stereo analysis. (Courtesy of SRI International, Menlo Park, Calif.)

- *Optic flow*. Suppose we have an imaging sensor moving through a scene. As the sensor moves forward, scene points will move in the image plane along curves known as *optic flow curves*. By analyzing these flow curves it is possible to determine the distance from points in the scene to the sensor. These distances can then be used to construct a three-dimensional model of the scene just as in the case of employing stereo analysis. If the sensor is moving forward with pure translational motion, the optic flow curves will be straight lines that converge at a point known as the *focus of expansion* (FOE) (Fig. 9-17). (If the sensor moves backward, the lines would converge at a *focus of contraction*.) If the FOE can be located in the image plane, and the distance moved by the sensor from frame to frame is known, then the distance from a point in the scene to the sensor can be found. If it is known only that the motion of the sensor was constant, but not how far the sensor moved, then the relative depth of points in the scene can be obtained (see Box 9-6). This is often sufficient,

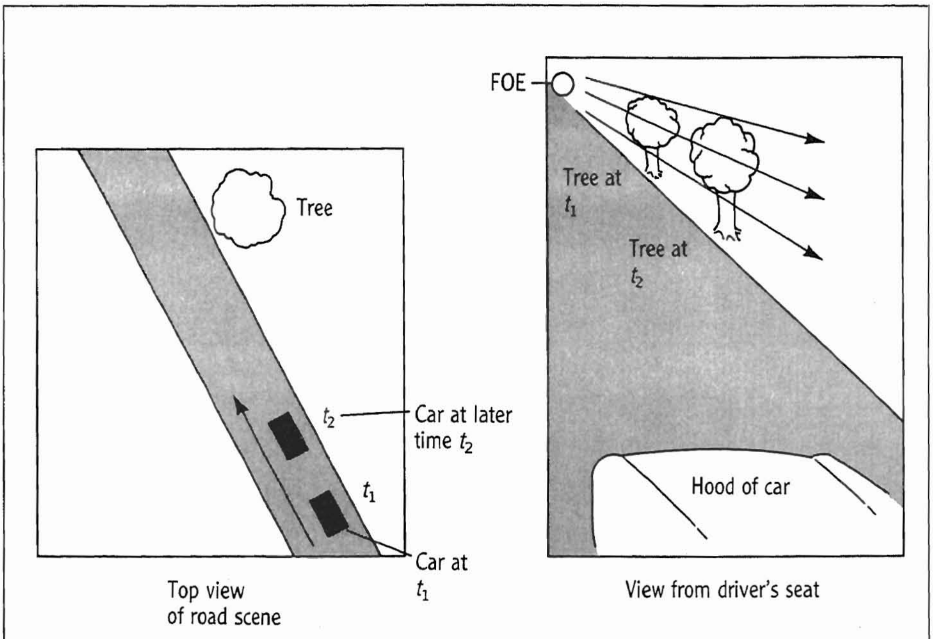


FIGURE 9-17 An Example of Optic Flow.

Seen from the driver's eyes, the hood of the speeding car is perceived as stationary, but the features of the tree-lined road appear to be moving along straight lines radiating from the focus of expansion (FOE).

for example, to partition the scene. If the motion of the sensor is more general, then the optic flow curves must be decomposed into rotational and translational components. Only the translational component contributes to the depth computation. One approach used is to analyze small regions of the image, using the disparities to obtain a trial FOE for each region. For portions of the image in which the rotational effects are strong, the flow segments will not converge to a FOE. If enough portions of the scene do provide a consistent FOE, then the translational portion can be separated from the rotational portion of the disparities. Optic flow is an interesting alternative to stereo because it offers a method for three-dimensional scene modeling that is not dependent on a solution to the matching problem.

INTERMEDIATE-LEVEL SCENE ANALYSIS (ILSA)

Intermediate-level scene analysis (ILSA) is concerned with integrating local or point

BOX 9-6 The Computation of Depth from Optic Flow

Assume a camera with center of perspective at point F is moving with constant velocity v along its principal axis, z, relative to some point P in three-dimensional space. Assume a focal length of one unit with the image plane parallel to the y axis, as shown in Fig. 9-18.

Let P^1 be the image of P. Then

$$y^1 = y/z \text{ and } v = dz/dt,$$

and w, the image plane velocity of P^1, is:

$$w = dy^1/dt = -(y/z^2)v = -y^1v/z$$
$$\text{and } z = -y^1v/w$$

Thus $d^2 = y^2 + z^2 = z^2[(y^1)^2 + 1]$, and d is approximately equal to $(y^1)^2v/w$. This means that if the camera velocity is known, and the image velocity of a point can be measured, then the distance from the camera to the point in space can be found.

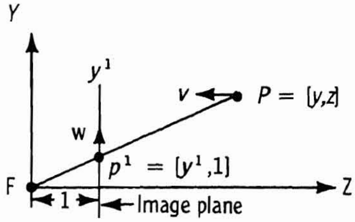


FIGURE 9-18
Geometry of Optic Flow.

features into global constructs, e.g., forming edge points into continuous contours, partitioning the image into coherent regions, assigning semantic labels to detected scene entities, evaluating sensor parameters (e.g., determining the spatial location of the sensor), deriving a model of the illumination sources, etc. Distinct representations and techniques are required for intermediate-level scene analysis because it is not possible to extend techniques employed in LLSA to the more complex global phenomena required to understand and interpret natural imagery. ILSA is primarily involved with the selection of models and the assignment of values to these models ("instantiation"). The models employed are of both a generic and scene- or domain-specific nature; ILSA cannot be completely divorced from final purpose.

As indicated in Chapter 8, vision in organic systems involves reasoned intelligent behavior. Where do these intellectual functions appear in the CV paradigm? Certainly not in the low-level analysis which has a largely mechanical flavor. Even though the achievements of CV are still, at best, comparable to those of fairly primitive organic systems, we will see that many of the intermediate- and high-level techniques described in this and the following sections satisfy the basic criteria for reasoning presented in Chapter 4. In particular, the representations used in intermediate- and high-level analysis, e.g., graphs and relational nets, are similar to those used in the cognitive areas of AI. The techniques are typically designed to perform an efficient search over potentially infinite solution spaces, and many of the methods have associated validation procedures which determine when an acceptable solution has been found. We will describe the following ILSA integration tasks:

1. *Image/scene partitioning*—the problem of breaking the image into coherent or meaningful units

2. *Edge linking and drawing a sketch*—organizing individual pixels that have been identified as candidate line or edge elements into contiguous segments; transforming primitive line drawings into more abstract representations of shape
3. *Recovering three-dimensional scene geometry from line drawings*—using the constraints between edges and surfaces to deduce three-dimensional scene geometry from a two-dimensional line drawing
4. *Image matching*—determining the correspondences between two images
5. *Object labeling*—assigning labels or class names to image structures
6. *Model selection and instantiation*—selecting a model with generalized parameters, and assigning values to these parameters to fit a given set of image data

Computational techniques currently used for some of these integration tasks are given in Appendixes 9-1 to 9-3.

Image/Scene Partitioning

A point-by-point description of a scene, such as that obtainable using LLSA techniques, is too complex and thus relatively useless (in that form) for most purposes. To produce a useful description, one of reasonable complexity in which higher-level scene attributes have been made explicit, the scene must be partitioned into meaningful or coherent components. How can this be accomplished without prior knowledge about the given scene?

It is conceivable that the human visual system first makes global judgments about the scene, and then decomposes this gestalt into a structured description using linguistic or visual primitives to describe localized regions in the image or scene. Except in the simplest cases, we have no notion of how to duplicate this approach in a practical manner with the computational techniques currently at our disposal.

Within the signals-to-symbols paradigm, a common approach to image partitioning is *image space clustering* (Box 9-7), the grouping of pixels based on both spatial contiguity, and homogeneity of attributes that can be measured by LLSA techniques. This approach is often implemented by requiring that regions be composed of adjacent pixels, where each pixel has an intensity value that does not differ from that of its neighbors by more than some specified amount. Other partitioning techniques, employing the same general strategy, include: (1) *feature space clustering*, grouping of pixels located anywhere in an image based on homogeneity of locally measured attributes, and (2) *boundary analysis/contouring* in which a region of an image is considered to be determined by its boundary; the partitioning algorithm links locally detected edge points into closed contours.

An image such as Fig. 9-22 (where a gestalt or overall impression is obtained from the interplay of a myriad of small, relatively meaningless intensity patches) cannot be meaningfully partitioned into regions by any of the above techniques. Isolating the small patches does not help in recovering the global aspect of the image. Further, many images do not yield a unique partitioning since the goal or purpose of the subsequent analysis can play an important role. A single image can

BOX 9-7 Partitioning via Image Space Clustering

Pixels in an image can be grouped into a common region if they have the same local characteristics, e.g., gray level or color, and satisfy a distance or connectivity criterion. Techniques for region or cluster finding can be based on the fact that good criteria for cluster separation can be defined in terms of connectivity of a graph. The first step in such approaches is to form a graph by (1) connecting each point to its *k* nearest neighbors (only pixels with some distinguished set of attributes are involved), or (2) by connecting any pair of points whose distance is less than a threshold distance. Each edge of the graph can be labeled with the distance between the two points. A typical graph is shown in Fig. 9-19.

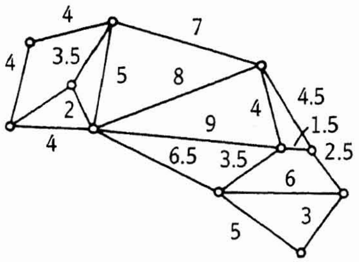


FIGURE 9-19
Typical Graph used to Cluster Pixels.

Distances are actual image distances between pixels. Each node is an image point having a desired characteristic or label.

One approach for partitioning a graph into separate clusters is to look for "cut points," nodes whose removal would disconnect the graph (e.g., see Fig. 9-20), or "bridges," edges whose removal would disconnect the graph. Another graph approach to partitioning is based on the concept of the *minimum spanning tree* (MST), the tree that connects all the nodes and whose sum of distances on edges is minimal. The general approach to partitioning data points represented by a MST is to look for edges that are long with respect to some average of lengths on both sides of the edge. For example, Fig. 9-21(a) shows a set of points formed into a graph, Fig. 9-21(b) shows one of the possible spanning trees of the graph, and Fig. 9-21(c) shows the minimum spanning tree. The long edges in the tree correspond to the gaps between the perceptually obvious clusters.

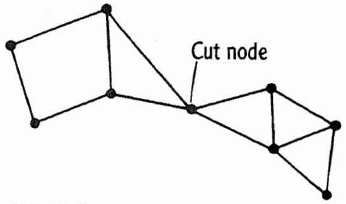


FIGURE 9-20
Use of Cut Nodes to Obtain Clusters.

Elimination of cut node (arrow) would disconnect the graph, and define two separate clusters.

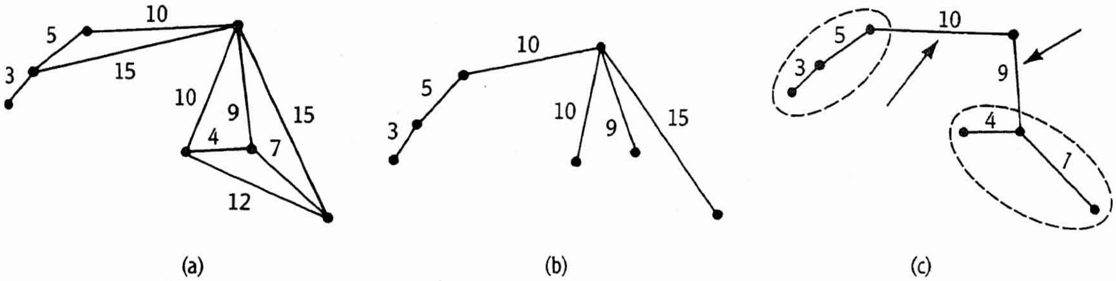


FIGURE 9-21 Minimum Spanning Tree.

(a) Original graph. (b) One of the spanning trees that connect all the nodes. (c) Minimum spanning tree of original graph; long edges (arrows) indicate possible separation of clusters.

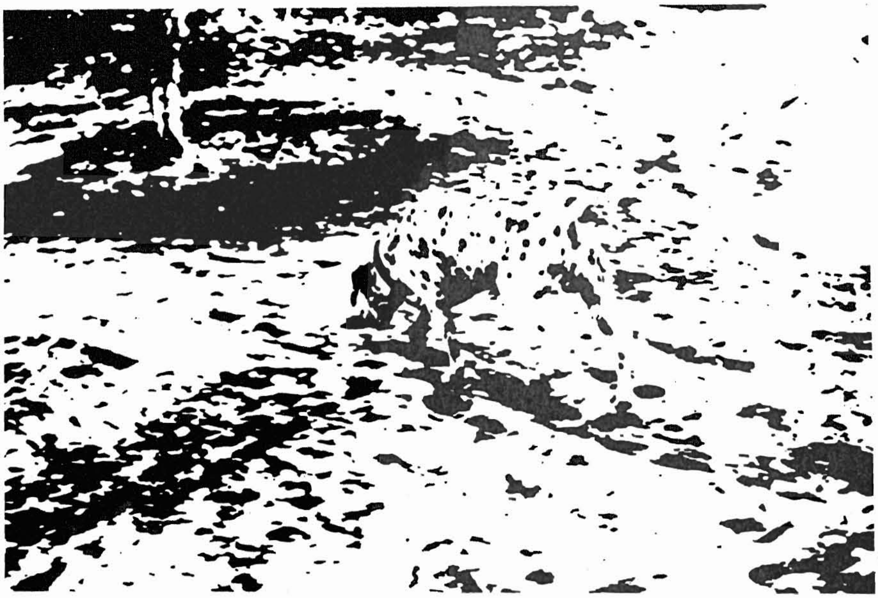


FIGURE 9-22
The "Dalmatian," an Image with a Myriad of Small, Meaningless Regions.

It is possible to find a dalmatian in the approximate center of this picture. (Photo © Ronald C. James with permission.)

also be partitioned in more than one way based on the level of detail desired, or on the point of view of the observer. Figure 9-23 shows a satellite image and the different partitionings made by experts having different disciplinary interests. Figure 9-24 shows an object that is partitioned differently by most people, depending on whether it is viewed right side up or upside-down.

In an important sense, image/scene partitioning is the creative step in visual perception that makes the rest of the descriptive process feasible. It not only decomposes the analysis problem into manageable units, but also provides the necessary structuring needed to index into a knowledge base of stored models. The partitioning techniques described above lack the competence for the type of performance required. What other approaches are possible?

Two of the issues that must be addressed by any partitioning scheme are:

1. What is the nature of the primitive vocabulary if it is not simply the set of locally measurable point attributes?
2. Without using domain-specific knowledge, how can we judge the merits of a proposed partition, or compare two alternative partitions?

There is significant evidence from psychological experiments ([Julesz 83],

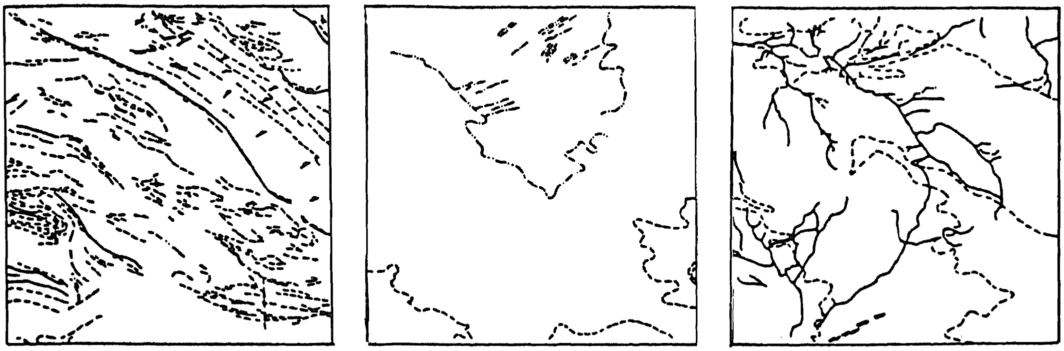


FIGURE 9-23 Different Line Drawing Interpretations of a Single Satellite Photograph Made by Experts in Different Disciplines (Geology, Forestry, and Hydrology).

(From *Ecological Surveys from Space*. NASA Office of Technological Utilization, NASA SP-230, 1970, with permission.)

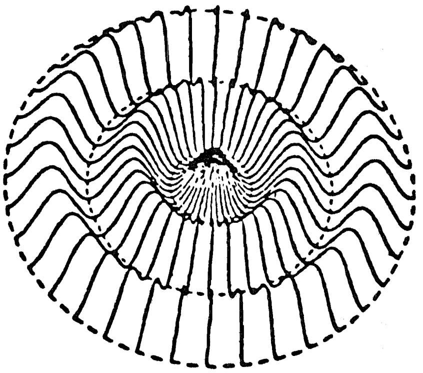


FIGURE 9-24
Different Partitionings based on Orientation of the Figure.

(After Hoffman and Richards, *Cognition* 18:65–96, 1985.) A cosine surface, which observers almost uniformly see organized into ringlike parts. A part stops and another begins roughly where the dotted circular contours are drawn. But if the figure is turned upside down the organization changes such that each dotted circular contour, which before lay between parts, now lies in the middle of a part.

[Treisman 85]) to indicate that groupings of certain primitive features can be detected almost instantaneously regardless of where they occur in the human visual field (e.g., see Fig. 9-11). Such features, called textons by Julesz and Bergen, consist of elongated blobs (especially line segments) distinguished by such properties as angular orientation, width, length, and color; ends of lines and crossings of line segments are also textons. Julesz hypothesizes the existence of a separate "preattentive" human visual system that can distinguish between different types of textons, and different densities of textons, but is unable to process information about positional relationships between different textons. Positional information essential for form perception can be extracted by a time-consuming process, called "focal attention," which is only available to the normal visual system.

There is still much debate about the nature of the primitives that are first extracted from the raw sensed data. Beyond lines or edges, and regions homogeneous in some local attribute, it is difficult to defend any higher-level construct as having sufficient utility to serve as a primitive for a general purpose vision system. It is certainly possible that most of the primitives employed (in human vision) for partitioning are not universal, but are derived for each scene domain based on some general set of principles. For example, if some pattern of points, or particularly shaped region or line segment appears often enough in a given scene, then such an entity would be a good candidate as a primitive for describing that scene, assuming it could be discovered in some reasonably efficient way.

How can we evaluate a proposed partition, or compare two competing decompositions of the same scene? If we recognize the fact that scene partitioning is an implied explanation of how the image was constructed, then in the absence of any absolute validity checking procedure or criteria, we must use various measures of believability as the basis for evaluation or comparison. A believable explanation should be complete, concise (*Occam's razor*), and stable:

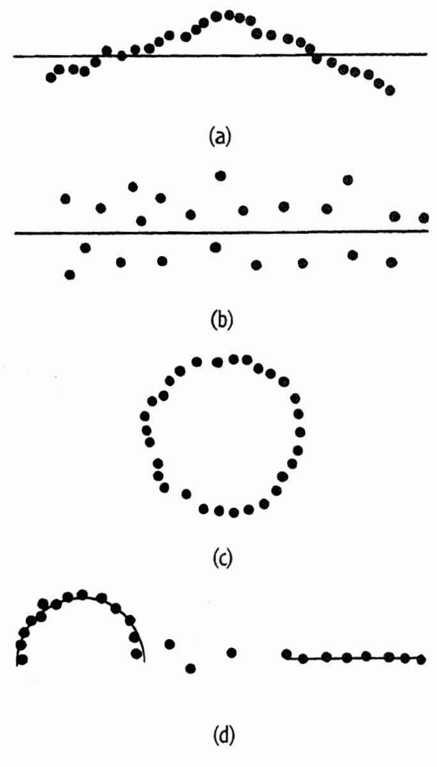


FIGURE 9-25
Criteria Underlying Effective Partitioning Decisions.

- *Completeness*. One way of measuring completeness is to require that deviations of the data from the hypothesized explanation (partition, model, etc.) have the characteristics of random noise; i.e., that all of the correlations and detectable patterns in the data are explicitly addressed in the explanation. For example, suppose we must decide if a *single* straight line is a good description of the data points shown in Fig. 9-25(a), i.e., is this data set coherent or should it be partitioned? We would tend to reject the straight-line explanation since successive deviations of data points from the hypothesized line are highly correlated. On the other hand, even though the deviations are larger in Fig. 9-25(b), a straight line is a more believable explanation here because of the random nature of the deviations.
- *Conciseness*. Conciseness or simplicity of explanation can be measured by the "length" of the explanation assuming

that the vocabulary is appropriate for the given problem domain. For example, if our vocabulary consisted only of the terms "circular arc," "straight line segment," and "image point," and we wished to construct a believable description of the object shown in Fig. 9-25(c), then the single term "circle" is a simpler explanation than a description composed of a concatenation of points or line segments. The object shown in Fig. 9-25(d) is decomposed into the shortest (simplest) description possible in terms of our given primitives (assuming a reasonable "cost" for each point, straight line segment, and circular arc).

- *Stability*. Believable explanations should be stable under slight changes of viewing conditions or of decision procedure parameters. For example, to protect against interpretation mistakes due to viewing an object from an unusual perspective, the "story should remain unchanged" when the relationship between the viewer and the object is slightly perturbed. (If you remember, the story was indeed changed in the impossible triangle of Fig. 8-12.)

A simple computational example of how the stability criterion leads to correct interpretations is illustrated in the problem of attempting to distinguish *intensity quantization boundaries* from *true* boundaries denoting actual scene content. If we shift the quantization thresholds slightly, the intensity quantization boundaries will typically shift spatially while the true boundaries will remain stationary.

Edge Linking and Deriving a Line Sketch

One of the main purposes of ILSA is to take clues about the nature of a scene,as discovered by LLSA, and compile them into more meaningful and abstract structures. The line sketch is one of the most natural and effective abstractions available for representing scene content. For example, some of the earliest expressions of human art are essentially line sketches (Fig. 9-26). The ability to depict structure with a few line strokes, however, is a creative ability only possessed by the most talented artists (Fig. 9-27). What are the computational approaches and problems

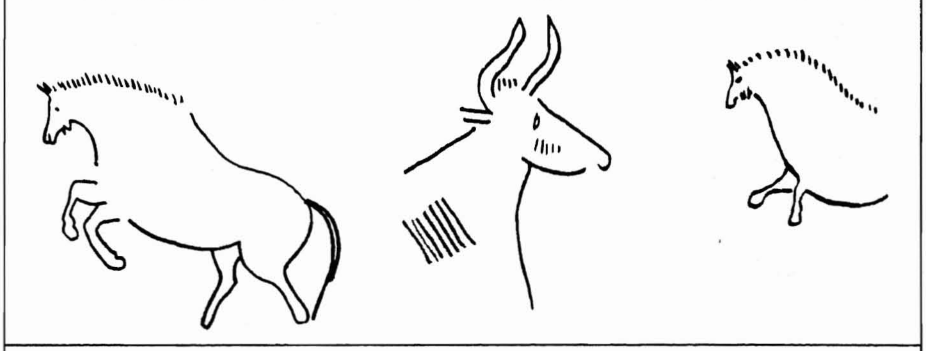


FIGURE 9-26 Early Line Sketches.

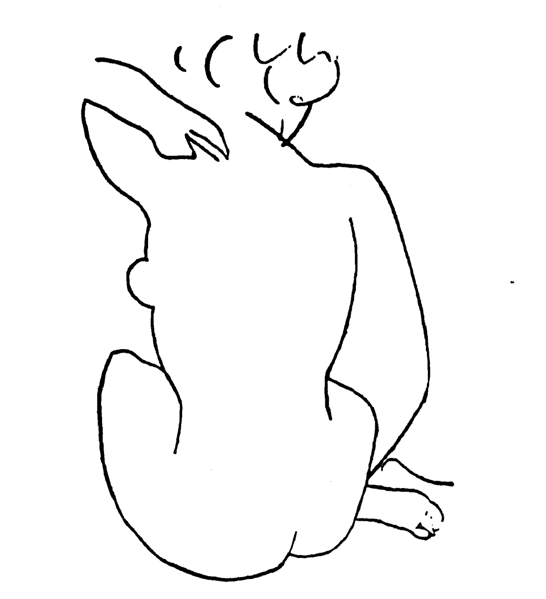


FIGURE 9-27
Sketch by Matisse that Captures the Nude Form in a Few Line Strokes.

(© Spadem, Davis/Vaga, New York 1986, with permission.)

to be dealt with in attempting to transform a gray-level image into a line sketch abstraction of the corresponding scene?

The human visual system is so good at interpreting line sketches that it is easy to overlook the fact that such sketches employ the same iconic symbol (the line symbol) to represent three completely different types of information, objective edges, subjective edges, and skeletons:

- *Objective edges.* These are the directly visible edges of regions or objects, e.g., locations in the image at which there are measurable discontinuities of intensity, color, or texture. Even if we restrict our attention to objective edges, there are usually many distinct ways to link the edge pixels found by low-level techniques. It is therefore generally necessary to appeal to some purpose or value function to provide a criterion for selecting one interpretation over others. An optimization technique for edge linking is described in Appendix 9-1. A more general solution, which is not a function of purpose or semantic constraints, is possible for simple scenes, i.e., those for which almost any observer would produce the same line sketch. The general idea is to find all the edge points in some contiguous region of the image and link these points using a *minimum spanning tree* algorithm (see Box 9-7). Long continuous paths extracted from the tree correspond to the perceptually obvious line structures in the image. This approach is illustrated in Fig. 9-28. The process involves extracting linear feature points based on local intensity characteristics (Fig. 9-28b); separating the extracted points into coherent clusters and linking the points in each cluster into a minimum spanning tree (Fig. 9-28c); pruning the tree of spurious and insignificant branches (Fig. 9-28d); and superimposing the major linear segments, as obtained above, on the original image (Fig. 9-28e).

- *Subjective edges.* These are edges known or deduced to be present in the scene, but not directly visible in the image, i.e., edges not represented by measurable local discontinuities (see Fig. 9-29). Edge linking cannot be expected to produce a complete line sketch. In many realistic situations, points associated with a particular edge will be too scattered (because of low

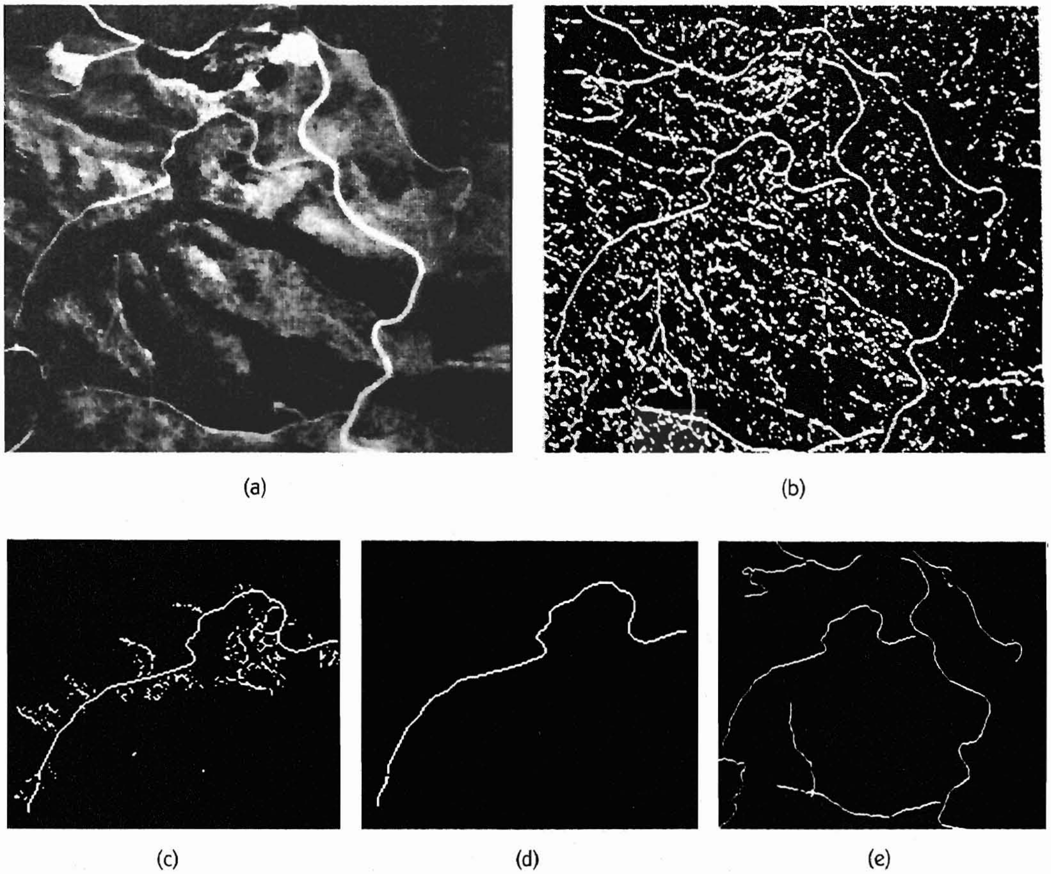


FIGURE 9-28 A Technique for Finding Perceptually Obvious Line Structures in an Image.

(a) Original image. (b) Extracted linear feature points. (c) A single cluster of feature points. (d) Linear segment extracted. (e) Line structures found in the image. (Photos courtesy of SRI International, Menlo Park, Calif.)

contrast, occlusions, and interference from adjacent but distinct edges) to be correctly linked by a simple contiguity criterion. Inferring the presence of subjective edges seems to require deduction from an assumed model, rather than inductive reasoning based on local evidence. For example, in the picture of the Dalmatian (Fig. 9-22), we must assume at some point in the analysis process that the image contains a dog, and then deduce the presence of the edges that form its outline. When there are no contextual constraints on what

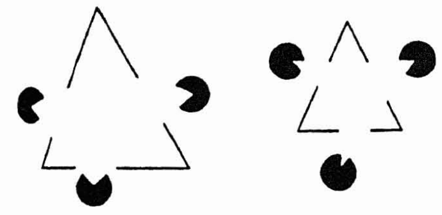


FIGURE 9-29 Examples of Subjective Edges.

(See also Fig. 8-13.)

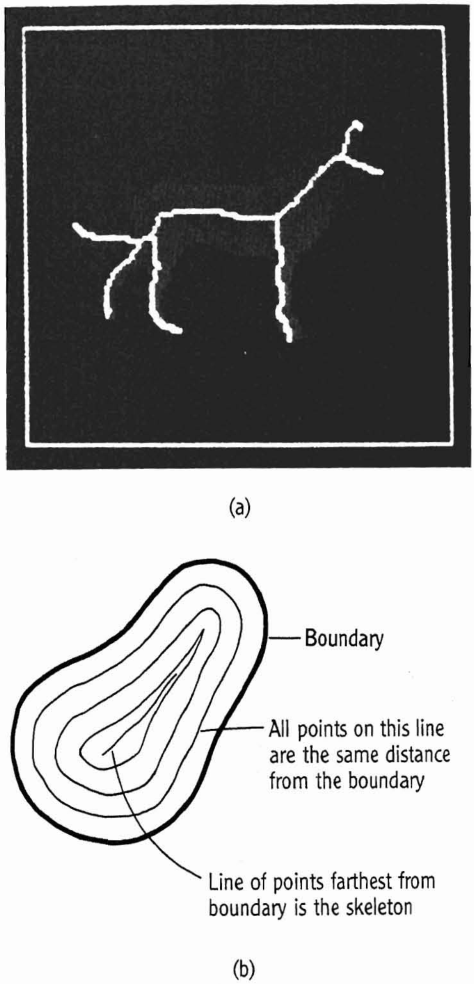

(a)
Boundary
All points on this line are the same distance from the boundary
Line of points farthest from boundary is the skeleton
(b)

FIGURE 9-30
"Stick Figures" Automatically Produced by a "Skeleton" Generating Technique.

(a) Example of stick figure derived from image of a dog. (b) Method for obtaining a skeleton by generating a nested sequence of contours. A "distance transform" algorithm for this purpose is described in [Fischler and Barrett 80].

can appear in a scene, it is difficult to comprehend how the human visual system is able to select appropriate models from an infinity of possible alternatives.

- *Abstract lines or skeletons.* These are the centerline, or spine, of long, thin objects; e.g., as in the use of stick figures to represent shapes (see Fig. 9-30). There may be locally detectable image structures corresponding to the abstract lines, but often this will not be the case. In simple scene domains, especially where the image information is essentially binary, the complete contours of isolated objects can be extracted to obtain a primitive line drawing representation of the image content. However, a human-produced sketch of the same image would almost certainly be a more abstract representation. For example, in the case of printed material, the width of the characters would be suppressed and only the skeleton would be provided. Techniques are available for extracting and using skeletons as the basis for both two- and three-dimensional shape representation.

If it is desired to represent a natural scene by a line sketch, it is generally necessary to eliminate all but a small subset of the detected and inferred edges as the final abstraction. The initial linking, insertion of abstract edges, and subsequent elimination process must be based on purpose or semantic knowledge of the scene domain and a depth of reasoning well beyond the capabilities of our current paradigm.

Recovering Three-Dimensional Scene Geometry from a Line Drawing

A person looking at a two-dimensional line sketch of a three-dimensional scene can usually partition the sketch into its coherent components and describe the corresponding scene. Given that the sketch is indeed two-dimensional, and thus an ambiguous representation of the three-dimensional world, what is the basis for this rather remarkable ability? In this subsection we will describe some computational techniques that attempt to achieve similar performance for a limited class of scenes.

It is possible to transform a gray-level image of a three-dimensional scene into a line drawing using edge analysis methods of the type discussed in preceding sections. We will assume for the present that unbroken lines representing actual edges are obtained, i.e., a perfect line drawing. We will further assume that the scene only contains objects with planar surfaces and that no more than three surfaces meet at one point in space. Given these *blocks-world* assumptions, it is possible to achieve close to human-level partitioning of the scene with relatively simple algorithms.

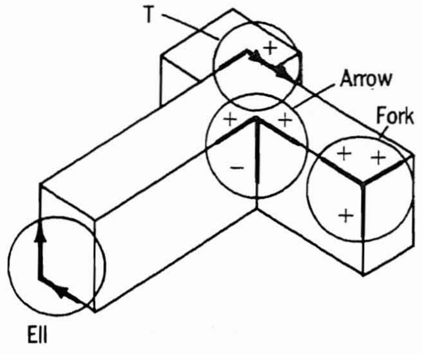


FIGURE 9-31
Three-Dimensional Shape from Line Drawings: Obtaining Junction Labels.

The simple but powerful idea is to assign one of three labels to each line in the image; these labels correspond to three types of three-dimensional edges: + (for a convex edge), − (for a concave edge), and → (for an occluding edge). Further, only four label types are needed to distinguish nodes based on the entering edge types, an ell, a fork, an arrow, and a T, as shown in Fig. 9-31. A good way to derive the complete set of physically realizable node (junction) labels is to view a simple solid figure from various viewpoints, as shown in the figure. If this approach is repeated for all possible viewing angles, the complete set of eighteen legal node labels shown in Fig. 9-32 is obtained.

This labeling scheme can be used to analyze a blocks-world line drawing. An iterative procedure is used in which we begin with all possible labels attached

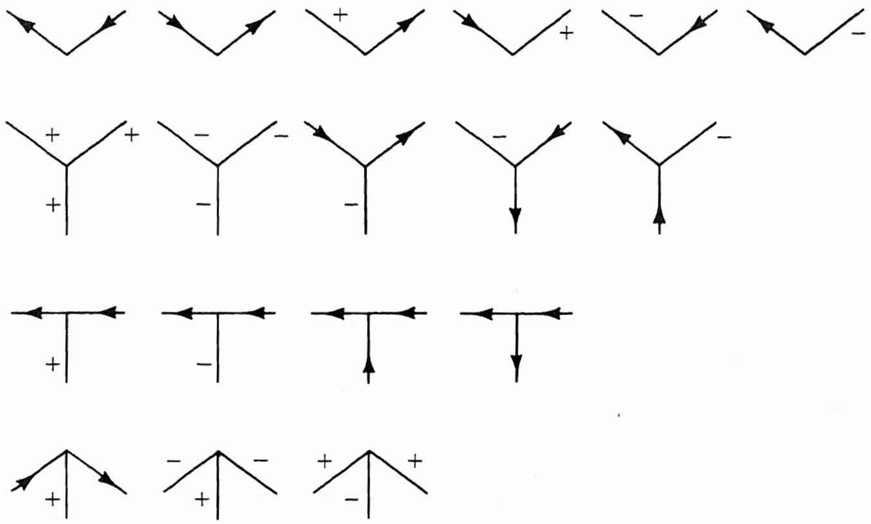


FIGURE 9-32
Three-Dimensional Shape from Line Drawings: Legal Junction Labels.

(From P. H. Winston. *Artificial Intelligence*. Addison-Wesley, Reading, Mass. 1984, with permission.)

to each edge, but by using the dictionary of legal junction types we can eliminate invalid labels. The key idea is that since an edge must have the same label at both of its end points, it imposes a constraint on the two nodes it joins. A globally consistent labeling will often produce a unique label for each edge. It is possible to obtain several different labelings for a given line drawing, but this is to be expected since, when we view a line drawing, we can often see the three-dimensional scene in more than one way, e.g., the stairway illusion (Fig. 9-33) that can be seen in two different ways.

The labeling approach does indeed determine that the object shown in Fig.

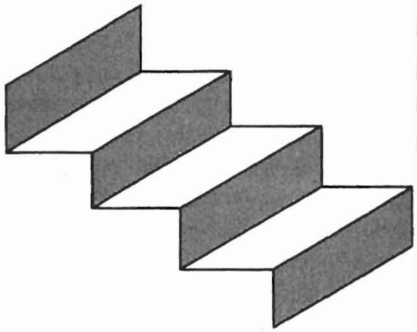


FIGURE 9-33 Stairway Illusion.

Are these stairs being viewed from above or below?

9-34 is impossible. However, other cases exist for which a depicted object cannot exist in the real world, but we are still able to find a legal set of labels. This failure has caused researchers to look for representations that can more adequately handle impossible objects, and can deal with more complex objects and imaging conditions. In the case of the blocks world, rather complex scenes—even scenes containing cracks and shadows—can be correctly analyzed, (Fig. 9-35), and necessary and sufficient conditions do exist to determine if a perfect line drawing corresponds to a physically realizable object, but these results have not yet been extended to more realistic scene domains or imperfect delineations.

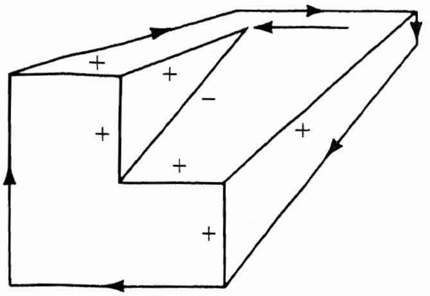


FIGURE 9-34 Impossible Figure.

An impossible object. The indicated ell junction is not among the legal ones. (From P. H. Winston. *Artificial Intelligence*. Addison-Wesley, Reading, Mass., 1984, with permission.)

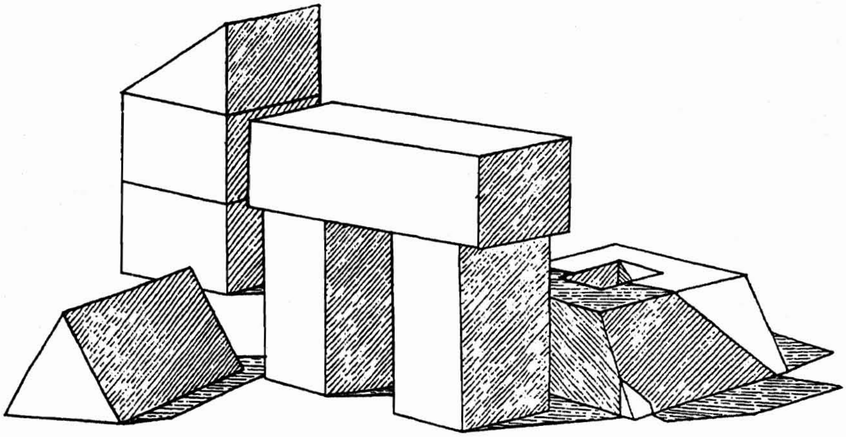


FIGURE 9-35
A Scene Typical of the Type that can be Correctly Analyzed and Partitioned by Existing Techniques.

(From D. I. Waltz. In P. H. Winston (editor), *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975, with permission.)

Image Matching

It is difficult to conceive of a more basic perceptual act than determining whether two images depict the same scene content, and more precisely, the specific or local correspondences. In the case of human stereo vision, the correspondences between the two almost identical images provided by our two eyes allows the brain to create a depth map of the scene, probably using a technique equivalent to simple trigonometric triangulation (see Box 9-5).

When the views to be matched are almost identical (say, differing only in translation and possibly scaled in intensity), then a simple computational solution to the matching/correspondence problem is area correlation, in which small fixed-size patches of the two images are compared. More generally, we are often faced with the problem of matching images that represent significantly different views of the scene. Such images may differ in three respects: (1) the viewing conditions may have changed—view angle, perspective distortion, occluded surfaces, illumination, shadows, highlights, and atmospheric conditions may be significantly different; (2) physical changes may have occurred in the scene—new or altered features such as roads, buildings, floods, or seasonal changes may be present, or objects may have moved, such as cars in a parking lot; and (3) the image acquisition and processing system may have changed—the sensors may have different noise and distortion characteristics, resolutions, spectral response, and the representations produced may be the result of different nonreversible transformations (e.g., an intensity array converted to a line sketch).

In registering one image to another, we wish to find the transformation between either the images or the sensor models, taking full advantage of the known or deduced physical characteristics of the scene and the viewing situation. If the nature of the variation of such characteristics cannot be quantified, then the variable describing these characteristics must be eliminated from the decision process. The matching process is based on the selection of the *features* to be matched, the *control strategy* that specifies how to search for potential matches, and the *criteria* for evaluating the match or selecting a best match.

Correlation approach. The traditional approach to image matching is based on signal processing and statistical decision theory concepts. Each image is treated as a (context-free) signal, with all known distortions, viewing and illumination artifacts, etc., removed prior to the matching step; the images (signals) to be matched are assumed to have some common area of overlap in which the only difference is nominally describable as additive gaussian noise. The matching technique is generally some form of *area correlation* in which the features to be matched are small fixed-size patches of the image. We search for matches by "comparing" a patch in the first image with all the patches in the second image that are potential match candidates, and we select the match that produces the highest "correlation coefficient."

Feature matching approach. As our ability to model the distortion and illumination differences between the two scenes decreases, we are motivated to extract descriptions that ignore the detailed (pixel-by-pixel) intensity variations in the images, and concentrate instead on selected measurable features (or attributes) and relations that are relatively invariant to viewing conditions and sensor variations. For example, intensity discontinuities, or edges, are more likely to remain invariant across two views of the same scene than the absolute intensity values whose differences produced the edges.

We often employ a *signal-processing* approach in determining if two images represent the same scene, by comparing the values of a set of feature measurements made on each of the images. Typical features might include area, perimeter length, spectral energy distribution, etc. The descriptions used for matching purposes in this case are the feature vectors, and the measure of similarity of the two images is the distance between the feature vectors in a Euclidean space defined by the features (see Chapter 3 and Appendix 9-1).

Relational matching approach. To establish detailed correspondences between the images, and to increase the reliability of the matching process beyond that possible using feature measurements, we must explicitly include geometric relationships between selected components of the scene in both the image descriptions and in the matching process. The image descriptions now are conceptually equivalent to a graph, where the nodes represent features or objects in the scene, and the branches represent relations; the matching process (called structural matching) involves the comparison of two graphs, or part of one graph with another.

The matching techniques discussed above (correlation, feature, and relational matching) form a natural ordering in two respects. First, there is an ordering based on the descriptive power of the representation. In area correlation the representation is the intensity array, and there is no *language* for introducing semantic information or for modeling view or sensor-related factors that cause changes in image appearance. The representation for feature matching is the feature vector, and there is no mechanism for introducing information about relations between features. However, the semantic net representation for structural matching potentially offers the full descriptive power of natural language.

The second ordering characteristic is that of modeling difficulty. As the descriptive power of the matching techniques increases, there is a corresponding requirement to define an enlarged vocabulary (names for additional features and objects) and a set of relationships between these vocabulary primitives. However, this brings with it the difficult requirement of detecting the presence or absence of these primitive linguistic constructs in an image to create the desired description.

The image-matching problem is fundamental to all of machine vision, and is therefore typical of what appears to be a pervasive difficulty facing us in all aspects of machine vision under the current paradigm: Our more powerful techniques are based on a symbolic formulation in which large amounts of low-level information are successively integrated into more global and abstract descriptions. Currently, we are limited in our ability to obtain relevant low-level information of suitable quality because our analysis depends on weak descriptive formalisms and a local perspective that is too restrictive to avoid ambiguity and error. If we attempt to use more global *primitives*, the number of such primitives necessary to provide descriptive completeness grows exponentially, and the level of modeling required for each such primitive makes such an endeavor impractical.

Object Labeling

This and the next section discuss the problems of assigning names to objects. Two forms of this problem are: (1) given a specific reference object, find instances of it in an image, and (2) label the objects in an image according to the generic classes to which they belong:

1. *Labeling specific objects*. In the simplest form of specific object labeling we assume that image shape does not differ significantly from reference shape; it is therefore easy to obtain a description suitable for matching. An example of this situation is the recognition of alphabetic characters of a given font, where the number of free parameters is limited, and we can use an attribute space in which the reference pattern and the unknown pattern appear as points. An unknown pattern is assigned to that reference pattern whose representative point is closest to it in the attribute space. However, finding suitable descriptions for complex objects becomes a significant problem. An example here is finding a specific person in a crowd. Since a person can assume various shapes when sitting, standing, or bending, a simple description will not suffice. The description must indicate the relationship between parts, the constraints in movement, the shape of parts, and the ability of the parts themselves to change shape (e.g., the shape of the mouth); we are also faced with the problem of how to structure the descriptions so that the reference objects can be compared to the descriptions derived for the sensed objects. The matching procedure must be able to deal with occlusion and flexibility of objects. Objects are often occluded by their own parts, by other objects, or by shadows, so that the derived descriptions will only partially match the reference descriptions.
2. *Labeling generic objects*. An example of generic labeling is finding all instances of a road in an image. Note that we are not looking for a known road whose description is available. Instead, a generic description of "roadness" is required. In more difficult problems, one might want to label objects with terms such as "tree," "bush," "meadow," etc. We are not asked to find a specific tree whose measurements are known. The reference object is now much more difficult to describe in terms that permit simple matching. In addition, there is a chicken and egg situation: How can we be sure that the shape of part of an unknown object corresponds to a branch, if we are not sure that the object we are attempting to identify is a tree? Most of our techniques depend on producing a description from image measurements, retrieving relevant models from our stored database (possibly) containing an immense number of such models,

and making the indicated comparison. If we do not know that we are looking at a tree in the first place, then the effort required to examine all components of all models in the database becomes exorbitant. Complete generic labeling of arbitrary scenes is well beyond the present state of the art. Labeling procedures now require that the number of generic classes be limited. One way of accomplishing this is by specifying the context of the image to the program, e.g., "outdoor scene," " office scene," etc., so that the program can select descriptions from its database that are appropriate to this context.

Model Instantiation

Visual perception is based on selecting models that are relevant to the analysis of a sensed scene, and then determining the values of the parameters of these models based on scene content (*instantiating* the model). In intermediate-level vision, typical models of interest are:

- *Geometric models*: e.g., lines, curves, polygons, planes, and surfaces
- *Illumination models*: equations that relate the light sources, surface reflectances, and image intensities
- *Sensor models*: equations that define the camera (sensor) orientation and location parameters, for a given image, in terms of a coordinate system tied to the sensed scene
- *Semantic models*: descriptions of objects and events that might appear in an image (e.g., person, building)

The model instantiation process works as follows: Once it has been determined that a particular model is appropriate for a given image, e.g., a triangle, the system could identify lines in the image that might be the sides of a triangle, find their lengths, and the angles between contiguous lines. If the model is that of a person, the appropriate instantiations could be size of the person, or male or female. In intermediate-level vision, model instantiation often results in numerical values for the model parameters.

There are three approaches to assigning values to the parameters of a model based on observed or experimental data. The classical approach is to use an optimization technique, such as *least squares*, to solve an *overconstrained* set of equations in order to define an instantiated model that best fits *all* the data, i.e., all the data is used simultaneously to solve for the parameters of the model. Problems arise when the data contain gross errors or intermixed data from multiple objects. For example, even a single measurement error, if large enough, can cause least squares to fail, and there is no general method for reliably eliminating such gross errors.

A second approach (e.g., the *Hough transform*, see Appendix 9-1) takes one data point at a time and finds all the parameters of the model consistent with this data point; i.e., we solve an *underconstrained* set of equations to find all solutions compatible with the given data point. The set of solutions determined for each data point is used to "vote" for all the corresponding parameter values. After all the data points have been processed, those parameter values receiving the most votes are taken as the desired solution.

The third approach (e.g., *random sample consensus* [Fischler 83]) randomly selects just enough data points to solve the model equations, and then attempts to confirm this instantiated model by testing it against the remaining data. If such confirmation fails, the process is repeated with another random selection of data points. This approach is surprisingly efficient, as well as robust, under a fairly wide range of reasonable conditions.

The determination of the appropriate model (discussed in the previous section), as opposed to the instantiation of the model, is a problem in detection or classification. One must utilize the evidence accumulated as the result of low level analysis to make this determination. People have a remarkable ability to select the appropriate model from what amounts to an almost infinite set of models. For example, in Fig. 9-36, how do we know that this is a picture of Paris when we have never seen the city from this particular viewpoint? In the case of computer vision, the designer is currently forced to indicate some small set of models to which the system can direct its attention.

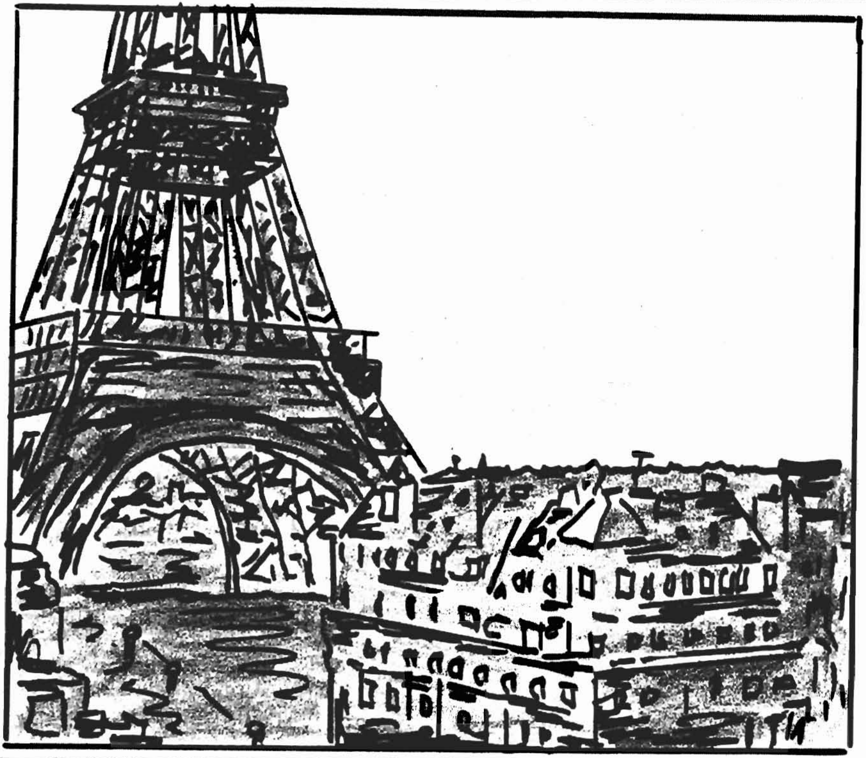


FIGURE 9-36 Recognizing a Scene: What City is Depicted Here?

(Drawing by Oscar Firschein.)

HIGH-LEVEL SCENE ANALYSIS (HLSA)

High-level scene analysis (HLSA) invokes the full body of AI techniques (e.g., symbolic logic, expert systems theory) and representations (e.g., relational nets) to provide a description of an image, or the corresponding scene, in terms of some given set of semantic models and linguistic relationships. Currently, there is little direct coupling between the information that can directly and automatically be obtained from our LLSA and ILSA techniques and the input needs of available HLSA systems. The problem here is that high-level scene analysis is strongly coupled to semantic knowledge and final purpose—a tremendous amount of knowledge is needed to bridge the gap between what is immediately visible in an image, and what can be deduced about the corresponding scene.

In trying to derive a symbolic description of a scene, one realizes that the saying, "A picture is worth a thousand words," may be too conservative. As a striking example of HLSA consider the political cartoon: The viewer is expected to recognize the participants, the topic under consideration, and the editorial view of the cartoonist, all from a simple line drawing. When we view a political cartoon from the 1800s we may no longer have the required world knowledge to understand its message.

Image/Scene Description

Having a human produce a natural language description of objects and their relationships in a scene would seem to be straightforward, and we might expect that the meaning of the natural language expression should also be readily determined. For example, the meaning of "The hat is in the box" should be derivable by having a dictionary entry for "in" that says "X is *in* Y if Y spatially includes most of X." It turns out, though, that the meaning of the word "in" is more subtle than that. Expressions such as those given below indicate quite different spatial characteristics, some of which are not captured by a simple definition of inclusion:

- The water *in* the vase (we mean the contents of the vase and not water composing the vase material)
- The crack *in* the vase (crack in the surface of the vase)
- The block *in* the circle (a block resting on a surface on which a circle is drawn
- The bird *in* the tree (a bird on a branch within the bounding region of the tree)

In addition, there are peculiarities of use, such as being able to talk about the table being in the garden, but not that the table is in the lawn. It is acceptable to draw a line in the margin, but not to draw a line in the blackboard. Some locative expressions are context dependent. Thus, given the scene:

```
        B
A
```

we would say that B is to the right of A. But if the scene changes to:

```
        B
A       C
```

we would be hesitant to make the unqualified assertion that B is to the right of A.

Natural language constructions such

as these are not merely curiosities: if we expect a robot to use descriptions prepared by nontechnical people to navigate in the real world or to carry out commands, it is important that the robot be able to decode these expressions to derive the meaning intended by the person. Thus, a person would be dismayed if the command "Get the box under the bush" resulted in the robot's digging into the ground to get under the bush, presumably to find a second, not yet visible box!

Recent studies in the semantics and pragmatics of expressions involving location [Herskovits 85], have revealed remarkable nuances of use, some of which were illustrated above.

We can gain insight into the difficulty of automatic preparation of a high-level description of a gray-level image by examining descriptions prepared by human subjects. Such descriptions depend on the background of the person, the goal of the description, and the complexity of the photograph. The descriptions can be in terms of natural language, or in the form of line drawings that extract the *essence* of the image. The importance of the background and point of view of the person preparing the description was illustrated in Fig. 9-23, where a satellite photograph was described in terms of three completely different line drawings by a geologist, a hydrologist, and a forestry expert.

An example of a natural language description of an aerial photograph, Fig. 9-37, by a layman is as follows: "The picture is an aerial photograph of a land

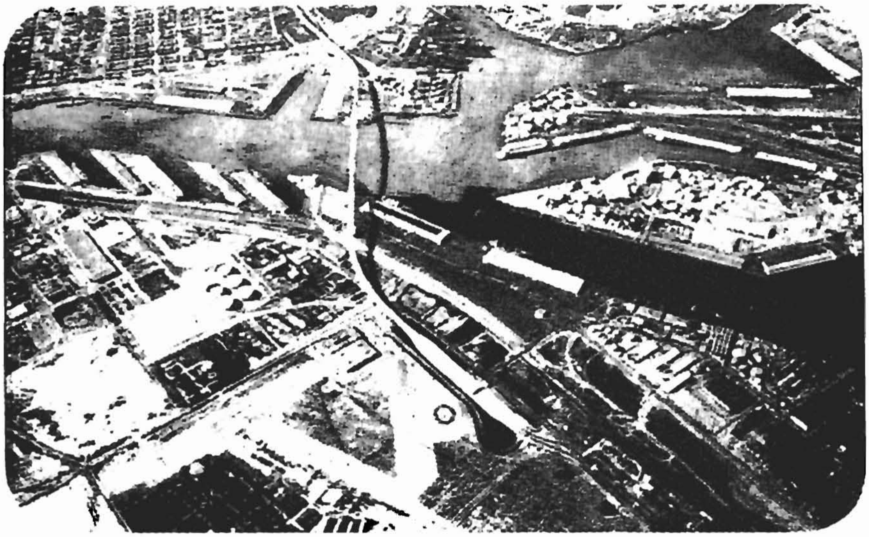


FIGURE 9-37 Aerial Photograph of an Industrial Area.

(Supplied by O. Firschein and M. A. Fischler.)

area invaded by a three-pronged fork-shaped waterway. Wharves line the sides of the waterway. A bridge, probably for auto traffic, but possibly for rail traffic, crosses the handle of the waterway. The land is used primarily for industry: many large low buildings and fluid storage tanks, such as those used to store oil or water are on the land. The area depicted has dimensions of perhaps one to two miles; the photo exhibits a significant parallax effect. The waterway, perhaps a river or canal, is perpendicular to the line of sight of the camera; its average width is about a quarter mile."

In examining this description, we note (1) probabilistic terms, "seems," "perhaps"; (2) many objects and relations between objects; (3) inferences, "probably for auto traffic"; and even (4) information about the camera viewing location. In addition to the sophisticated reasoning needed to derive such descriptions, to store them in a database for future retrieval purposes, we must represent information in a way that captures all the different aspects of entities being described, i.e., the knowledge representation problem, as discussed below.

Knowledge Representation

If the description, *A bridge crosses the waterway*, is in the database and the question is asked, *What spans the waterway?*, there must be some equivalence established between *crosses* and *spans* for a retrieval match to be made. In addition, if we have stored *A is part of B* and *B is part of C*, we need some mechanism for deducing that A is part of C. Thus, in representing knowledge about a scene, we are faced with the classic representation questions: (1) What formalism should be used for relating facts and drawing deductions from a collection of facts?; and (2) What basic relationship and descriptive words should be used in the formalism? A good image representation should have the additional important characteristic of capturing some of the implicit iconic information, e.g., objects that are near in the image should be *near* each other in the representation. Of the two representations described below, frames and semantic nets, only the semantic nets retain some of the iconic aspects of the original image.

Frame representations. In the chapter on language we discussed frames, scripts, and scenarios, an attempt to capture the components of typical situations for use in understanding natural language. Similarly, it is possible to develop frames applicable to image analysis, with each frame representing a stereotyped situation or object that might be found in a scene. The frames supply needed information and indicate what image data is relevant. For example, an *office frame* might specify what constitutes an office, e.g., desk, telephone, etc., and this frame can be used to provide the image analysis system with an expectation as to what objects might be in such a scene.

Semantic networks. The semantic network, as described in Chapter 3, has been used for HLSA. In image description applications of semantic networks, objects in the scene are represented by nodes, and the arcs from node to node represent the relations between the objects. A basic set of primitives is chosen to describe objects and relationships, and all descrip-

The picture is a black and white aerial photograph of a land area invaded by a triton-shaped or three-prong-shaped waterway. Wharves line the sides of the waterway. . . .

Concept kernels of the description fragment	Concept classes
The picture is a black and white aerial photograph	Picture property
Photograph is of a land area	Attribute
Land area is invaded by a triton-shaped, i.e., a three-prong-shaped, waterway	Operative
The waterway has sides	Set membership
The sides are lined with wharves	Attribute

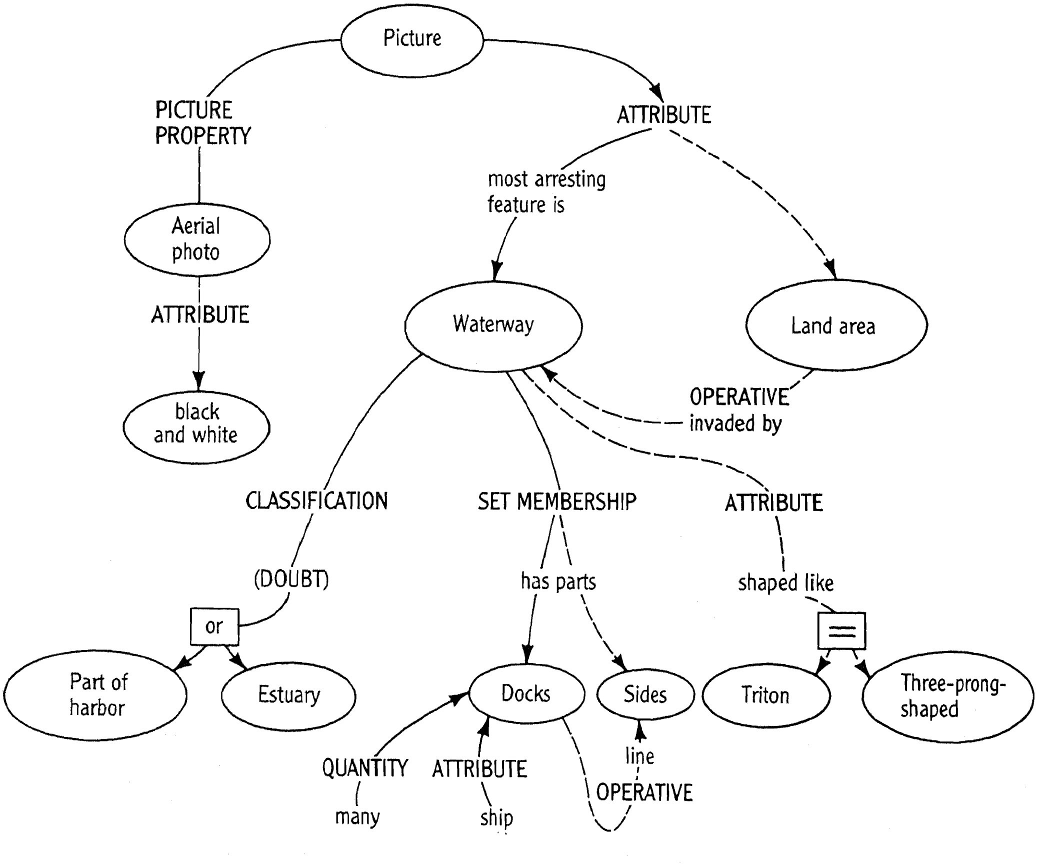


FIGURE 9-38 Semantic Network used for Image Description.

------ Denotes description fragment given above.

———— Denotes additional descriptive information about scene.

tions are converted into compositions of these semantic primitives. The number and type of primitives that form the basic vocabulary is important because the choice of primitives will determine the expressive power of the representation. A semantic network for a portion of the description of the aerial photograph (Fig. 9-37) is shown in Fig. 9-38.

Care must be taken to separate generic concepts, such as *bridges* from a specific token such as the George Washington Bridge, otherwise errors in deductions can result. For example, if we have the following network:

A→is part of→B
B→is part of→C ,

then we can follow the links to deduce that A is-part-of C. However, linking can result in incorrect deductions if the generic and specific nodes are intermingled, or if the inheritance characteristics are not carefully isolated. For example, if we have a generic description of a bridge as something that spans a road or body of water, then a specific highway bridge that is in the state of construction must not inherit the characteristic "spanning," if only the abutments have been constructed.

The semantic network representation is not a formal mathematical system with unifying principles. Its use tends to be rather *ad hoc*, with various researchers employing different net interpretation schemes based on the same general concepts.

The Problem of High-Level Scene Analysis

There is no program at the present time that can automatically create a description of a scene at a human level of performance. Further, existing programs for converting from a natural language description to a semantic network are of a rudimentary nature, and work only in very limited domains of discourse. The basic difference between describing a document and describing an image is that a textual document is usually created in accordance with some specific objective of the author. While a potential user may be more interested in a tangential fact of the document, the use cannot be too far removed from the intended theme of the document. Most images, on the other hand, have no central or organizing theme, and a description of the same object from two different points of view may be completely unrelated.

Reasoning About a Simple Scene

Given the scene shown in Fig. 9-39, we are able to reason about what has happened and what is likely to happen next: we use our knowledge of how physical objects behave in the world to deduce that the young woman has pushed the man, causing him to lose his balance, and we predict that he will fall into the well.

Little work has been carried out in obtaining programs that can reason about scenes. Funt [Funt 80] developed a program called WHISPER that reasons about simple line drawings of objects to predict the behavior of a structure constructed from blocks. The program generates calls to a low-level analysis program to determine what shapes are involved and how the shapes make contact. Some of the questions that the vision program must be able to answer are: (1) Do shapes A and B

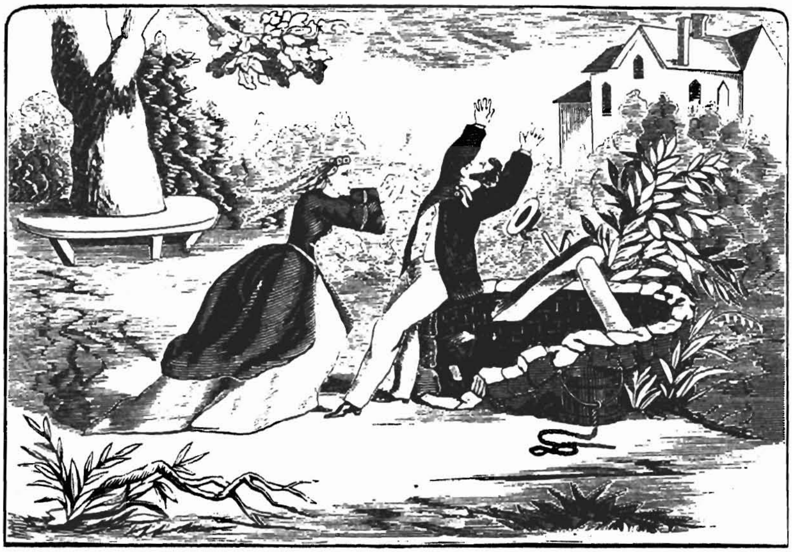


FIGURE 9-39
Example of Reasoning about a Scene: What Actions do we Expect will Follow?

(From S. Appelbaum. *Advertising Woodcuts from the 19th Century Stage*. Dover Publications, New York, 1977.)

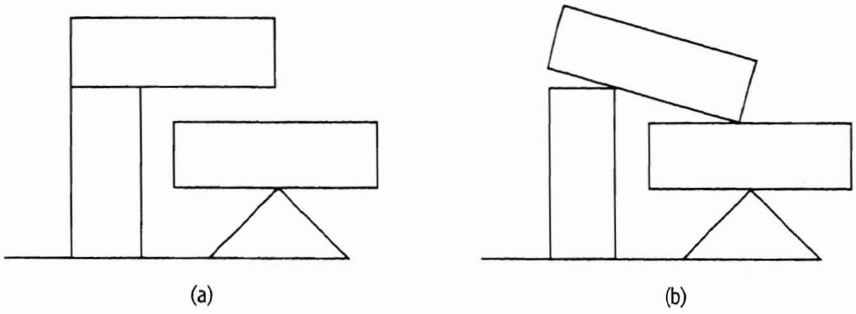
touch? (2) Is shape C symmetrical around a given axis? (3) Where is the center of the area of shape D? (4) How far can shape E rotate around a given point before it will intersect some other shape?

The high-level reasoner consists of procedures which reason about the physical world in such common-sense terms as, "If a block is hanging over too far, it will topple." To determine that a block is hanging over too far, the high-level reasoner must generate calls to the sensor, and the reasoner then assigns domain-dependent meanings to the answers returned. In the case of Fig. 9-40, WHISPER would find that the top rectangular block will fall, and will collide with the block balanced on the triangle, causing it to fall.

Note that in this program, rather than employing a strictly *data-driven* formalism, specific sensor-based observations are called for by the reasoner, and the interpretation of these observations depends on the goals and purposes of the reasoner.

DISCUSSION

We began this chapter with a description of the signals-to-symbols paradigm, and



(a) Starting state.

(b) Result of first predicted event.

FIGURE 9-40 Reasoning about a Simple Scene.

then described the various assumptions and techniques that represent the current realization of this paradigm. We now take a critical look at this entire approach. The questions we want to address here are: (1) Is signals-to-symbols an adequate paradigm? In particular, what are its weaknesses?, and (2) What are the attributes required by a machine vision system if it is to be capable of human-level performance?

A Basic Concern About Signals-to-Symbols

The signals-to-symbols paradigm was presented in a hierarchical manner: from low- through intermediate- to high-level representation and analysis. In actual practice, the processing will rarely follow such a linear route; we really do not know how to impose an effective control structure on a computational vision system that must contend with images from unconstrained scene domains. For example, very few of the low-level techniques we described would be meaningful in helping to extract the information needed to perceive the Dalmatian shown in Fig. 9-22. On the other hand, if we tried to guide our processing by guessing what was present in the image, there would be an effective infinity of possible guesses. Finally, even knowing that we want to look for a Dalmatian, it is not clear how we bypass the lack of meaningful low-level information to achieve the final perceptual gestalt.

The signals-to-symbols paradigm is the only game in town given today's digital computers which can only process numbers or symbols. However, by employing a representation without an iconic or isomorphic component, we incur the following penalties:

Limited vocabulary. We are forced to

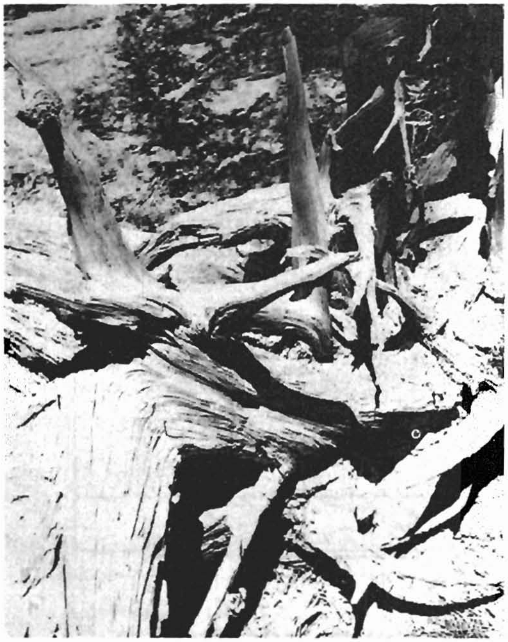


FIGURE 9-41 A Scene Difficult to Describe.

(Photo courtesy of Joseph Firschein.)

describe a scene as a network of relationships among a relatively small number of discrete named entities. Thus, we must describe a perceptually continuous scene with a description based on a limited vocabulary—the result is often a weak and inadequate description, or an unusably complex one. The driftwood shown in Fig. 9-41 is an example of a scene that is difficult to describe adequately with even the full power of natural language.

Loss of iconic representation. We lose the constraints and innate spatial relationships of the image when we go to a symbolic representation that does not have a corresponding innate spatial structuring. We have gone from an *iconic* to a *nonisomorphic* representation. In the iconic representation relationships are innate; in the nonisomorphic symbolic representation we must explicitly express relationships such as "near," "to the right of," etc.

Available iconic representations, suitable for computer implementation, are quite primitive. Typically, the image is stored as an array of numbers representing a local attribute of a scene, such as intensity. Implicit information about the shapes, relative positions, and proximity of objects is inherent in this pictorial template representation. However, the pictorial template and its currently known generalizations fall considerably short of what will probably be required for a general solution to the problem of modeling real-world vision.

There is a large volume of experimental evidence to indicate that humans use a sophisticated iconic representation in at least some of their visual tasks. For example, Roger Shepard [Shepard 71] has shown that when subjects are asked to mentally transform the spatial orientation of solid figures, they perform mental operations that are highly analogous to the transformations used to reorient the corresponding physical objects in space. Kosslyn [Kosslyn 80] has shown that the stored mental images used by human subjects appear to preserve distance: operations such as scanning took longer when the objects mentally searched for were farther apart.

Necessary Attributes of a Machine Vision System

Almost all computer vision research to date has dealt with the problem of identifying objects and describing their geometric relationships based on their appearance in an image. However, this mode of analysis represents a small subset of the reasoning employed by humans in interpreting a scene. Equally important is the ability to answer to such questions as:

Function. What are the objects in the scene doing?

Purpose. What are the objects supposed to be doing?

Competency. What are the objects able to do?

Intent. What does each object intend to do?

Anomaly. What is unusual or "wrong" with the scene?

Event analysis. What has happened?

Prediction. What is going to happen?

Evaluation. Why did the event happen?

The size and sophistication of the appropriate database, and the deductive apparatus needed to carry out the above type of analysis is far beyond what we are currently capable of doing with available techniques.

Summary

Key points that are implied in the above discussion can be summarized as follows:

Iconic representation. Many vision problems cannot be adequately described in a purely abstract formalism; this implies the need for employing some sort of iconic representation, as well as a computing device capable of supporting such a representation.

Learning. We cannot, in a practical sense, make explicit all of the knowledge needed to create a system capable of general purpose vision; some learning ability must be provided.

Need for experimentation. Because of limited understanding of the visual world, many vision problems will have to be solved by a process resembling physical experimentation; where the complexity of the problem environment prevents us from modeling it at a suitable level of detail, the *experimental space* of a device capable of general purpose vision may have to be extended out to the real world. Thus there must be an active interplay between sensing and interpretation.

Appendixes

9-1

Mathematical Techniques for Information Integration

Intermediate-level scene analysis was defined as the aggregate of operations in which local or point events are integrated into global phenomena. For example, edge points are connected to form continuous contours and semantic labels are assigned to detected scene entities. This appendix describes some of the mathematical techniques used to perform these integration operations:

- *Relaxation.* Local values or labels are adjusted to be *compatible* with neighborhood values or labels. The adjustment process continues until all values are compatible with their neighbors. The nature of the solution is determined by specified consistency and boundary conditions which remain unaltered during the computation.
- *Combinatorial optimization.* Given a set of objects, and a "cost" associated with each subset or configuration of these objects, the problem is to select a subset that satisfies a set of constraints, and at the same time minimize or maximize the value of a function of the costs. An optimal solution is selected by organizing the computation so that only the most promising alternative choices are pursued.
- *Model instantiation.* The *free* parameters of a model are assigned those values that permit the model to best describe a given set of data.
- *Statistical classification.* Objects are to be assigned to N predesignated classes. Each object is represented by a vector whose components correspond to measurements made on the object. Vectors corresponding to *ideal* measurement sets are specified for each of the N classes. Objects are classified by some function of their feature (measurement) space *distance* to the ideal measurement vectors.

Relaxation

The labeling or assignment process is a basic one in scene analysis. For example, assigning an edge strength and edge direction to a pixel is a form of labeling. On a higher level, we might label a line in an image with a code that indicates concavity or convexity of the edge it represents in the three-dimensional world. Intuitively, we know that if global information is used we can obtain a better assignment of labels than if we only use local information. A major problem is that of computational cost, since the larger the region used as the basis for establishing the labeling, the more time-consuming and complex the computation.

Relaxation techniques for scene analysis use iteration (repeated tries) as a means of obtaining a global interpretation by using only local understanding and local operations. Multiple passes are made through the image, and the labeling results are modified in each pass based on constraints, or compatibility of the current assignment of the labels at each pixel and those of its neighboring pixels. The intent is to have the local information propagate globally by means of label modification.

Forms of Label Assignment. In order to implement a relaxation process, we must have some way of making an initial assignment of labels to each pixel. For example, in edge analysis this can be done by using multiple masks, each representing a distinct edge orientation. At every pixel, those masks that produce a response greater than a certain threshold value cause a corresponding label to be assigned to that pixel. Labels can appear in two forms: (1) *discrete labeling* that does not involve probability assignments, and (2) *probabilistic labeling* in which a strength or score is assigned to the labels. An example of discrete labeling is the set of labels *vertical, horizontal, diagonal, no edge,* that can be assigned to a pixel. Probabilistic labeling would be the assignment to a pixel of probability of horizontal$=0.6$, probability of vertical$=0.2$. In the discrete labels approach, successive iterations eliminate labels that cause compatibility problems with a neighbor, while in the probabilistic case the probabilities for each label of each pixel are modified.

Relaxation with Discrete Labels. The relaxation process used with discrete labels can be best presented by an example. Suppose that we are labeling pixels in edge analysis, that the possible labels are horizontal (H), vertical (V), and none (N), and that compatibility rules require: (1) An H pixel must have H neighbors to the left or right; (2) a V pixel must have V neighbors above or below; (3) an N pixel must have N labels in three out of four of its neighbors.

Suppose we have the following situation at a pixel, P, labeled with H and N, and surrounded by four neighbors:

	V H	
H N	H N	H V
	V N	

We see that the N label of P does not satisfy compatibility requirement 3, since we cannot find three neighbors having an N label. The N label would therefore be dropped from the center pixel. (In most discrete relaxation approaches, once a label is deleted there is no mechanism for regenerating it at a later stage of computation.) The H label does satisfy condition 1 that requires H labels on either side, so the H label would be retained.

The iterative procedure terminates when there are no pixels that have more than one label or when no additional changes occur in one complete iteration. Note that it is possible to reach a situation in which all the labels at a pixel are deleted because none of the labels satisfy the required compatibility conditions.

Relaxation with Probabilistic Labels. As indicated previously, a preprocessing operation must provide the initial set of labels and their probabilities. A revised probability for each label at each pixel is typically obtained by an updating expression of the form shown in the box below.

The support for *k* around pixel i is a number between −1 and +1, with −1 indicating that the presence of label *k* is incompatible with the neighborhood labeling. This support value is a function of the *compatibility* between label *k* and the label L of each neighbor and the probability that L is a valid label for each neighbor. The normalizing factor is used to keep label probabilities between 0 and 1.

$$\text{Revised probability of label } k \text{ at pixel i} = \frac{\text{(Previous probability) (1 + support for k)}}{\text{Normalizing factor}}$$

Although appealing intuitively, this type of updating rule has no absolute justification, and its convergence properties are not generally well understood; in many examples, results first improve and then degrade if too many iterations are used.

Discussion. Relaxation is a computational mechanism that attempts a global analysis by using local consensus and iterating the procedure many times. Relaxation is attractive because the local operations can proceed in parallel, and the technique therefore has the potential for high-speed mechanization. Relaxation has also been suggested as the computational mechanism employed by biological systems [Feldman 85]. As currently employed, relaxation procedures tend to be *ad hoc*; their mathematical and semantic properties are poorly understood.

Combinatorial Optimization

Perception can be defined as finding a best interpretation of sensed data in terms of a set of *a priori* models. The term "best" often implies some sort of optimization, i.e., a selection from a set of alternatives.

Almost all optimization problems dealt with in scene analysis are either of (1) a statistical nature, e.g., a statistical measure, often in a feature space, is used to make a selection of the best set of entities, or (2) combinatorial optimization in which selection from a set of alternatives is made on the basis of maximizing or minimizing some objective function. We discuss statistical optimization below; here we describe some of the combinatorial techniques that have been employed in computational vision. The general approach is to transform the original vision problem into a problem in which a *cost* or figure of merit can be assigned to each possible combination of elements. Although the best combination could be found by evaluating the cost for every possible combination and selecting the configuration with the lowest cost, the large number of combinations makes this exhaustive approach infeasible. The optimization techniques organize the computation so that combinations that are not good candidates for solution are not considered.

Optimization Problems in ILSA. Examples of ILSA optimization problems are (1) finding the best way of linking pixels, as in edge finding, and (2) image matching.

1. **The edge-linking problem.** Suppose we have a technique that assigns a cost to each pixel in an image to indicate the likelihood of the presence of a road (or edge or line) at each pixel location. The costs are

assigned so that a low cost indicates a high likelihood of a road. Given this array of cost values, we would like to find a path through this array such that the sum of the costs along this path is minimized. In the image, this minimum cost path would then be marked as the road. Note that the resulting solution assures continuity of the global structures as well as the best collection of locally "roadlike" elements.

2. **The image-matching problem.** We are given a reference image consisting of blobs of various shapes, and a sensed image that we would like to match with this reference image. If the sensed image is a distorted version of the reference image, then we cannot use a straightforward correlation technique in which we move the sensed image over the reference image, looking for the best match. Instead, we imagine the sensed image to be on a transparent rubber sheet, so that blobs can be displaced from one another by stretching and compressing the rubber sheet. We now perform a matching operation by laying the sensed image over the reference map and stretching and compressing the rubber sheet to obtain the best possible match of the various blobs. In this matching operation we use a cost which is the weighted sum of two components: (1) a cost based on individual comparison of blobs in the sensed and reference images, and (2) a cost based on the amount of stretching or compressing required. The problem is to find the stretch and match combination that results in the smallest overall cost.

An Edge-Linking Algorithm. An approach to combinatorial edge linking is to use two arrays: (1) a local cost array that provides a measure of the edge likelihood at each pixel location, and (2) a *total path cost array*, that stores the lowest cost of the path from a starting point to each pixel. The total path cost array stores the results of each iteration, and the computation ends when no further changes can be made to this array.

We begin with a set of initial values stored in the total path cost array. All the elements are set to a very high value except that element that we would like to be start point of the path. We assign the start pixel its local cost array value. Appendix 9-2 shows an algorithm in which, for each pixel, P, we form the sum of each of its neighbor's total path cost array value plus the local cost array value of P. If the minimum of these eight sums, SMIN, is less than the current total path cost array value of P, we replace P by SMIN. The iteration is repeated until no changes occur in the total path cost array. All paths and their costs can be determined directly from the final configuration of the total path cost array.

An Image-Matching Algorithm. Suppose we have a reference image consisting of *N* components, and these components are constrained in position with respect to one another. For example, the pieces could be lines constrained to form an approximate rectangle. The image-matching algorithm must find an optimal fit of the reference figure to a structure visible in the sensed image.

An example of such an algorithm can be described using the rectangle example (see Appendix 9-3). First, each component of the reference is matched separately with the sensed image and a list is kept of acceptable match positions and the quality of the match at each position. We order the list so that the component most likely to be correctly matched appears at the top of the list, followed by the next most reliable, etc.

We now use the most reliable piece, A, and its best match locations in the sensed image, MA1, MA2, ..., and the next most reliable piece, B. Since we know the constraint relationship between A and B, we know for each of the MA1, MA2, ..., the approximate locations where the MB matches should be. We can use the list of best matches for B and determine the maximum stretch for each MB entry which satisfies the positional constraint condition relative to an MA entry.

If the combined score of the original MB match plus the stretch cost is too great, then that MB point will be eliminated from further consideration. If, for some MAn there is not at least one MBk, then we eliminate the MAn match point from further consideration. After one has formed all the acceptable

(MAn,MBk) pairs, then the next piece, C, on the list is examined for MC points that satisfy the constraints with regard to A and B components. Unmatched (MAn,MBk) pairs are eliminated and the procedure continues to D, the last piece on the list.

Discussion. The design of the cost function is crucial in the optimization approach, since it will determine the complexity of the computation and will affect the quality and characteristics of the solution. For example, in the path finding problem, adding a constant bias to each cost value tends to smooth and straighten the optimal path. This effect occurs because, as the bias increases, the length of the track becomes relatively more important in comparison to the local quality as defined by individual pixel costs. Similarly, raising each cost to a power introduces a very strong inhibition against going through a point with a high cost. Thus, the designer can introduce *a priori* knowledge (e.g., a preference for curving roads in mountainous regions, or straight roads in flat terrain) by suitable tailoring of the cost function.

Classification and Model Instantiation

This section presents some classification and model fitting techniques that have the common characteristic of using a "parameter" or "attribute space" as the underlying representation. A parameter space associates a different parameter with each coordinate axis (or possibly the same parameters from two different images). We map from the image space representation to parameter space to assign labels to individual pixels in the image, or to find those collections of pixels in an image that satisfy a model such as "house," "airport," etc. A simple example of a parameter space is the intensity histogram of an image, in which intensity is used as one axis and the other axis is the count of pixels in the image for each intensity value.

We will discuss three types of parameter space decision problems:

1. *Supervised classification,* in which the location of an "ideal" for each class in parameter space is known, and location of unclassified points in parameter space relative to the ideal points is used to make the classification assignments (statistical decision theory).
2. *Unsupervised classification,* in which point clusters in parameter space are assumed to correspond to meaningful or coherent image structures.
3. *Model instantiation,* in which point clusters in parameter space are used to find the parameters of a modeled object visible in a given image.

Supervised Classification. An example of supervised classification arises in the analysis of images acquired by earth resources satellites. A multispectral image, i.e., a set of N registered images corresponding to N different frequency bands, is obtained for some portion of the earth's surface. Each picture element then has N associated measurements. The classification problem is to assign a class label, e.g., "corn," "water," "rock," to each pixel.

If we use an N dimensional parameter space whose axes correspond to the N frequency bands, then the measurement vector at each pixel location can be mapped as a point in this measurement space. To classify the pixel, we must specify in the parameter space a set of *ideals,* points that are typical of each class. Classification then consists of assigning a pixel to the class of the *closest* ideal.

Unsupervised Classification. An example of unsupervised classification is finding a set of intensity thresholds to partition a gray-level image into coherent objects by feature space clustering. Such a procedure, based on histogram analysis, is described in Appendix 9-3.

Model Instantiation. Parameter space clustering can be used to assign values to the parameters of a model of some scene entity appearing in an image. For example, suppose we are searching for straight lines in an image. The parameters in a model for a straight line might be the slope, m, and the y intercept, b, in the relation $y=mx+b$.

We form a parameter space, (m,b), so that any line through a particular pixel will be mapped into a point in (m,b) space. If we pass a set of k lines of k different slopes through a pixel x,y in image space, we will get k different points mapped in the m,b space. If this is done for each pixel in the image that has some minimum value of *edge* *strength,* then we have the situation in which each such pixel *votes* for the m,b combination that represents possible lines passing through it. After all N edge pixels are mapped into $k \times N$ points in m,b parameter space, the point in parameter space that receives the most votes represents the best instantiation of the underlying model. Rather than using quantized histogram *buckets* to count the number of points satisfying a particular m,b combination, we could use a cluster analysis technique in a nonquantized space.

In practice, the m,b parameterization for straight lines is not suitable because m and b can become infinite. A better choice of parameters is the normal to the line , and the angle of this normal, as shown in Fig. 9-42(a). The histogram in (normal line, angle) space for lines passing through the point x_1,y_1 is shown in Fig. 9-42(b). Fig. 9-42(c) shows how a dominant line can be determined using this histogram.

Note that this approach results in a mapping of many points to the parameter space for each pixel in the image. We are essentially trying out many possible instances of the model (in this case a straight line), and relying on the clustering in parameter space to obtain the best value of the parameters for the model. This approach, called the *Hough transform*), can be extended to general shapes, as described below.

If the object we are searching for is an arbitrary shape, as shown in Fig. 9-43, we select an arbitrary point, P, interior to the region and draw the vector from this interior point to points on the boundary.

For each point on the boundary, we have two vectors, the vector R to the interior point, and the tangent vector at the boundary point. We now form an R-table containing each tangent vector and its associated R-vector. The R-table is used as follows. For each point in a given image that is a strong edge point we find the edge orientation, i.e., the tangent vector at that point. We then use this vector as a look-up entry into the R-table and find the one or more R-vectors that are associated with this tangent vector. Thus, for each edge point in the image we now have one or more R-vectors.

Now for each edge point, position the tail of its one or more R-vectors at the edge point. Note where the head of each R-vector falls and plot this R-vector-head point in a histogram whose axes have the same x and y values as the image. If the modeled object is visible in the image, then we will get a clustering of R-head points in this histogram corresponding to point P in the model. If a strong cluster does exist, then we have found the location of point P and can also locate the boundary of the object in the image.

As another example of mapping each image pixel into many histogram points, consider the following problem. Suppose we have two images taken of the same area but having different intensity characteristics due to illumination changes. Suppose, moreover, that there is some x,y displacement between the images. We would like to find both the spatial displacement and the intensity mapping function.

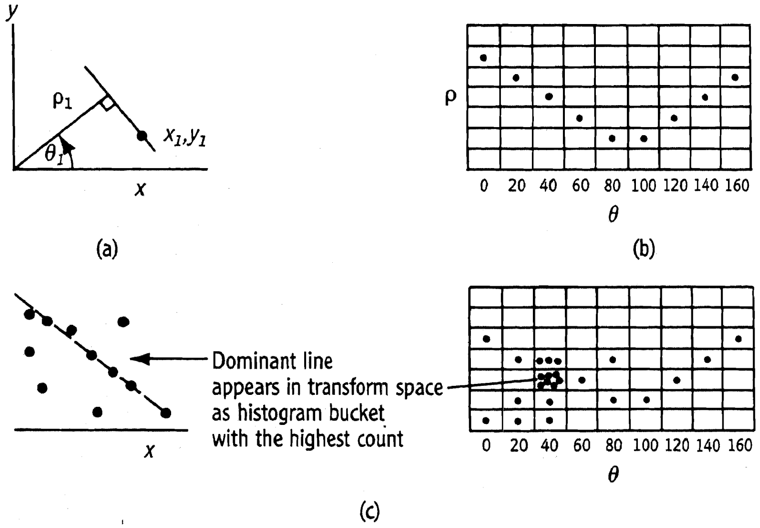
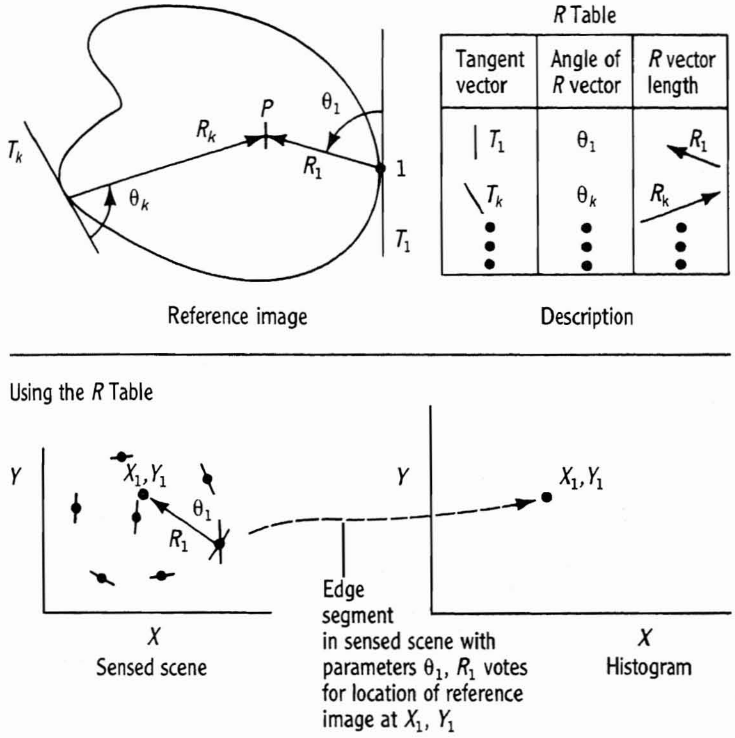


FIGURE 9-42 A Parameter Space for Straight Lines.

(a) ρ, θ parameters.
(b) ρ, θ histogram for lines passing through point x_1, y_1.
(c) Determining dominant line.



Tangent vector	Angle of *R* vector	*R* vector length
| T_1	θ_1	R_1
\\ T_k	θ_k	R_k
⋮	⋮	⋮

FIGURE 9-43 Hough Transform for General Shapes.

We can proceed by forming a histogram in which one axis represents the intensity of a pixel in image 1, and the other axis represents the intensity of a corresponding pixel in image 2, as shown in Fig. 9-44. We show two different placements of image 1 over Image 2. For each placement we obtain a histogram of the number N for each combination (intensity 1, intensity 2) that represents the number of overlapped pixels in the images that have this intensity combination. For one of the x,y offsets we find that we obtain a better (more compact) clustering in its histogram than for the other offsets. If we carry out this procedure for many different displacements of the two images, we can find a histogram that provides us with the best (most compact) mapping of intensities, and also provides us with the corresponding positional match between the images.

Statistical Classification

We have previously discussed the concept of constructing a feature space whose axes correspond to measurements made on an object. We have also indicated that when an unknown object is represented as a point in this space, there are several methods for assigning it to a class by using its distance from an ideal point of that class, or by noting where the unknown point falls in a previously partitioned feature space. In this section we will discuss how statistical theory can be used to provide the distance metric or the partitioning criteria.

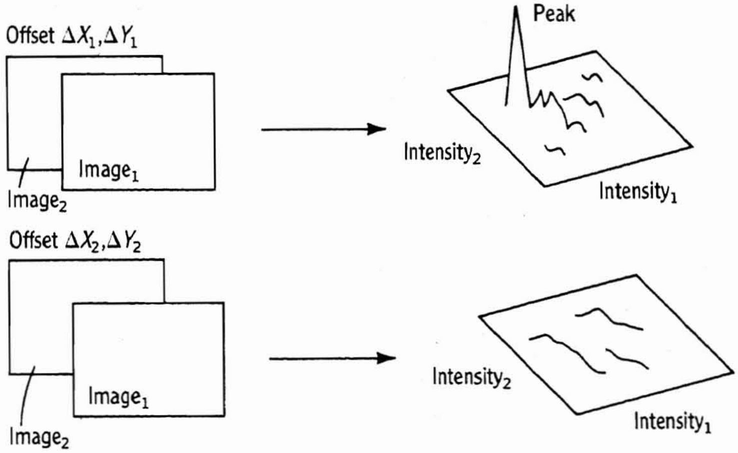


FIGURE 9-44 Image Matching using a Two-Dimensional Histogram.

The components of the statistical classification process are shown in Fig. 9-45. On the right is a *classifier* that assigns an unknown object to a class based on the classifier parameters and a set of measurements made on the object. On the left is shown the design or *training* process for obtaining the parameters of the classifier. To determine these parameters, we must know or assume the *a priori* probability of occurrence of each class, the variation in each measurement for each class (the probability density function, PDF, for each class), and the cost of misclassification (the cost of assigning an object actually of class X to class Y.)

In the discussion below, we use a single measurement and two classes for clarity, but the expressions can be extended to any number of measurements and classes.

The Conditional Probability Density Function (PDF). A typical PDF for a single measurement, *weight*, and two classes, *man* and *woman*, are shown in Fig. 9-46. We use the term *conditional PDF* and the notation p(weight measurement|woman) to denote the probability of a weight measurement, given that we are dealing with the class "woman." Note that in this example each PDF has a single peak. For most objects of practical concern, a PDF with more than a single pronounced peak indicates that the choice of measurement is not a good one.

The process of estimating the PDFs using a given set of labeled reference objects is known as *training* and will be described later.

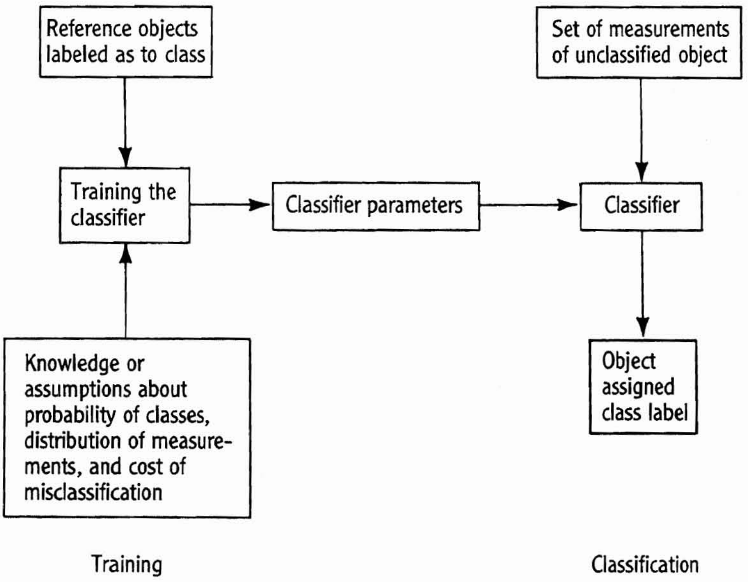

Reference objects labeled as to class
Training the classifier
Knowledge or assumptions about probability of classes, distribution of measurements, and cost of misclassification
Classifier parameters
Set of measurements of unclassified object
Classifier
Object assigned class label
Training
Classification

FIGURE 9-45
Components of the Statistical Classification Process.

Bayes's Theorem. We can use Bayes's theorem to modify the given or *a priori* probability of a class, by using the conditional PDF and the measurement obtained for the unclassified object:

$$p(C_i|m) = \frac{p(m|C_i)p(C_i)}{\text{normalizing factor}}$$

where $p(C_i|m)$ is the probability of class C_i given measurement m (called the *a posteriori* probability); $p(m|C_i)$ is the conditional PDF-supplied value, given measurement m and assuming class C_i;and $p(C_i)$ is the *a priori* probability of class C_i. The normalizing factor is used to make the set of *a posteriori* probabilities sum to unity.

Given two classes C_1 and C_2, we classify an unknown object as being a member of C_1 when the probability of class C_1 given measurement m is greater than that of class C_2, i.e., $p(C_1|m) > p(C_2|m)$. From Bayes's theorem this becomes, choose C_1 when $p(m|C_1)p(C_1) > p(m|C_2)p(C_2)$, otherwise choose C_2. If the PDFs are given as histograms derived from a reference set of measurements made on *ideal* objects representing each class, we would scale each histogram based on the *a priori* probability of its class, and assign the unknown object to the class that had the largest resulting value for measurement(s) m.

We note that since $p(C_1)$ and $p(C_2)$ are constants, and the $p(m|C_i)$ are single peaked functions for the example shown in Fig. 9-46, the decision criteria in this case amount to assigning all persons with mea-

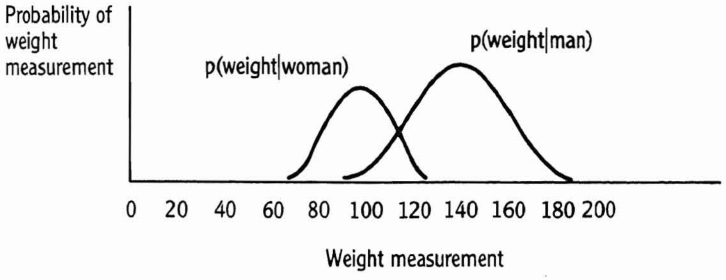


FIGURE 9-46
Conditional PDFs of Weight Measurements for Two Classes.

sured values of *m* less than some value, M*, to the class *woman* and all persons with values of m greater than M* to the class *man*. Thus, the decision criteria based on statistical arguments can be interpreted as a simple partitioning of feature space into two regions separated by a simple boundary. For a multidimensional feature space the partitioning is performed using multidimensional planes (hyperplanes) or multidimensional surfaces (hypersurfaces). An alternative interpretation is to consider the statistical analysis as altering the initial Euclidean distance metric in such a way that any point can be assigned to the class corresponding to the nearest ideal point.

Training. The heart of the statistical classification approach is the conditional PDF. Either *parametric* or *nonparametric* classifiers can be designed, based on what is known about the PDF and what must be estimated. A parametric classifier assumes a functional form of a PDF based on some knowledge about the objects being classified. Statistical procedures are used on the reference set to estimate parameters such as the mean and variance of the assumed PDF. In a nonparametric classifier we do not know enough about the objects to assume a PDF and therefore the PDF form is estimated from the reference data. This requires much more reference data than the parametric case. The process of estimating the PDFs using the reference objects is often referred to as *training*, and if the reference objects have been labeled (classified) by the designer, the term *supervised training* is used.

9-2

A Path-Finding Algorithm

This appendix presents a dynamic programming approach to finding the lowest cost path in a cost array. The example array is given in Fig. 9-47(a) and the numbers represent the local cost of having the path go through that element. In this example, we want to traverse the array from the lower left (start element) to the upper right (end element) so that the sum of the path elements will be minimum.

The algorithm uses a "path cost array" that specifies the total cost of the path from the starting point to each element of the array. We begin the process by initializing the path cost array with high values for all elements except that element we wish to make the start of the path; this element is assigned its local cost array value. In our example we want the start of the path to be in the lower left hand corner, so we assign that element the value 3, its value in the local cost array. All other elements are assigned the suitably large arbitrary value 100, as shown in Fig. 9-47(b).

A revised path cost array value is obtained by adding the local value of an element in L and the path cost array value of one of its neighbors;

```
                                                  9 8 2 2 1
                                                  8 3 7 8 8
                                                  7 2 6 3 8
(a)                                               7 8 3 8 2
                                                  9 8 7 6 2
                                                  3 2 2 2 8

                                         100 100 100 100 100
                                         100 100 100 100 100
                                         100 100 100 100 100
(b)                                      100 100 100 100 100
                                         100 100 100 100 100
                                           3 100 100 100 100

        100 100 100 100 100              100 100 100 100 100
        100 100 100 100 100              100 100 100 100 100
(c)     100 100 100 100 100              100 100 100 100 100
        100 100 100 100 100               18  19  14  22  24
         12  11  18  24  26               12  11  12  13  11
          3   5   7   9  17                3   5   7   9  17

        100 100 100 100 100              100 100 100 100 100
        100 100 100 100 100               24  19  23  25  25
(d)      25  16  20  17  25               23  16  20  16  21
         18  19  14  19  13               18  19  14  19  13
         12  11  12  13  11               12  11  12  13  11
          3   5   7   9  17                3   5   7   9  17

                                          28  27 ,21–23–24
                                          24  19´ 23  24  24
(e)                                       23  1ˈ6 20  16  21
                                          18  19 ˋ14  19  13
                                          12 ,11´ 12  13  11
                                           3´ 5    7   9  17

         81  64   4   4   1              100 100 ,58–62–63
         64   9  49  64  64              100  54´ 81  96  96
(f)      49   4  36   9  64               94  4ˈ5ˍ 68 ,32ˍ 87
         49  64   9  64   4               94 100 ˋ41´ 83 ˋ23
         81  64  49  36   4               84  67  56  47  1ˈ9
          9   4   4   4  64                3 – 7–11–15´  79
```

FIGURE 9-47 Arrays used in the Path Computation.

(a) Local cost array (L). (b) Initial path cost array (P). (c) Results of first (left) and second (right) iterations. (d) Third (left) and fourth (right) iterations. (e) Fifth iteration, showing solution path. (f) Effect of squaring the local cost array on solution path: Squared local cost array (left) and solution path cost array (right).

we perform this computation for each of its eight neighbors, and if one of these sums, SMIN, is less than the element's current path cost, T, we replace T by SMIN (in P). We show in Fig. 9-47(c) the result of performing this operation while sweeping through each row from left to right, and processing rows from bottom to top.

Notice in the first iteration how change spreads from the lower left corner. The repeated bottom-to-top sweeping rule we are using here is not an efficient one, since change is generated at the bottom of the array; a better strategy would have been to start at the bottom, but to reverse the sweep direction at each additional iteration. Continuing the iterations we obtain the array shown in Fig. 9-47(d). Notice that no changes are occurring in the bottom rows. The final result is obtained on the fifth iteration, Fig. 9-47(e).

To obtain the minimal cost path, we start with the final element of the path (the end element), and look for the neighboring element having the lowest value. Thus we thread through the array with the path 24–23–21–19–16–14–11–3. To illustrate the relevance of the *cost* assignment process, note what happens when the computation is repeated, but with the local cost array squared, i.e., a monotonic transformation of the cost function. Since the differential between low and high costs has become more extreme, it now pays to lengthen the path to avoid high-cost pixels, and we obtain a longer and more "wiggly" path, as shown in Fig. 9-47(f).

The path is now 63–62–58–54–45–41–32–23–19–15–11–7–3.

9-3

Relational (Rubber Sheet) Image Matching

This appendix shows how dynamic programming can be used to solve a problem in image matching. We have a reference image

6 4

3 5

and we want to find the best match in the sensed image:

5 2 8 8

7 5 1 3

8 1 5 7

4 3 2 4

If the reference image was "rigid," then we would move the sensed image over the reference image looking for the location where the sum of the absolute differences between the sensed and reference elements was smallest. (The absolute difference of corresponding pixel values is the local cost function chosen for this example.) In the present situation, we are willing to allow a certain amount of "give" between the reference elements; however, a penalty will be paid for such stretching. For reference purposes, we will label the reference image

A B

D C

We will assume that springs exist between pairs (B,A),(C,B), and (D,C) , and that there are the following costs to stretching:

- Element A has no spring cost
- Element B to element A: B adjacent to right of A costs nothing
  B displaced one unit to the right costs 1
  Other movements of B from A cost infinity
- Element C to element B: C adjacent below B costs nothing
  C displaced one unit down from B costs 1
  Other movements of C from B cost infinity
- Element D to element C: D adjacent to left of C costs nothing
  D displaced one unit to left costs 1
  Other movements of D costs infinity

For convenience, we display below the cost of match for each reference element. Each array was obtained by taking the absolute difference of a reference element with each element of the sensed array.

The arrays on page 300 show the results of the computational sequence. The leftmost arrays show the placement of element A. Looking at A's cost matrix, we choose the lowest cost positions, those having a cost of 1. For clarity we have drawn a separate array for each best location of A. For each best placement of A we then consider the best placements of B. For example, B can be placed adjacent to the first placement of A for a cost of 1+2+0 units, or B can be stretched for a cost of 1+4+1. The first term is A's best cost, the next term is B's match cost, and the third term is the stretch cost. Once A and B have been placed, wę have a total cost for the A,B combination. In our example, we choose the lowest cost in each array to determine the positioning of the corresponding component. In actual practice we could carry along more than one position for each component.

In the placement of element C on the first row of arrays, given the best placement of A and B costing 3 units, we can place C for 3+0+0 or 3+4+1 units, as shown. The placement that results in 3 units of cost is chosen, and is used in the rightmost column to obtain a placement of D at a cost of 3+4+0 units.

A match cost	B match cost	C match cost	D match cost
1 4 2 2	1 2 4 4	0 3 3 3	2 1 5 5
1 1 5 3	3 1 3 1	2 0 4 2	4 2 2 0
2 5 1 1	4 3 1 3	3 4 0 2	5 2 2 4
2 3 4 2	0 1 2 0	1 2 3 1	1 0 1 1

Cost Arrays

```
Placement A     Placement A,B     Placement A,B,C     Placement A,B,C,D
  1 - - -         A 3 6 -           A B - -             A B - -
  - - - -         - - - -           - 3 - -             7 C - -
  - - - -         - - - -           - 8 - -             - - - -
  - - - -         - - - -           - - - -             - - - -

  - - - -         - - - -           - - - -             - - - -
  1 - - -         A 2 5 -           A B - -             A B - -
  - - - -         - - - -           - 6 - -             - - - -
  - - - -         - - - -           - 5 - -             6 C - -

  - - - -         - - - -           - - - -             - - - -
  - 1 - -         - A 4 3           - A - B             - A - B
  - - - -         - - - -           - - - 5             - 8 7 C
  - - - -         - - - -           - - - 5             - 6 6 C

  - - - -         - - - -           - - - -             - - - -
  - - - -         - - - -           - - - -             - - - -
  - - 1 -         - - A 4           - - A B             - - A B
  - - - -         - - - -           - - - 5             - 6 6 C
```

Thus, the lowest cost placements have a value of 6 and they are the following:

```
- - - -     - - - -     - - - -     - - - -     - - - -
A B - -     - A - B     - A - B     - - - -     - - - -
- - - -     - - - -     - - - -     - - A B     - - A B
D C - -     - D - C     - - D C     - D - C     - - D C
```

In actual practice the computations would be in tabular form. Note that if we add the additional constraint that D must be in the same column as A, then only three solutions are valid.